ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Implementación en FPGA de un algoritmo de compresión de señales EEG multicanal

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Federico Favaro

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Director de Tesis

Juan Pablo Oliver ..... Universidad de la República

Tribunal

Sebastián Fernández ..... Universidad de la República
Álvaro Martín ..... Universidad de la República
Julio Pérez ..... Universidad de la República

Director Académico

Juan Pablo Oliver ..... Universidad de la República

Montevideo
miércoles 4 de setiembre, 2019

Implementación en FPGA de un algoritmo de compresión de señales EEG multicanal, Federico Favaro.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 80 páginas.
Compilada el miércoles 28 octubre, 2020.
http://iie.fing.edu.uy/

Resumen

Los dispositivos lógicos programables ofrecen gran flexibilidad y rendimiento, características que les han hecho ganarse un lugar en diversas aplicaciones, tales como procesamiento de video y audio, infraestructura de redes y aceleración de algoritmos. Una de sus principales ventajas es que proveen la capacidad de procesamiento y el paralelismo típicos del hardware, pero una reprogramabiliadad que les permite adaptarse a los vertiginosos cambios tecnológicos a muy bajos costos.

Las FPGAs basadas en RAM estática, tecnología que domina el mercado de las FPGAs, poseen las características mencionadas anteriormente. Sin embargo, esto es posible a costa de un aumento en el área de silicio, un alto consumo estático y prácticamente nulas capacidades de operación en modos de bajo consumo. Estas cuestiones, entre otras, han mantenido a las FPGAs alejadas de las aplicaciones de bajo consumo. No obstante, recientemente han surgido nuevos dispositivos orientados a este tipo de aplicaciones y las FPGAs han ganado terreno, por ejemplo, en sistemas portátiles alimentados a batería.

En esta tesis se propone estudiar el problema de utilizar FPGAs en aplicaciones de bajo consumo, pero que a la vez tienen fuertes requerimientos de cómputo y manejan grandes tasas de datos, lo que dificultaría su solución mediante plataformas basadas en microcontroladores. Se aborda el problema desde el desarrollo de un electroencefalógrafo inalámbrico portátil alimentado a batería, aplicación que posee las mencionadas características. En particular, el trabajo se centra en la implementación en hardware de un algoritmo de compresión de señales neurales multicanal, como primer paso en el desarrollo de un sistema inalámbrico de adquisición y transmisión en tiempo real de señales de electroencefalograma (EEG).

Los sistemas de EEG inalámbricos requieren la adquisición, almacenamiento y transmisión de señales biomédicas provenientes de múltiples canales. Esto puede generar grandes volúmenes de datos, especialmente cuando se procesan decenas de canales y las frecuencias de muestreo rondan los kilo–hertz. En esta clase de aplicaciones la compresión de datos juega un rol fundamental, ya que permite disminuir los requerimientos de almacenamiento y transmisión, que a su vez redunda en menor consumo energético.

Para obtener un punto de comparación, en una primera etapa se implementa el algoritmo en una plataforma basada en el microcontrolador de bajo consumo MSP432. Luego se implementa el algoritmo en hardware y se evalúa su performance en tres FPGAs diferentes: Cyclone V 5CEBA4, iCE40HX y MachXO2. El diseño fue caracterizado para 21, 31 y 59 canales y distintas frecuencias de operación,

utilizando muestras reales de EEG provenientes de bases de datos públicas. La verificación se llevó a cabo mediante simulaciones y pruebas reales en las FPGAs.

En todos los casos las FPGAs logran importantes mejoras en la velocidad de compresión. La MachXO2 es entre 11 y 12 veces más rápida que la implementación en microcontrolador, alcanzando frecuencias de muestreo en el entorno de 13 kSps para 59 canales. La iCE40HX permite utilizar frecuencias de muestreo alrededor de 7 kSps, y logra incrementos respecto al MSP432 de aproximadamente 6×. La Cyclone V 5CEBA4 es aún más rápida que las otras dos, pero debido al alto consumo de potencia no resulta adecuada para la aplicación.

En todos los casos se midió el consumo del *Core* de las FPGAs durante la compresión. En la iCE40HX se relevó un consumo entre 3 mW y 5 mW, valores similares a los obtenidos en el microcontrolador. En base a los resultados de performance y consumo obtenidos para esta plataforma, se puede concluir que es adecuada para su utilización en sistemas de bajo consumo que manejen altas tasas de datos, como es el caso de un EEG inalámbrico. Los resultados de potencia en la MachXO2 oscilan entre 21 mW y 38 mW, números relativamente altos considerando la aplicación. Sin embargo, esta FPGA posee un modo de *stand–by* que le permite operar en ciclos de trabajo y lograr reducciones en consumo que la acercan a los niveles del microcontrolador.

Tabla de contenidos

Tabla de contenidos

Capítulo 1

Introducción

La electroencefalografía consiste en el registro de la actividad eléctrica del cerebro proveniente de la interacción entre las neuronas. A lo largo de la historia, esta ha jugado un rol fundamental en el entendimiento del cerebro humano. Sin embargo, los sistemas tradicionales de electroencefalograma (EEG) son cableados y reducen considerablemente la movilidad del paciente, causando incomodidad y dificultando la realización de estudios de larga duración. Por otro lado, los avances tecnológicos de los últimos años han dado lugar a que complejos sistemas de monitoreo sean lo suficientemente compactos para ser vestidos[1] por las personas. Esto permite, no solo la realización de nuevos y mejores diagnósticos médicos [1], sino que habilita al paciente a continuar con las actividades de su vida cotidiana.

La electroencefalografía inalámbrica tiene múltiples aplicaciones en medicina clínica e investigación neurocientífica. Por ejemplo, tanto el diagnóstico de epilepsia como la detección de los ataques epilépticos siempre han sido un desafío [2]. Sistemas de EEG vestibles permiten monitorear al paciente durante horas o incluso días, aumentando enormemente las probabilidades de predecir episodios epilépticos y notificar al paciente o incluso administrar algún tipo de tratamiento para prevenirlos [3].

Otro campo de aplicación donde existen una inmensidad de oportunidades es el de las interfaces cerebro-máquina. Estas tienen múltiples aplicaciones relacionadas con la asistencia a personas minusválidas, que van desde el manejo de un cursor en una computadora hasta el control de un brazo robótico [4] o incluso una silla de ruedas [5]. Pero también tiene sus usos fuera de la medicina, por ejemplo en la cognición aumentada [6] o en entretenimiento [7].

Tradicionalmente, los estudios clínicos de EEG requieren anchos de banda entre 0.5 y 50 Hz [1]. Pero existen casos en los que es necesario aumentar el ancho de banda analizado, en los que las frecuencias pueden llegar hasta 600 Hz [8]. Por ejemplo, en estudios recientes sobre el diagnóstico y predicción de ataques epilépticos mediante el análisis de oscilaciones de alta frecuencia, se sugiere que

[1] A lo largo de esta tesis se empleará el término *vestibles* (del inglés, *wearables*) para referirse a los dispositivos electrónicos inteligentes que se incorporan en alguna parte del cuerpo.

puede haber información útil a frecuencias superiores a 1 kHz [9].

Por otro lado, la miniaturización y el perfeccionamiento de las interfaces analógicas y los conversores analógico–digitales hacen posible la adquisición de cientos de canales en simultáneo [10]. La multiplicidad de canales en combinación con las altas frecuencias de muestreo imponen tasas de datos difíciles de manejar en dispositivos vestibles, ya que, por sus características, poseen limitada capacidad de cómputo, escasa memoria y requieren bajo consumo de energía.

A modo de ejemplo, un sistema de adquisición con 64 canales, 16 bits por canal y una frecuencia de muestreo de 2 kHz resulta en una tasa de datos de 2 Mbps. Este valor está por encima del límite de lo que la mayoría de los protocolos inalámbricos de bajo consumo pueden manejar. Por ejemplo, Bluetooth Low Energy 5 alcanza un máximo teórico de 2 Mbps, que debido al *overhead* necesariamente va a ser menor a nivel de aplicación [11].

Es deseable entonces que dispositivos portables de estas características hagan un manejo eficiente de los datos. Es aquí donde la compresión juega un rol fundamental, por lo que siempre ha sido una de las alternativas preferidas para disminuir el flujo de datos. La compresión tiene dos grandes ventajas: por un lado, disminuye el consumo de energía al reducir el tiempo de utilización del transmisor inalámbrico y, por otro, reduce los requerimientos de ancho de banda del transmisor, permitiendo utilizar tecnologías de menor complejidad [12]. Sin embargo, algoritmos de compresión eficientes y de baja latencia imponen una fuerte carga de procesamiento en el dispositivo de bajo consumo. Por lo tanto, existe un compromiso entre el ahorro de energía en la transmisión inalámbrica y el gasto extra por incrementar el procesamiento de datos.

Esta tesis tiene como punto de partida el proyecto CSIC[2] I+D "Electroencefalógrafo inalámbrico de bajo consumo de energía", donde se investigó el ahorro de energía que puede obtenerse en electroencefalógrafos inalámbricos a través del uso de esquemas de codificación eficientes (compresión). En dicho proyecto participaron docentes del Departamento de Electrónica y del Núcleo de Teoría de la Información, pertenecientes al Instituto de Ingeniería Eléctrica y el Instituto de Computación de la Facultad de Ingeniería, Universidad de la República.

Como primer resultado del proyecto, los autores en [13] reportaron un algoritmo de compresión eficiente de señales biomédicas multicanal. El algoritmo desarrollado es de baja complejidad (la complejidad crece linealmente en almacenamiento y número de operaciones con la cantidad de canales), baja latencia y sin pérdidas. Al momento de la publicación, la tasa de compresión obtenida era la mejor reportada en la literatura. Por sus características, el algoritmo resulta ideal para su utilización en sistemas de adquisición en tiempo real de señales de EEG.

En un esfuerzo por reducir aún más la complejidad algorítmica, los autores desarrollan dos nuevas versiones aplicando modificaciones no triviales a [13], con el objetivo de posibilitar su implementación en un microcontrolador de bajo consumo, con reducidas capacidades de cómputo y memoria. Las nuevas versiones del algoritmo y los resultados de su implementación en la plataforma de bajo consumo

[2] Comisión Sectorial de Investigación Científica de la Universidad de la República, Uruguay.

fueron publicados en [14] y [15].

La participación del autor de esta tesis en el proyecto mencionado fue la adaptación de las nuevas versiones del algoritmo para su implementación en el microcontrolador de bajo consumo. También realizó la caracterización del sistema en términos de performance (*throughput*) y consumo energético. Habiendo obtenido prometedores resultados, y siendo el algoritmo relativamente simple en términos de operaciones aritméticas, surgió la motivación de llevar a cabo una implementación en hardware del mismo. Lo anterior implica diseñar un circuito lógico, utilizando lenguajes de descripción de hardware (HDL[3]), que desempeñe la función del algoritmo de compresión. Un diseño de este tipo puede ser luego sintetizado en un Circuito Integrado de Aplicación Específica (ASIC, por sus siglas en inglés) o en una FPGA. La interrogante, que sirvió como punto de partida para este trabajo, es si una implementación del algoritmo en una FPGA puede lograr sustanciales mejoras en la performance, manteniendo la capacidad de compresión y sin aumentar el consumo de energía.

Las principales características del algoritmo que favorecen la implementación en hardware son: la simpleza de las operaciones aritméticas (que consisten en sumas y desplazamientos de números enteros), pocos bucles y de reducidas iteraciones (que permiten una baja latencia) y un reducido tamaño en las ventanas de datos a procesar (que también impacta en la latencia).

En resumen, esta tesis se centra en la implementación en hardware del algoritmo de compresión de señales de EEG descrito en [15]. Además de proveer una implementación funcional del algoritmo validada en hardware real (FPGA), un principal objetivo de este trabajo es obtener reglas generales sobre la utilización de FPGAs, como alternativa a los microcontroladores, en aplicaciones que requieren bajo consumo de energía y manejo de grandes volúmenes de datos.

El alcance de la tesis incluye la optimización del diseño para lograr *throughputs* que satisfagan los requerimientos de sistemas de EEG de alta performance (en el orden de Mbps). El diseño se caracterizó en tres plataformas distintas basadas en FPGA, utilizando las tradicionales variables: área, velocidad y consumo. Reducir esta última es un objetivo central de este trabajo, ya que se apunta a un dispositivo inalámbrico alimentado a batería. En primera instancia se comparan las implementaciones en FPGA y la del microcontrolador, en términos de consumo de potencia y máxima frecuencia de muestreo. Además, se define una métrica de performance por watt, calculada como la máxima frecuencia de muestreo alcanzable sobre la potencia consumida. Esto permite, para una cantidad de canales dada, evaluar que tecnología va a otorgar la mayor performance (frecuencia de muestreo) por watt consumido. Siempre que no se superen los requerimientos de consumo, esta medida resulta útil para determinar qué plataforma resulta más conveniente para determinada aplicación.

En la Figura 1.1 se muestran los principales componentes que conforman un sistema inalámbrico para la adquisición de señales de EEG. El bloque AFE y ADC (del inglés, Analog Front–End y Analog to Digital Converter) se encarga de filtrar, amplificar y adquirir (convertir a digital) las señales de EEG provenientes de los

[3]Del inglés, *Hardware Description Language*.

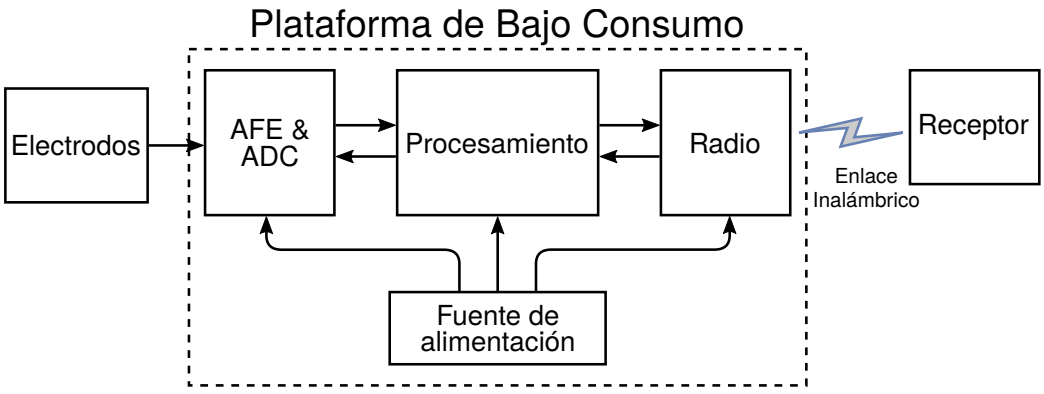

Figura 1.1: Diagrama de un sistema de adquisición de señales de EEG inalámbrico.

electrodos. El bloque Procesamiento se encarga del manejo y procesamiento de los datos y del control general del sistema. La Radio transmite de manera inalámbrica las muestras adquiridas (procesadas o no) a un Receptor externo. Este trabajo se centra en la etapa de compresión, que está incluida en la unidad de Procesamiento. De forma complementaria y, con el objetivo de realizar una aproximación al sistema completo, también se hará mención a los restantes bloques.

Como se mencionó anteriormente, las características de estos sistemas hacen que la transmisión inalámbrica sea el componente de mayor peso dentro del consumo energético. Por lo tanto, reducir el tiempo de utilización de la radio va a impactar fuertemente en el consumo; razón por la cual se estudia la compresión de datos como método de ahorro energético. Si se logran tasas de compresión como las de los algoritmos reportados en [13], la reducción en consumo va a ser importante, siempre y cuando el consumo agregado debido al procesamiento de datos no sea demasiado alto. Por esta razón, en este trabajo se caracteriza únicamente la implementación del algoritmo de compresión, dejando de lado el manejo de la interfaz analógica y la radio. Luego, tomando como punto de referencia la implementación en el microcontrolador con transmisión Bluetooth, es posible sacar conclusiones generales sobre un sistema completo basado en FPGA.

1.1. Revisión bibliográfica sobre trabajos relacionados

A lo largo de las últimas décadas se han propuesto diversos algoritmos de compresión de señales biomédicas, donde la mayoría saca provecho de la correlación espacial (entre muestras provenientes de diferentes electrodos) y temporal (entre muestras tomadas de un electrodo en diferentes instantes). Sin embargo, muchos de ellos no son apropiados para dispositivos portables de adquisición en tiempo real. Para empezar, estarían descartados los métodos que requieran la lectura completa de todas las muestras. Existen otros métodos que requieren procesar bloques de datos, como es el caso de MPEG-4 Audio Lossless Coding (ALS) [16], que también ha sido usado para la compresión de señales biomédicas [17]. El problema de ALS es que para ser efectivo requiere bloques de 2048 o más muestras, lo que implica

una latencia de varios segundos en aplicaciones de EEG.

Este problema también afecta a los métodos que utilizan transformaciones lineales para eliminar la correlación [18–20]. Otra desventaja de utilizar estos métodos, es que la cantidad de operaciones crece de forma superlineal con el número de canales y el tamaño del bloque de datos a procesar. Este es el caso de [19], donde los autores proponen algoritmos de compresión de señales de EEG con y sin pérdidas, basados en la transformada de ondícula (*wavelet*) y codificación volumétrica. Otro ejemplo es el algoritmo propuesto en [18], que aprovecha la correlación entre canales mediante la transformada de Karhunen–Loeve y utiliza una transformada tiempo–frecuencia para reducir la redundancia temporal. El problema se ve acrecentado al realizar descomposición en valores singulares, como en el caso de [20], donde analizan diversos métodos de descomposición de matrices y tensores para reducir la correlación. Un incremento en la cantidad de operaciones se traduce directamente en un aumento de la potencia consumida y limita la frecuencia máxima de muestreo, ya que extiende el tiempo de procesamiento. Por estas razones, estos métodos no son los más adecuados para plataformas de bajo consumo.

En [21] los autores estudiaron diferentes algoritmos de compresión sin pérdidas aplicados a señales de EEG de uno o múltiples canales, entre los cuales se encuentra un método similar al propuesto en [13]. Además, proponen nuevos algoritmos basados en el modelo autorregresivo multivariado y acondicionamiento de error basado en contexto, con codificación de Huffman. Concluyen que sus métodos combinados obtienen resultados en ratio de compresión superiores al resto para las bases de datos utilizadas. Pero, utilizan sus propias implementaciones de los algoritmos contra los que se comparan. Si se contrastan sus resultados con los de [15], se puede observar que el ratio de compresión es superior para algunas bases de datos, pero no para otras. Por otro lado, su algoritmo es más exigente computacionalmente que [13] y, por lo tanto, menos apropiado para una plataforma de bajo consumo.

La implementación de algoritmos eficientes para sistemas inalámbricos de adquisición de señales de EEG presenta un gran desafío. Principalmente porque las plataformas de bajo consumo no logran satisfacer los requerimientos de cómputo y memoria necesarios. Una alternativa que tiene pocos requerimientos de hardware y un bajo consumo de energía es Compressed Sensing (CS), que ha sido usado para sistemas de adquisición de EEG inalámbricos [22]. Los autores identifican que el problema de los algoritmos actuales de CS radica en la mala calidad de la señal reconstruida. El causante de esto es que las señales de EEG no son dispersas ni en el dominio del tiempo ni en el de las transformadas. Como alternativa proponen un método basado en Block Sparse Bayesian Learning (BSBL), que no depende de que la señal a comprimir sea dispersa, en el cual logran relativamente buenos resultados en la calidad de reconstrucción.

En [23] los autores presentan el diseño e implementación de un sistema inalámbrico de adquisición en tiempo real de señales de EEG, que implementa compresión de datos usando la transformada discreta de ondícula (DWT). Reportan resultados (simulaciones de LabView) de ratio de compresión relativo en función de parámetros como largo de los filtros y distorsión de la señal recuperada. Obtienen buenos resultados en ratio de compresión pero a costa de una gran distorsión en las señales

adquiridas. Utilizan la plataforma myRIO–1900 de National Instruments, la cual está basda en un FPGA de Xilinx y un procesador ARM Cortex-A9. No reportan medidas de consumo energético, pero la plataforma que utilizan tiene un consumo base de 2.6 W y llega a un máximo de 14 W, valores demasiado grandes para un dispositivo inalámbrico de bajo consumo.

En [24] el autor presenta el diseño en hardware de un algoritmo de compresión de EEG compuesto por un predictor adaptativo de tipo fuzzy, un esquema basado en votación y un codificador de entropía de tres etapas, donde las dos primeras usan códigos de Huffman y la segunda de Golomb-Rice. Realizaron la implementación en tecnología CMOS (del inglés, Complementary Metal Oxide Semiconductor) de 0.18 μm, alcanzando resultados en ratio de compresión relativo, para 23 canales, de 58 % y un consumo de potencia de 2.6 mW, operando a 100 MHz y con un voltaje de alimentación de 1.62 V.

En [25] presentan la implementación, en tecnología CMOS de 65 nm, de un compresor sin pérdidas de señales de EEG y ECG, diseñado para dispositivos de monitoreo inalámbricos de bajo consumo. El algoritmo consiste en un predictor basado en modulación por impulsos codificados discretos (DPCM) y codificación de Golomb–Rice. Para cuatro canales obtienen resultados en ratio de compresión relativo de 51 % y un consumo de potencia simulado de 170 μW, a una frecuencia de 24 MHz y un voltaje de alimentación de 1.0 V.

Habiendo realizado una extensa revisión de la literatura se puede concluir que las publicaciones sobre dispositivos de adquisición de señales de EEG inalámbricos, que realicen compresión de datos y estén centrados en el bajo consumo de energía, son relativamente escasas. En especial las que incluyen implementaciones en hardware. Esto es esperable, ya que los requerimientos de capacidad de procesamiento y memoria impuestos por esta clase de sistemas, normalmente exceden las capacidades de plataformas de bajo consumo alimentadas a batería.

1.2. Descripción del documento

El resto del documento está organizado de la siguiente manera: el Capítulo 2 introduce el algoritmo de compresión utilizado y los resultados de su implementación en una plataforma de bajo consumo basada en un microcontrolador. En el Capítulo 3 se detalla el diseño en hardware del algoritmo. En el Capítulo 4 se presenta la tecnología utilizada y, en particular, lo relativo al consumo de potencia en FPGAs. Los resultados y comparación entre ambas implementaciones son presentadas en el Capítulo 5. En el Capítulo 6 se mencionan cuestiones relacionadas al sistema completo. Por último, las conclusiones y comentarios sobre trabajo futuro se presentan en el Capítulo 7.

Capítulo 2

Algoritmo de compresión

Los sistemas de EEG inalámbricos requieren la adquisición, almacenamiento y transmisión de señales provenientes de múltiples canales. Esto puede generar grandes volúmenes de datos, especialmente cuando se procesan cientos de canales y las frecuencias de muestreo rondan los kilo–hertz. En esta clase de aplicaciones la compresión de datos juega un rol fundamental, ya que permite disminuir los requerimientos de almacenamiento y transmisión, que a su vez redunda en un hardware más simple y menor consumo energético.

Los dispositivos vestibles para la adquisición en tiempo real de señales de EEG poseen, en general, limitada capacidad de cómputo y requieren bajo consumo de energía. Por lo tanto, se beneficiarían en gran medida de algoritmos de compresión de baja complejidad. Además, en ciertos casos es necesario adquirir las señales sin distorsión, lo que implica que la compresión sea sin pérdidas.

El algoritmo propuesto en [13] reúne todas las condiciones mencionadas anteriormente y posee los mejores resultados de ratio de compresión de la literatura al momento de la publicación. Este algoritmo es denominado RLS y está basado en aritmética de punto flotante (ver Sección 2.1). Tomando como base RLS, se desarrollaron dos nuevos algoritmos [15] que utilizan aritmética de enteros (entre otras novedosas optimizaciones) y reducen considerablemente los requerimientos de cómputo y memoria a un leve costo en el ratio de compresión. Uno de los algoritmos modificados, descrito en la Sección 2.2, fue el elegido en este trabajo para realizar la implementación en hardware. A efectos comparativos, en la Sección 2.3 se presenta la implementación del algoritmo en una plataforma de bajo consumo basada en un microcontrolador.

2.1. Algoritmo RLS

Al igual que la mayoría de los algoritmos de compresión de EEG, el método propuesto en [13] hace uso de la correlación temporal y espacial de las señales. Esta proviene de sus propiedades naturales, tales como: continuidad temporal; correlación en la actividad neuronal entre distintas regiones, y suavizado espacial, causado por la presencia de tejido entre la fuente (neuronas) y el punto de medida (electro-

dos). A continuación se describe brevemente el funcionamiento del algoritmo:

- La primera etapa es de predicción. Tanto el codificador como el decodificador predicen el valor de cada muestra en base a las anteriores. Al comprimir, la diferencia entre la muestra y su predicción es codificada usando códigos de Golomb–Rice [26]. El decodificador recibe la muestra codificada y obtiene la original usando las muestras procesadas anteriormente.

- Las muestras de cada canal son codificadas en un orden pre-establecido que sigue una estructura de árbol. Hay un canal raíz que es predicho usando únicamente sus muestras pasadas, mientras que todos los demás poseen un canal padre (que corresponde a su padre en el árbol) que los asiste en la predicción. Esto significa que información pasada (y presente) del canal padre es utilizada para predecir la muestra actual del canal hijo.

- Las predicciones se calculan mediante un promedio ponderado de un conjunto de predictores lineales de diferentes órdenes, lo cuales son combinados usando un método de ponderado exponencial [27] para lograr la predicción final.

- Los predictores lineales son adaptivos y se actualizan en tiempo real usando una implementación eficiente del algoritmo RLS (del inglés, Recursive Least Squares) [28] para múltiples canales.

El algoritmo propuesto en [13] impone requerimientos de performance y memoria difícilmente alcanzables por una plataforma de bajo consumo. El nuevo algoritmo propuesto en [15] utiliza, en lugar de RLS (que requiere punto flotante), una extensión a múltiples canales de un simple algoritmo de predicción, adaptativo y basado en números enteros, que fue propuesto por Speck en [29]. El nuevo compresor, que los autores denominaron MCS (*Multi–Channel Speck*), logra tasas de compresión similares a las de la versión RLS, pero debido a su simpleza algorítmica, utiliza una fracción de los recursos y es considerablemente más rápido que su predecesor. Otras importantes optimizaciones fueron: la implementación de un ponderado exponencial basado en enteros y la selección manual del conjunto de predictores que, en base a resultados empíricos, lograron la mejor performance. Si además se sustituyen los predictores adaptivos por fijos, se logra una importante mejora en performance a un leve costo en capacidad de compresión. Esta última modificación conforma el segundo algoritmo, al que los autores llamaron MCF (*Multi-Channel Fixed*), que fue el utilizado en el presente trabajo para realizar la implementación en hardware.

2.2. Algoritmo MCF

Esta sección resume brevemente los pasos principales del algoritmo de compresión, para dar al lector un panorama general de lo que fue implementado en hardware. Por más detalles referirse a [15].

Se considera un sistema de N canales, donde se denota $x_i(n)$ a la muestra escalar del canal i en el tiempo n. Todas las muestras escalares son cuantizadas y representadas mediante números enteros en un intervalo finito.

Los cuatro predictores fijos utilizados son los siguientes:

$$\hat{x}_i(n) = x_i(n-1) \tag{2.1}$$
$$\hat{x}_i(n) = 2x_i(n-1) - x_i(n-2) \tag{2.2}$$
$$\hat{x}_i(n) = 3x_i(n-1) - 3x_i(n-2) + x_i(n-3) \tag{2.3}$$
$$\hat{x}_i(n) = x_i(n-1) + x_j(n) - x_j(n-1) \tag{2.4}$$

donde $\hat{x}_i(n)$ es la predicción de la muestra a comprimir y $x_j(n)$ es la muestra del canal padre j. Las ecuaciones 2.1, 2.2 y 2.3 corresponden a predictores lineales de primer, segundo y tercer orden respectivamente. La ecuación 2.4 corresponde a un predictor bilinear de primer orden.

La predicción final se obtiene mediante un promedio ponderado exponencial, que utiliza potencias de dos en lugar de la función exponencial. La predicción en tiempo n del canal i se calcula como:

$$\hat{x}_i(n) = \frac{\sum_{r=1}^{4} w_r(n)\hat{x}_i^r(n)}{\sum_{r=1}^{4} w_r(n)}, \tag{2.5}$$

donde w_r es un peso positivo que decae con el error medio absoluto $\bar{e}_r$ de cada predictor. Los pesos se definen como:

$$w_r = \begin{cases} 2^{s_{max} - c_n \bar{e}_r}, & \text{si } s_{max} \geq c_n \bar{e}_r \\ 0, & \text{de otra manera} \end{cases} \tag{2.6}$$

donde s_{max} es una constante y c_n es un parámetro adaptativo usado para mantener los pesos dentro del intervalo $[W_{min}, W_{max}]$. El error medio absoluto $\bar{e}_r(n)$ se obtiene mediante la siguiente ecuación:

$$\bar{e}_r(n) = (1-\beta)\bar{e}_r(n-1) + \beta e_r(n), \tag{2.7}$$

donde $0 < \beta < 1$ es un parámetro fijo y $e_r(n)$ es el valor absoluto del error del predictor r en el tiempo n. Para implementar 2.7 utilizando únicamente aritmética de enteros se realiza el cambio de variable $s_r(n) = \beta^{-1}\bar{e}_r(n)$ y se obtiene la siguiente expresión recursiva:

$$\begin{aligned} s_r(n) &= \beta^{-1}\bar{e}_r(n-1) + e_r(n) \\ &= s_r(n-1) - \bar{e}_r(n-1) + e_r(n) \end{aligned} \tag{2.8}$$

que solamente depende de sumas y restas. Luego, eligiendo β como potencia de dos con exponente negativo, se obtiene $\bar{e}_r(n)$ aplicando un simple desplazamiento de bits:

$$\bar{e}_r(n) = s_r(n) \gg b, \quad \text{con } \beta = 2^{-b} \tag{2.9}$$

Durante la ejecución del algoritmo, los pesos que ponderan a los predictores tienden a estabilizarse, cambiando solamente cuando la estadística de la señal varía significativamente. Por lo tanto, para evitar cálculos innecesarios los pesos son actualizados cada $T(n)$ muestras, comenzando con $T(0) = 1$. Para determinar el tiempo en que se realizará la próxima actualización, se comparan los pesos actualizados con los anteriores y se aplica la siguiente fórmula:

$$T(n+1) = \begin{cases} \min\{2T(n),\ T_{max}\} & \text{Si los pesos coinciden} \\ \max\{T(n)/4,\ 1\} & \text{Si los pesos difieren} \end{cases} \tag{2.10}$$

Luego, la próxima actualización se realiza en el tiempo $n + T(n+1)$.

La diferencia entre la muestra y su predicción final se codifica mediante Golomb–Rice. Si el error de predicción es bajo, los valores a codificar se concentran cerca del cero con una distribución de probabilidad similar a la geométrica. Esto implica que los códigos de Golomb van a lograr buenas tasas de compresión, ya que son óptimos para tales distribuciones.

El código de Golomb de orden m aplicado a un número natural a se divide en dos partes: un cociente q expresado en codificación unaria y un resto r en codificación binaria. Esto es:

$$\begin{gathered} Golomb_m(n) = unaria(q) \oplus binaria(r) \\ q = \lfloor a/m \rfloor, \quad r = a \bmod m \end{gathered} \tag{2.11}$$

Golomb–Rice es una simplificación de lo anterior donde m es potencia de dos ($m = 2^k$). Esto permite operar utilizando el exponente natural k, lo que acelera considerablemente las operaciones tanto en software como en hardware. Para lograr una codificación eficiente es crucial encontrar el valor óptimo de k. En el enfoque secuencial (que es el utilizado en este algoritmo), los datos son codificados individualmente y el parámetro k se determina en base a los datos anteriores. Para el cálculo de k se utiliza el método propuesto en [30] que se muestra en la siguiente expresión:

$$k = min\{i \mid 2^i M \geq A\} \tag{2.12}$$

donde A es el acumulado del valor absoluto del error de predicción y M la cantidad de muestras codificadas. Los valores de A y M se dividen a la mitad cada cierta cantidad fija de muestras (parámetro configurable). La ecuación 2.12 puede implementarse de forma extremadamente simple, iterando sobre k como se muestra en el algoritmo 1.

El algoritmo de compresión permite la codificación con pérdidas, la cual aumenta la tasa de compresión al costo de una distorsión por muestra controlada. Pero dado que esta característica no fue considerada en este trabajo, todos los resultados que se presentan, tanto para microcontrolador como para las FPGAs, refieren a la compresión sin pérdidas.

Algoritmo 1: Cálculo del parámetro k de Golomb.

```
k = 0;
aux_M = M;
while aux_M < A do
    k = k + 1;
    aux_M = M ≪ k;
end
```

2.3. Implementación en microcontrolador

Esta sección describe la implementación del algoritmo de compresión en una plataforma de bajo consumo basada en un microcontrolador. Los resultados obtenidos son usados como referencia para las posteriores implementaciones en FPGAs.

2.3.1. Plataforma de bajo consumo

La plataforma de adquisición de señales de EEG, representada en las Figuras 2.1 y 2.2 está compuesta por un AFE, un ADC, un microcontrolador de bajo consumo, una radio Bluetooth (BT) y un sub–sistema de alimentación. Todos los componentes están alimentados de una fuente de continua de 3.3 V. La señal de EEG es adquirida desde un conjunto de electrodos conectados al AFE.

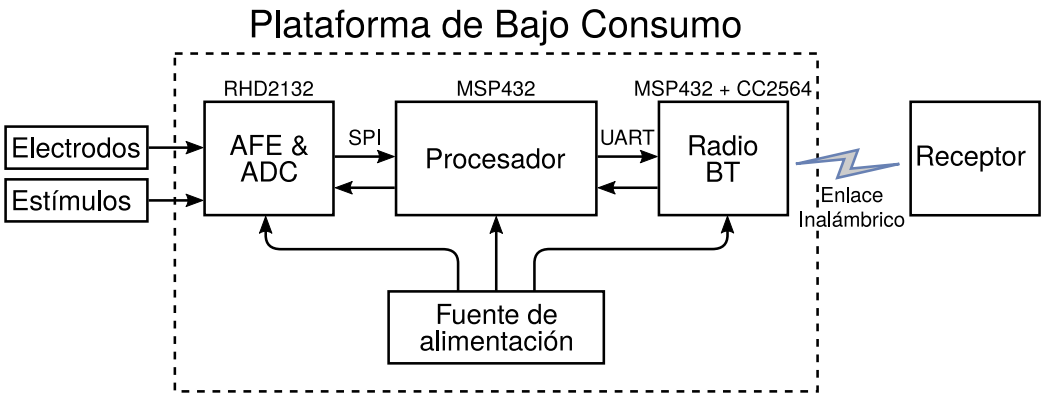

Figura 2.1: Diagrama de bloques del sistema de EEG inalámbrico basado en un microcontrolador.

Las etapas de AFE y ADC están comprendidas en una plataforma, desarrollada en [31], basada en dos integrados RHD2132 de Intan Technologies. Cada chip RHD2132 es capaz de adquirir, amplificar, digitalizar y transmitir via interfaz serie SPI (del inglés, Serial Peripherial Interface) hasta 32 canales a 30 ksps. El RHD2132 posee bajo ruido referido a la entrada (2.4 μV_{rms}), ancho de banda programable y bajo consumo de energía. Por ejemplo, el consumo total de corriente de dos chips adquiriendo 64 canales a 500 sps por canal es 1.8 mA y a 1 ksps por canal es 2.1 mA.

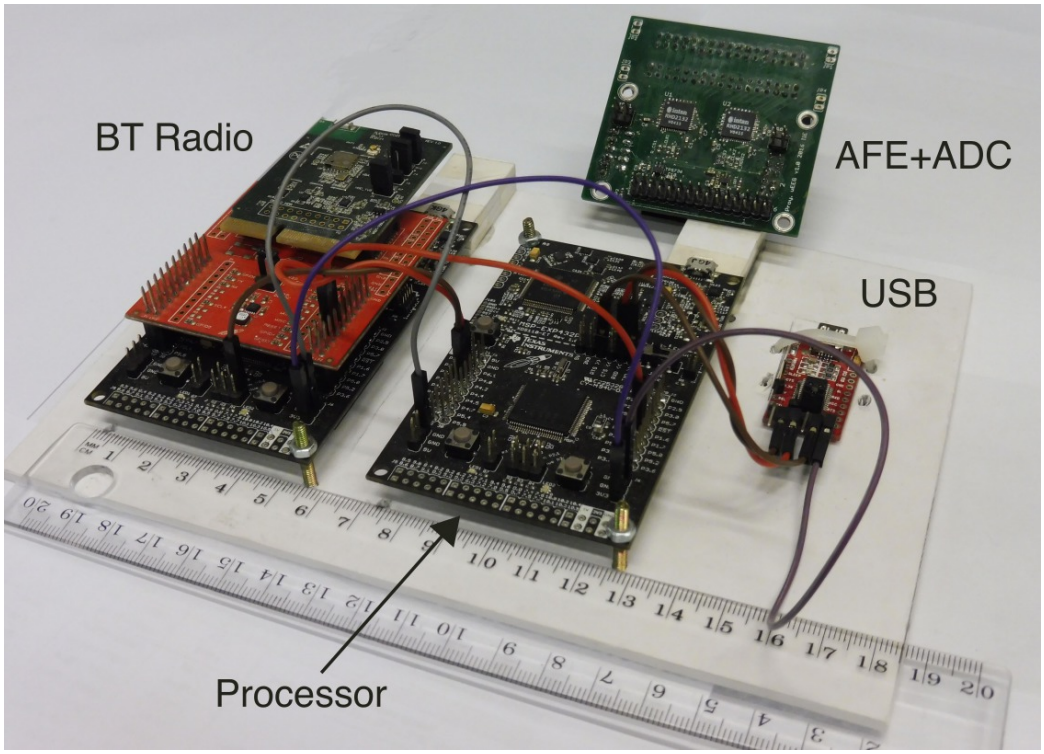

Figura 2.2: Plataforma de bajo consumo para la adquisición de señales de EEG, basada en un microcontrolador y una radio Bluetooth.

El bloque procesador consiste en un microcontrolador MSP432P401R de Texas Instruments de 32-bits basado en la arquitectura ARM Cortex-M4F. Posee una frecuencia máxima de reloj de 48 MHz, 256 kB de memoria FLASH y 64 kB de memoria RAM. El consumo típico es de 4.6 mA en modo activo y ofrece múltiples modos de bajo consumo (sleep modes), donde el consumo de corriente se reduce a cientos de nano–amperes. Además, el microcontrolador posee un importante número de periféricos, dentro de los cuales se encuentran las interfaces SPI para comunicarse con los chips RHD2132, una UART para la comunicación con la radio BT y un temporizador para controlar la frecuencia de muestreo del sistema.

La radio BT es un módulo basado en el integrado CC2564 de Texas Instruments que implementa Bluetooth 4.1. Este soporta el modo de bajo consumo (BLE), el modo básico (BR) y el modo de transferencia de datos mejorada (EDR). En este trabajo se utilizó el modo EDR con el perfil de puerto serie (SPP), lo que habilita tasas de transferencia de datos relativamente altas. Por ejemplo, una configuración de 31 canales a 1 ksps por canal generaría una tasa de datos que excede las capacidades de BLE 4.1.

El software embebido en el procesador es responsable de recibir las muestras adquiridas, ejecutar el algoritmo de compresión y enviar los datos comprimidos al módulo BT. La arquitectura elegida es Round–Robin con interrupciones, donde las rutinas de atención a interrupción son usadas ampliamente para intercambiar datos, dejando al procesador en un modo de bajo consumo cuando no es requerido.

El temporizador del microcontrolador es usado para disparar la adquisición de un nuevo conjunto de muestras. Las muestras (una por canal) son recibidas por la interfaz SPI y almacenadas en una memoria (*buffer*). Luego de que las muestras de todos los canales son recibidas se ejecuta la compresión. La salida del compresor es almacenada en otro *buffer* para ser posteriormente transferida al módulo BT a través de la UART. Una vez completado el ciclo, el microcontrolador entra en modo de bajo consumo.

Para probar la plataforma en un ambiente controlado se quitaron los chips RHD2132 y se utilizaron muestras provenientes de bases de datos de BT públicas (ver Sección 2.3.2). En una primera prueba se cargan las muestras en la memoria interna del microcontrolador para obtener el tiempo mínimo de compresión. En una segunda instancia se reciben las muestras desde un PC conectado a través de una interfaz USB, de forma de obtener resultados más realistas, ya que en este nuevo escenario se considera el tiempo que insume la recepción de los datos.

2.3.2. Resultados experimentales

Se realizaron una serie de experimentos para evaluar el desempeño del algoritmo MCF en la plataforma de bajo consumo. Se presentan los resultados de ratio de compresión, tiempo de ejecución, utilización de memoria, consumo de potencia y máxima tasa de datos (*throughput*) alcanzable. El algoritmo fue inicialmente desarrollado en una PC de escritorio y luego el código fuente fue modificado para su operación en la plataforma. El software para el microcontrolador fue compilado utilizando GNU v4.8.4 (Linaro).

Las señales de EEG utilizadas en los experimentos fueron obtenidas de las siguientes bases de datos públicas:

- DB1a y DB1b [32,33]: Señales de EEG de 64 canales, 160 Hz, 12 bits por muestra (bps) obtenidas de 109 sujetos usando el sistema BCI2000. Las muestras se dividen en 2 minutos de tareas de visualización motora[1] (DB1a) y 1 minuto de calibración (DB1b).
- DB2a y DB2b [34] (BCI Competition III): Señales de EEG de 118-canales, 1 kHz, 16 bps de 6 sujetos realizando tareas de visualización motora (DB2a). DB2b se obtiene mediante el subsampleo a 100 Hz de DB2a.
- DB3 [35] (BCI Competition IV): Señales de EEG de 59 canales, 1 kHz, 16 bps de 7 sujetos realizando tareas de visualización motora.
- DB4 [36]: Señales de EEG de 31 canales, 1 kHz, 16 bps de 15 sujetos realizando tareas de clasificación y reconocimiento de imágenes.

Para los experimentos realizados en la plataforma de bajo consumo se utilizaron configuraciones de 21, 31 y 59 canales y las señales de EEG fueron seleccionadas de las bases de datos DB2, DB4 y DB3, respectivamente. Las muestras utilizadas en las pruebas con frecuencias de 250 Hz y 500 Hz fueron obtenidas mediante un subsampleo de los datos originales.

[1] Representación mental de un movimiento sin realizar ningún gesto motor.

Ratio de compresión

Los archivos de cada una de las bases de datos fueron comprimidos por separado y el ratio de compresión (CR) resultante, medido en bits por muestra (bps), fue calculado como L/N_s, donde N_s es la suma de la cantidad de muestras escalares de todos los archivos de la base de datos y L es la suma del número de bits de todos los archivos comprimidos. El método de cálculo implica que cuanto menor es el valor de CR, mejor el resultado de la compresión.

Tabla 2.1: Ratio de compresión (CR) en bits por muestra (menor CR es mejor) del algoritmos MCF para diferentes bases de datos. Comparación con versión MCS y con el estado del arte.

Algorithm	DB1a	DB1b	DB2a	DB2b	DB3	DB4
MCS	4.82	4.94	5.34	6.97	5.47	3.81
MCF	5.09	5.18	5.96	7.41	5.90	4.35
[13]	4.70	4.79	5.21	6.93	5.42	3.58
[16]	5.37	5.45	5.69	7.69	5.99	3.73

La Tabla 2.1 muestra los CRs obtenidos, para todas las bases de datos, por los algoritmos MCF, MCS, RLS [13] y ALS [16] (este último reporta el mejor CR para las mismas bases de datos en [16, 19, 20, 37]).

Se observa que la performance de MCF es inferior que la de MCS; esto se debe a la utilización de predictores fijos en lugar de los adaptivos. Pero es gracias a esto que se logran importantes mejoras en el tiempo de procesamiento y la utilización de memoria. Sin embargo, la performance de MCF es superior que la del mejor algoritmo ALS reportado en [16, 19, 20, 37] para todas las bases de datos menos DB4.

Consumo de energía

El setup usado para medir el consumo de corriente del sistema se presenta en la Figura 2.3. Para medir la corriente se colocó un resistor shunt en serie con la fuente de continua (3.3 V). La caída de tensión a través del shunt es amplificada y adquirida mediante un ADC de 12 bits conectado a un PC.

Las muestras de entrada son provistas al sistema mediante una interfaz serie (USB–UART) desde el PC y la tasa de transferencia de datos es controlada mediante la combinación de un temporizador interno del microcontrolador y control de flujo por hardware. Al completarse la compresión de un vector de muestras el procesador entra en modo de bajo consumo hasta que expire el temporizador.

En la Tabla 2.2 se muestra el consumo de corriente del sistema en función de la tasa de datos (*throughput*) a la entrada. El *throughput*, en la tercera columna, se calcula como la cantidad de bits que procesa el compresor por segundo. Esto es, el número de canales multiplicado por la cantidad de bits por muestra y por la frecuencia de muestreo. La cuarta columna es la corriente promedio consumida por el procesador y la quinta columna es el consumo del procesador más el módulo Bluetooth.

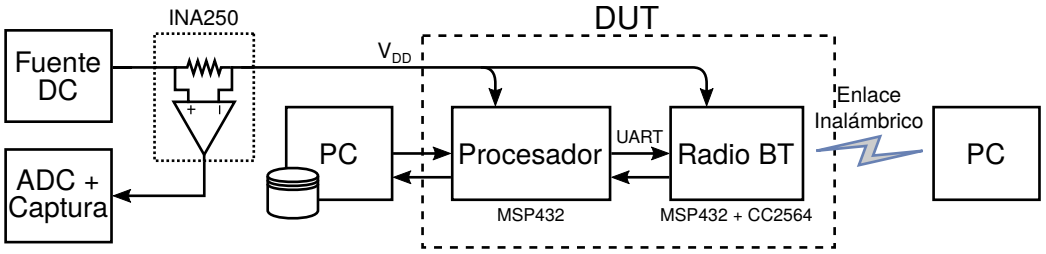

Figura 2.3: Diagrama del banco de medida de consumo de corriente.

Número de canales	Frecuencia de muestreo (Sps)	*Throughput* (kbps)	Consumo@3.3V uC (mA)	Consumo@3.3V uC+BT (mA)
21	250	84	1.53	20.9
21	500	168	1.93	22.1
21	1000	336	2.65	24.0
31	250	124	1.70	21.7
31	500	248	2.27	23.0
31	1000	496	3.34	24.9
59	250	236	2.21	23.2
59	500	472	3.29	26.2

Tabla 2.2: Consumo de corriente del procesador aislado y del procesador más el módulo Bluetooth en función del throughput a la entrada para el algoritmo MCF.

Tiempo de ejecución y utilización de memoria

El desempeño de la plataforma en términos de tiempo de procesamiento y utilización de memoria se muestra en la Tabla 2.3. La segunda columna muestra el tiempo promedio necesario para procesar todos los canales y la tercer columna indica el máximo teórico de frecuencia de muestreo (considerando que el procesador está siempre en modo activo y sin tener en cuenta el tiempo que toma el trasiego de datos). Los datos reportados son para la versión sin pérdidas del algoritmo MCF. La versión con pérdidas no aumenta la utilización de memoria y el incremento en tiempo de procesamiento es menor a un 3 %.

Cantidad de canales	Tiempo de proc. por muestra (ms)	Max. frecuencia de muestreo (sps)	Utilización de RAM (kB)
21	0.432	2313	11.7
31	0.593	1686	14.8
59	1.232	812	23.4

Tabla 2.3: Performance de MCF en la plataforma de bajo consumo.

Para medir el tiempo de compresión el procesador fue aislado del resto del

sistema y el software se modificó para que las muestras fueran leídas de memoria FLASH y el resultado de la compresión escrito en memoria RAM. Las medidas de tiempo de ejecución fueron realizadas con la herramienta *Count Event* (incluida en el IDE Code Composer Studio), contando ciclos de reloj entre dos *breakpoints* y luego realizando la división entre la frecuencia de reloj para obtener el tiempo transcurrido. La frecuencia de reloj del MSP432 se fijó en 48 MHz para todos los casos.

Capítulo 3

Implementación en Hardware

En el presente capítulo se detalla la implementación en hardware del algoritmo de compresión. En primer lugar se describe el compresor de un canal (que es la parte central del diseño) y todos los sub–bloques que lo conforman. Luego se presentan dos arquitecturas diferentes para la compresión de múltiples canales. La primera explota al máximo el paralelismo de las FPGAs y alcanza tiempos de compresión mínimos a costa de una gran utilización de recursos. La segunda se basa en la reutilización de recursos y apunta a dispositivos de menor área y consumo de potencia.

Para realizar la descripción del hardware se utilizó mayoritariamente el lenguaje VHDL y en menor medida Verilog. En un esfuerzo por lograr un diseño sintetizable por la mayor cantidad de herramientas posibles, en la parte central del circuito no se usaron primitivas o bloques propietarios de los fabricantes. En algunas FPGAs lograr tal independencia en el lenguaje es imposible, ya que hay funcionalidades que requieren la utilización de bloques propietarios. Estos casos se dan exclusivamente en la capa superior (*top–level*) del diseño, e involucran básicamente bloques de generación de relojes y configuración de pines de entrada y salida.

En etapas iniciales del desarrollo, tanto las muestras como el resultado de la compresión se almacenaban en memoria interna del chip. Este método es fácil de implementar pero tiene la limitante de no permitir evaluar al sistema con grandes cantidades de muestras, lo cual es necesario para lograr la estabilización de los parámetros adaptativos del algoritmo. Por esta razón, se agregó al diseño una interfaz SPI maestro que recibe las muestras y envía los datos comprimidos. Otra ventaja de esta modificación es que permite evaluar al sistema en un escenario más realista.

Para todos los diseños se incluye un bloque encargado de manejar la frecuencia de los relojes del sistema. Este bloque difiere según la FPGA, pero en general es (o está basado en) un bucle de seguimiento de fase o PLL (del inglés, Phase–Locked Loop). Se generan dos relojes, uno utilizado por el bloque SPI en la comunicación serie y otro para el resto de la lógica.

El diseño es parametrizable (en etapa de síntesis) en: cantidad de canales, cantidad de bits por muestra y varios parámetros relacionados con el algoritmo. Estos son:

- Máximo desplazamiento permitido al ponderar los predictores; el cual define el máximo valor de los pesos.
- Factores de crecimiento y disminución del intervalo para actualizar los pesos.
- Cantidad de muestras que se promedian en el cálculo del error absoluto medio de los predictores.
- Valor inicial de parámetro k de Golomb–Rice.
- Cantidad de muestras que se consideran para dividir a la mitad los valores de N y A (estadísticas de Golomb).

La verificación del diseño se realizó de forma incremental, primero en cada bloque por separado y luego en los diseños completos. Para todos los casos se escribieron bancos de prueba (*testbench*) en VHDL, que luego se simularon utilizando el ModelSim versión 10.5b. Para el diseño completo, el *testbench* guarda en un archivo de texto la salida comprimida, que luego es comparada con la salida de la versión de software del algoritmo. Para las pruebas en FPGAs se envía la salida comprimida a un PC para realizar la verificación. Este procedimiento es diferente según la arquitectura y se explica en las próximas secciones.

3.1. Compresor de un canal

Todas las operaciones aritméticas involucradas utilizan enteros con signo. Los desplazamientos de bits y las sumas extienden la cantidad de bits del resultado para evitar desbordamiento (*overflow*). La división entera, realizada en la etapa de ponderado exponencial, reduce la cantidad de bits al tamaño original de la muestra de entrada. Esto se debe a que el resultado de la división corresponde a una estimación de la muestra y, por lo tanto, puede ser representada con la misma cantidad de bits. Esto se ve claramente reescribiendo la ecuación 2.5 como:

$$\hat{x_i} = \sum_{r=1}^{4} \lambda_r \hat{x_i}^r, \quad \text{con} \sum_{r=1}^{4} \lambda_r = 1 \tag{3.1}$$

donde resulta evidente que $\hat{x}$ está acotado por los valores mínimos y máximos de $\hat{x_i}^r$.

Para este trabajo se configuró un ancho de palabra de 16 bits para las muestras de entrada y un valor máximo de desplazamiento (exponente de los pesos) de 10. Esto significa que la suma de los cuatro predictores ponderados es representada con 28 bits, ya que cada predictor desplazado al máximo requiere 26 bits y las sumas necesarias para combinarlos agrega 2 bits adicionales. Esta señal de 28 bits es el dividendo en la división entera, mientras que el divisor es la suma de los pesos. Esta última se representa con 13 bits, que corresponden a la cantidad necesaria para representar la suma de los pesos máximos.

La Figura 3.1 muestra un diagrama del flujo de datos del compresor para un canal, donde pueden identificarse las diferentes etapas del algoritmo descrito en 2.2. A continuación se describen someramente los distintos bloques:

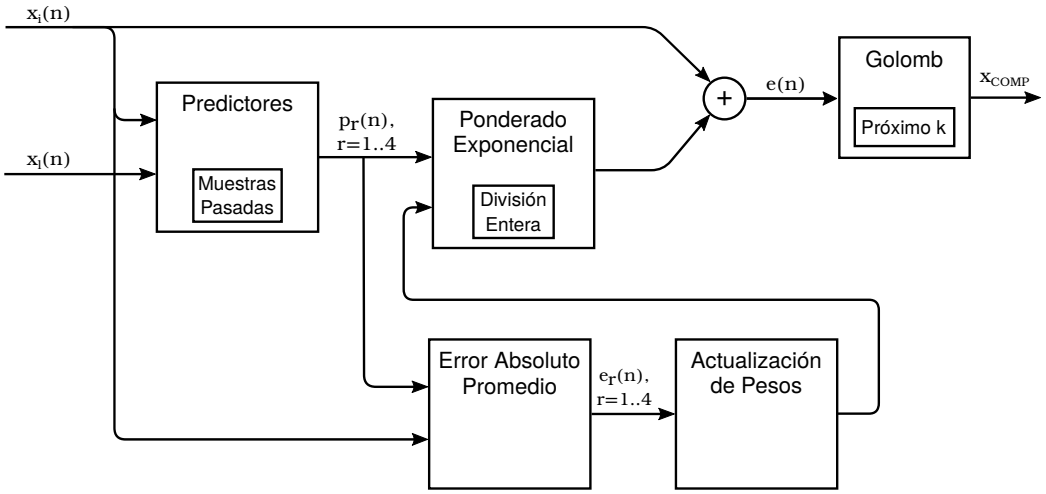

Figura 3.1: Diagrama del flujo de datos para el compresor de un canal.

Predictores

Este bloque implementa las operaciones de las ecuaciones 2.1, 2.2, 2.3 y 2.4. Las multiplicaciones por constantes se realizan mediante sumas y desplazamientos de bits. Por ejemplo, la multiplicación $3x_i(n-1)$ resulta en:

$$(x_i(n-1) \ll 2) - x_i(n-1) \tag{3.2}$$

El bloque además posee los registros necesarios para almacenar las muestras pasadas.

Ponderado exponencial

Como se muestra en la ecuación 2.5, este bloque combina los cuatro predictores para obtener la predicción final. Para ello, primero se ponderan los predictores mediante *barrel shifters*[1], ya que los pesos son potencias de dos. Como se explicó en la Sección 2.2 los pesos pueden ser cero. En ese caso se activa la salida del comparador y se pone a cero la señal correspondiente al predictor ponderado (ver Figura 3.2). Luego, se suman los predictores ponderados y se ingresan al bloque divisor junto a la suma de los pesos, la cual es calculada en el bloque Actualización de Pesos. Se requiere de un bloque divisor, ya que el resultado de la suma de los pesos no es necesariamente una potencia de dos. Este bloque es uno de los más demandantes del diseño en cuanto a utilización de recursos y latencia. Debido a la gran profundidad de lógica que posee, fue necesario intercalar barreras de registros para lograr frecuencias de reloj aceptables, resultando en una latencia de ocho períodos de reloj.

[1]Bloque de lógica combinatoria que permite desplazar una palabra una cantidad variable de bits. El número de posiciones que se desplaza se determina mediante una entrada de control.

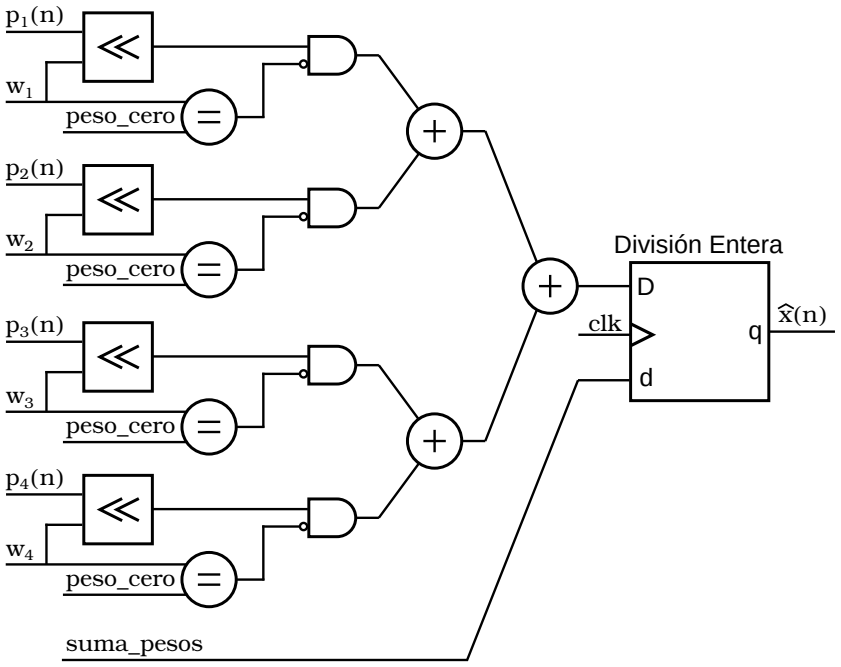

Figura 3.2: Implementación del bloque de Ponderado Exponencial.

Actualización de Pesos

El bloque de actualización de pesos recibe el error acumulado de cada predictor y actualiza los pesos de acuerdo a la ecuación 2.6. En la Figura 3.3 se muestra un diagrama de flujo del funcionamiento del bloque. Los pesos actualizados deben estar dentro del intervalo $[W_{min}, W_{max}]$, de lo contrario, son calculados nuevamente. Por lo tanto, el bloque implementa un bucle en el que se ajusta el parámetro c_n y se vuelven a calcular los pesos hasta que se cumpla la condición. Esto es un potencial problema, ya que se está introduciendo una latencia indeterminada al sistema. Sin embargo, se probó empíricamente que esto no sucede (ver Sección 3.2), principalmente por dos razones:

1. Los pesos se actualizan cada cierto intervalo variable, que en régimen alcanza $T_{max} = 256$ (ver Sección 2.2).

2. La actualización de pesos se hace en paralelo con el bloque Golomb.

A continuación se listan algunos comentarios generales sobre la implementación del bloque:

- El valor del parámetro c_n es potencia de dos, por lo tanto, se almacena únicamente su exponente. Para esto se utiliza un contador que incrementa o decrementa su valor de a uno.

- Para el cálculo de los exponentes de los pesos, primero se desplaza a la derecha el error acumulado $\hat{e}_r(n)$ una cantidad igual al exponente de c_n. Luego el resultado se le resta a s_{max}.

- Cuando $s_{max} < c_n \bar{e}_r$ el peso debe ser cero, en ese caso, se le asigna al exponente una palabra reservada. Esta es identificada en el bloque de ponderado exponencial como el código para poner a cero el correspondiente predictor.

- Además del contador de c_n y los registros donde se almacenan los exponentes actuales de los pesos, es necesario otro grupo de registros para guardar los exponentes anteriores. Esto es necesario para saber si al calcular los pesos su valor efectivamente se modificó.

- $T(n)$ es una potencia de dos y su valor se almacena como $T(n)-1$ ($máscaraTn$ en la Figura 3.3), de manera que, para determinar si es tiempo de actualizar los pesos se realiza el AND bit a bit entre $máscaraTn$ y el contador de actualizaciones ($cont\_act$). A modo de ejemplo, si $T(n) = 16$ y $cont\_act = 8$, la operación a realizar es:

$$\begin{array}{r} 00001111 \\ \wedge\ 00001000 \\ \hline 00001000 \neq 0 \end{array}$$

  para la cual no corresponde actualizar. Para aumentar $T(n)$ se desplazan los bits un lugar a la izquierda y se agrega un uno en la posición menos significativa. Para disminuir $T(n)$ se desplazan los bits dos lugares a la derecha y se completa con ceros a la izquierda.

- El control lo realiza una máquina de estados interna que maneja las señales de habilitación de todos los bloques mencionados e implementa el bucle para el cálculo de pesos.

Error absoluto promedio

Este bloque se encarga de acumular el promedio del valor absoluto del error para cada predictor. Dado que se implementa la ecuación 2.8, no se acumula exactamente el error medio $\hat{e}_r(n)$ sino que se utiliza la variable auxiliar $s_r(n)$. El β^{-1} de la ecuación 2.7 se fija en 256, es decir que $b = 8$. Sin embargo, el cambio de variable de $s_r(n)$ para calcular los pesos se deshace en el bloque de Actualización de Pesos, cuando se multiplica por el parámetro c_n. Es decir, el b queda comprendido dentro del exponente de ${c_n}^2$

El cálculo de los $s_r(n)$ se realiza en un solo período de reloj y el resultado se almacena en registros. Los siguientes pasos (ver Figura 3.4) describen la operación del bloque:

- Se obtiene $e_r(n)$ realizando la diferencia entre la muestra actual y cada una de las predicciones individuales. Esta operación agrega un bit a los resultados.

[2]Esta característica ya existía en la versión de software del algoritmo. Se menciona en el presente capítulo para dar un panorama completo de la implementación en hardware.

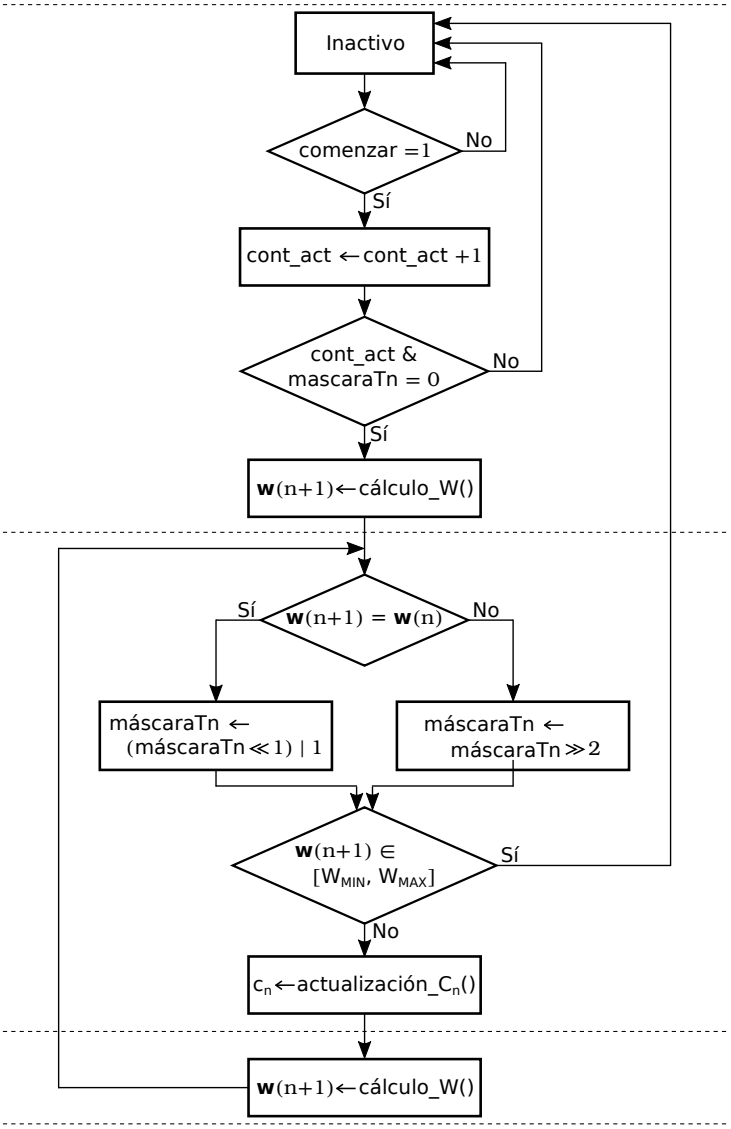

Figura 3.3: Diagrama de flujo del bloque Actualización de Pesos, donde $\mathbf{w}(n)$ representa los pesos en el tiempo n, *cont_act* guarda la cuenta del tiempo entre actualizaciones y *máscaraTn* representa el tiempo hasta la próxima actualización ($T(n)$). Las líneas punteadas dividen los eventos según el flanco de reloj en el que ocurren.

- Se obtiene $\hat{e}_r(n-1)$ deshaciendo el cambio de variable en $s_r(n-1)$, es decir, dividiendo entre 256 (desplazamiento de ocho bits a la derecha). Previo a la división, se suma 128 a $s_r(n-1)$ para realizar un redondeo sobre el bit menos significativo luego del corrimiento[3].

- Luego se suman $s_r(n-1)$ y el valor absoluto de $e_r(n)$, y se resta $\hat{e}_r(n-1)$.

[3]Esta característica también existía en la versión de software del algoritmo.

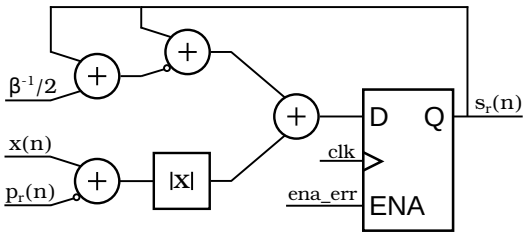

Figura 3.4: Implementación del bloque de Error Medio absoluto para uno de los predictores (Ecuación 2.9). Se suma $\beta^{-1}/2$ a $s_r(n-1)$ para realizar un redondeo sobre el bit menos significativo luego del corrimiento

Codificador de Golomb–Rice

Este bloque implementa la codificación de Golomb–Rice [38], con parámetros adaptativos, que se aplica sobre la diferencia entre cada muestra y su predicción. Esto quiere decir que la salida del bloque es el resultado de la compresión. La salida es serie, lo que implica que la latencia del bloque es variable, ya que el largo de la compresión no es constante.

En la Figura 3.5 se presenta un diagrama del diseño del bloque Golomb. En el primer flanco de reloj se registra el resultado de aplicarle, a la entrada x, la siguiente función de intervalo y superposición:[4]

$$x' = \begin{cases} 2x, & \text{si } x \geq 0 \\ -2x-1, & \text{si } x < 0 \end{cases} \tag{3.3}$$

En el siguiente flanco comienza a generarse la salida codificada, de a un bit por flanco de reloj. Los bits que conforman la salida se toman de multiplexores, que son manejados por el bloque de Control. Este se basa en la salida de los bloques Unaria y Binaria, y en contadores, para determinar que bits deben ser colocados a la salida.

Al finalizar la codificación, se habilita el sub–bloque Estadísticas, que computa el valor de k a utilizar en la próxima entrada. Este sub–bloque contiene sus propios contadores para almacenar las estadísticas, y una máquina de estados que controla la habilitación de los contadores y el bucle para el cálculo de k (ver ecuación 2.12). El método de cálculo de k contribuye a la latencia variable del bloque Golomb.

Consideraciones generales

A continuación se listan algunos aspectos relacionados al diseño completo del compresor para un canal que no se muestran en la Figura 3.1.

- La información del árbol de codificación, que provee la asignación de los canales padres, está implementada en VHDL como lógica combinatoria.

[4]Esta característica, que ya existía en la versión de software del algoritmo, es necesaria dado que Golomb funciona para números no negativos.

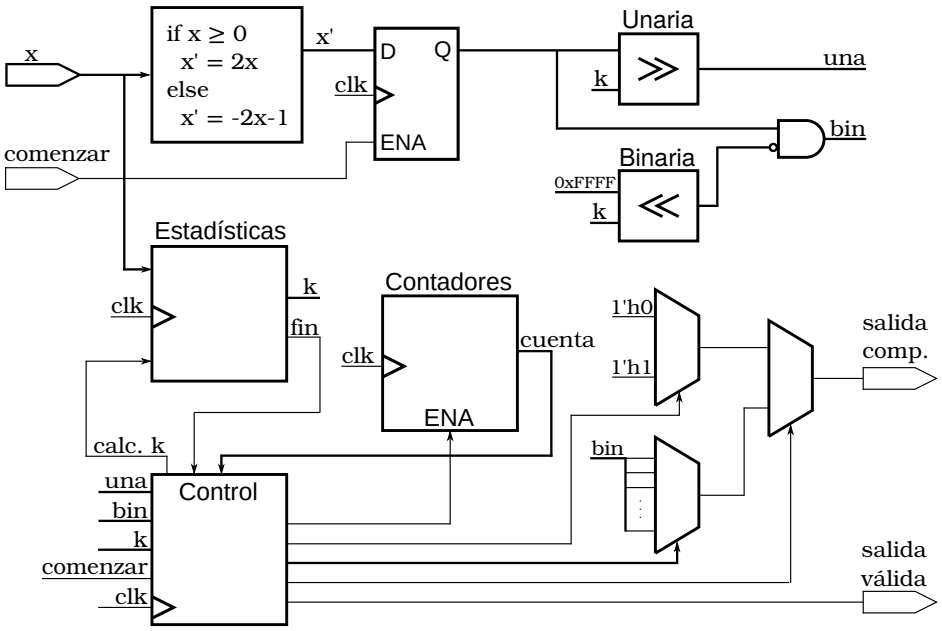

Figura 3.5: Diagrama de bloques de Golomb.

- Un bloque de control se encarga de habilitar las distintas etapas del circuito. En los bloques de latencia fija, que son todos menos Golomb y Actualización de Pesos, la transición de estados depende de contar ciclos de reloj. Para habilitar los bloques de latencia variable, los cuales implementan su propias máquinas de estados, se les da un pulso en una de sus entradas y luego se espera una señal de finalización.

- A la salida de Golomb se conecta un bloque que realiza la conversión serie a paralelo de las muestras comprimidas. El ancho de palabra de salida de este bloque es configurable en tiempo de síntesis.

3.2. Tiempo de compresión

Como se mencionó anteriormente, el diseño posee un tiempo de compresión por muestra variable. Esto se debe principalmente a la latencia del bloque Golomb, que depende del tamaño de la muestra comprimida (sz) y del valor del parámetro k calculado. A partir del diseño y conociendo los valores recién mencionados, es posible determinar la cantidad exacta de ciclos de reloj que tarda el circuito en entregar el resultado. En la Figura 3.6 se representa la latencia de los distintos bloques, indicando en qué ciclos de reloj está activo cada uno. El Control ocupa tres ciclos de reloj en detectar el fin de la compresión, obtener las nuevas muestras y comenzar la próxima compresión. El bloque Actualización de Pesos requiere un ciclo cuando no corresponde modificar los pesos y una cantidad variable w de ciclos cuando sí corresponde.

Se comprobó empíricamente, para una de las bases de datos (DB2), que w no

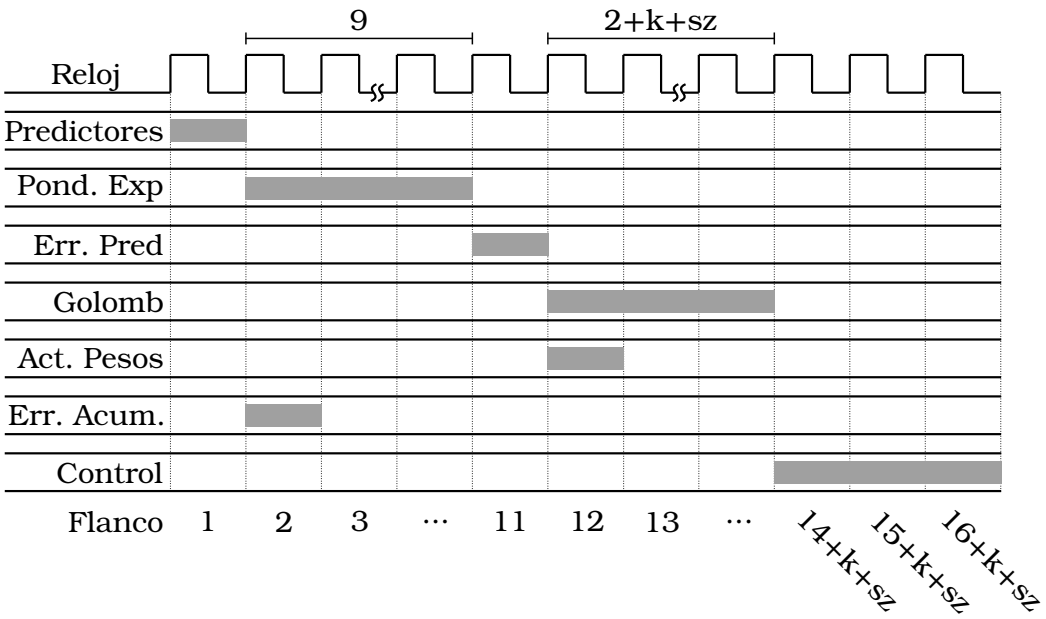

Figura 3.6: Ciclos de reloj en los que actúan las distintas etapas del bloque compresor.

afecta al tiempo de compresión promedio. Para ello, primero se determinó cual era el canal con mayor latencia en base a la tasa de compresión y a los valores de k obtenidos en la versión de software del algoritmo. Para este canal se calculó el tiempo promedio de compresión por muestra del diseño en hardware, sin considerar la latencia de la actualización de pesos $(16 + k + sz)$, y luego se comparó con el resultado obtenido mediante simulaciones. Para la obtención de este último, se simuló la compresión de un número M de muestras y luego se dividió el tiempo total de simulación entre M. En la Figura 3.7 se muestra una porción de la simulación y un ejemplo de medida para la compresión de tres muestras. Para 50 mil muestras, el resultado obtenido mediante simulaciones y el calculado utilizando los valores promedio de k y sz, coinciden.

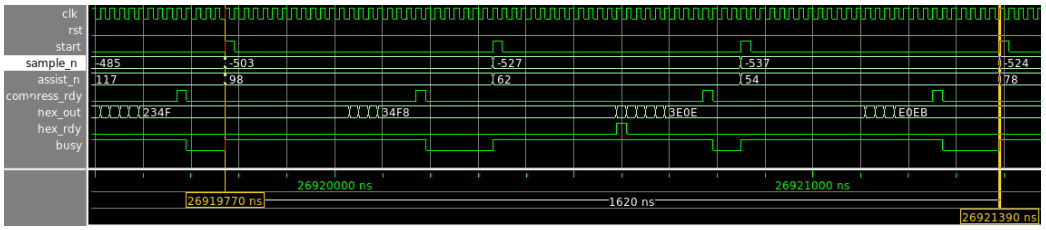

Figura 3.7: Simulación de un canal para la verificación del tiempo de compresión promedio.

Para determinar la máxima frecuencia de muestreo es necesario conocer el tiempo de compresión por muestra promedio. Por lo tanto, resulta beneficioso que la latencia del bloque compresor dependa de pocas variables, y que estas se puedan determinar sin mayor dificultad.

3.3. Arquitectura Paralela

La primera arquitectura implementada consiste en una instancia del bloque compresor por cada canal, es decir, las muestras de todos los canales se procesan en paralelo. Esta estrategia alcanza las mayores velocidades de compresión, pero requiere una gran cantidad de recursos; lo que implica una FPGA realmente grande y, en consecuencia, de alto consumo de potencia. Se optó por explorar esta arquitectura en primera instancia para investigar los mínimos tiempos de compresión alcanzables y, en segunda instancia, porque su implementación es trivial a partir del bloque compresor de un canal.

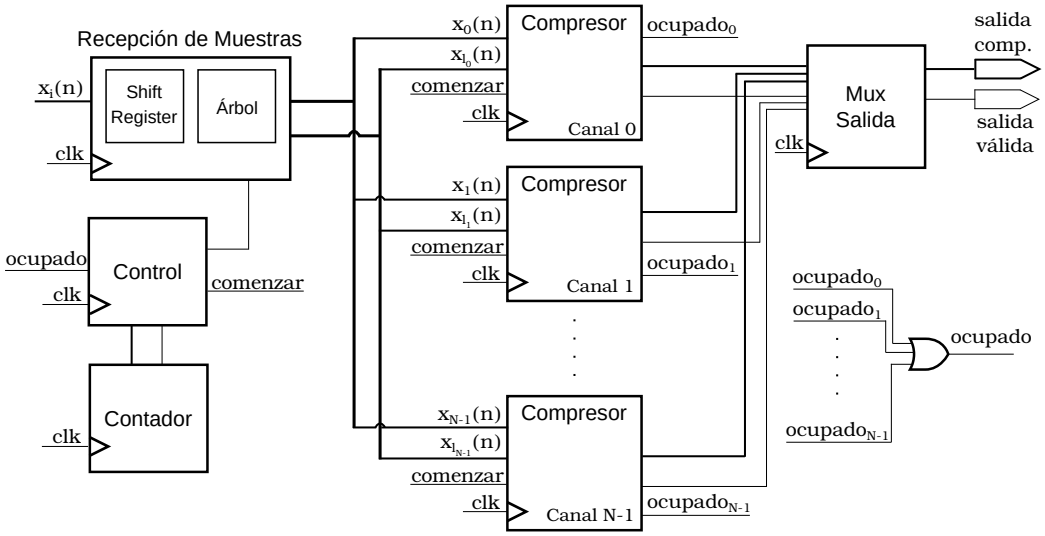

Figura 3.8: Diagrama de la arquitectura paralela.

Un diagrama del circuito con la arquitectura paralela puede verse en la Figura 3.8. Para proveer las muestras a los distintos canales se agrega un bloque de Recepción de Muestras, que está compuesto por dos partes. La primera es un *shift–register* encargado de recibir las muestras, que para este diseño llegan de a una por vez. Sería más conveniente una recepción en paralelo de las muestras, pero se diseñó de esta forma pensando en utilizar como hardware de adquisición de las muestras el chip RHD2132, que posee comunicación serie. La segunda parte consiste en un banco de registros que copia el contenido del *shift–register* una vez que hayan llegado todas las muestras (para permitir una nueva recepción), y el bloque de lógica combinatoria que provee la información del árbol de codificación.

Para validar el diseño se agrega un bloque Multiplexor conectado a la salida de todos los canales, que se encarga de entregar las muestras comprimidas de a una por vez. El bloque contiene un banco de registros que copia las muestras comprimidas cuando están disponibles. Luego recorre los registros de a uno y, si tienen una nueva muestra, la pone a la salida. Este bloque se creó pensando en un diseño que incluyera una interfaz serie (hacia una radio), entonces, él sería el encargado de ir entregando las muestras comprimidas, de a una por vez, para ser transmitidas.

Además de la salida comprimida, el bloque debe indicar a qué canal corresponde, ya que esta información es necesaria para reconstruir las muestras en el decodificador.

Para la prueba se conectó la salida del bloque Multiplexor a un FIFO y se leyeron las muestras desde un PC a través de una interfaz JTAG–USB. Esta interfaz es demasiado lenta para una lectura continua de la salida del circuito; por lo tanto, fue necesario almacenar varias muestras comprimidas y consumirlas de a poco. La lectura de las muestras se automatizó mediante un *script tcl*.

El bloque Multiplexor se utilizó solamente para verificar el correcto funcionamiento del circuito en la FPGA, ya que no necesariamente es representativo de la operación final. Luego de validado el circuito, para su posterior caracterización se sustituyó el bloque Multiplexor por un XOR de todas las salidas. Esto se realiza como método sencillo para reducir la salida de un diseño y que la herramienta de síntesis no lo optimice. Si en un diseño existe lógica que no tiene efecto sobre el resto del circuito, como es el caso de las señales que deberían ir conectadas a un puerto de salida pero no lo están, la herramienta las elimina.

La arquitectura paralela permite alcanzar velocidades de compresión que exceden ampliamente los requerimientos planteados en este trabajo, pero no es la más adecuada para su utilización en un sistema de EEG inalámbrico. En primer lugar, debido al excesivo consumo energético que implica utilizar FPGAs con los recursos necesarios. En segundo lugar, aprovechar al máximo la velocidad de compresión implicaría que la tasa de datos de las muestras de entrada sea muy alta. Por ejemplo, para 21 canales y una frecuencia de reloj de 50 MHz se obtiene una frecuencia de muestreo de 1.85 MSps, lo que significa una tasa de datos de entrada de 415 Mbps. Con el hardware actual, lograr la frecuencia de reloj que requiere el bloque SPI para lograr la mencionada tasa de datos es imposible. Por esta razón, para probar la arquitectura paralela se sustituye el *shift–register* del bloque Recepción de Muestras con una memoria ROM con las muestras precargadas.

Otro aspecto a resolver sería lograr el ancho de banda necesario para transmitir las muestras comprimidas utilizando protocolos inalámbricos de bajo consumo. Por estas razones, esta arquitectura se toma como una prueba de concepto y se pasa a la implementación secuencial desarrollada en la Sección 3.4. La arquitectura paralela podría ser de utilidad en otro tipo de aplicaciones de adquisición y transmisión de señales multicanal de gran ancho de banda que se beneficien de la compresión de datos, pero que no tengan fuertes restricciones de consumo energético.

La caracterización de la arquitectura paralela se realizó en una FPGA Cyclone V 5CEBA4F23C7N, donde se logró implementar hasta un máximo de 14 canales. Este reducido número se debe a la gran cantidad de recursos que requiere el diseño en relación al tamaño de la FPGA. En las otras plataformas más pequeñas no se llegó a probar esta arquitectura, ya que no cabrían más de uno o dos canales.

3.4. Arquitectura Secuencial

Los requerimientos de área de la compresión en paralelo superan con creces a los recursos disponibles en una típica FPGA de bajo consumo. Por esta razón, se optó por una segunda arquitectura que implementa un único bloque compresor

para realizar la compresión de manera secuencial. Esto requiere de las siguientes modificaciones al bloque compresor original:

- El compresor debe almacenar, para cada canal, cierta información a medida que se comprimen las muestras. Por lo tanto, si se utiliza el mismo hardware para distintos canales, es necesario almacenar esta información (denominada contexto) para no perderla al cambiar de canal. Esto se resuelve agregando al diseño una memoria RAM y un mecanismo que permite, mediante lecturas y escrituras a memoria, alternar los contextos de los diferentes canales .
- El canal raíz utiliza un predictor menos que los demás, ya que no tiene canal padre. Esto implica diferencias en la implementación de varios de los bloques del compresor para dicho canal. Para resolver esto se agrega una señal de entrada, que tiene la función de bloquear el hardware relacionado con el predictor extra. Evidentemente, esta señal se debe activar externamente cuando el canal a comprimir sea el raíz.

El ancho de palabra y la profundidad de la memoria de contexto dependen de los parámetros elegidos y de la cantidad de canales, respectivamente. Para los valores elegidos en este trabajo el contexto de cada canal es de 270 bits, salvo el del canal raíz que ocupa 225 bits.

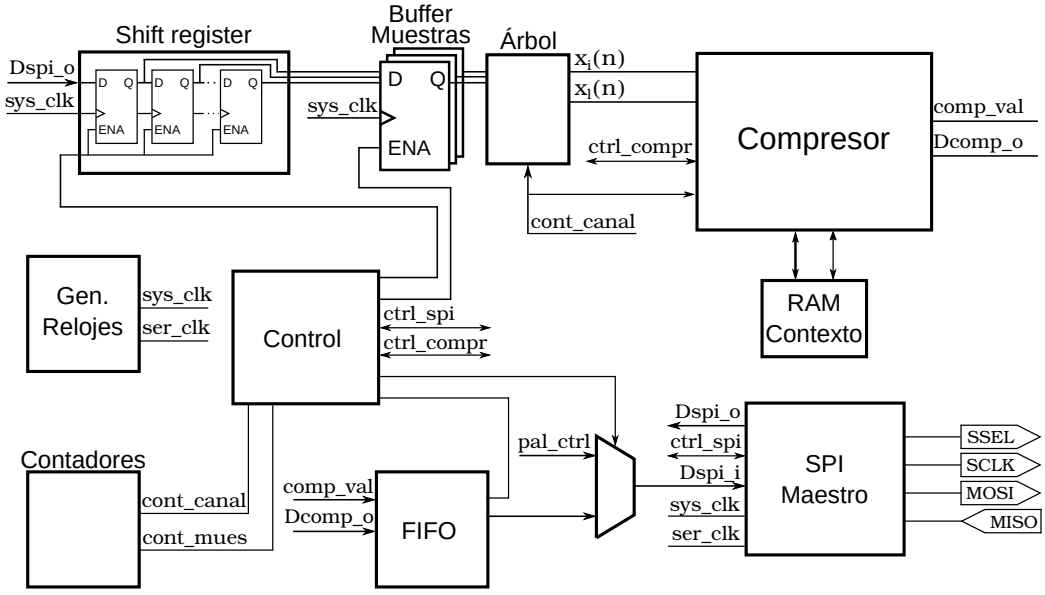

Figura 3.9: Diagrama de la arquitectura secuencial.

En la Figura 3.9 se observa un diagrama de bloques del diseño secuencial. Se utiliza un bloque SPI maestro que recibe las muestras a comprimir y envía el resultado de la compresión. Las muestras se reciben de a una, lo que implica que el ancho de palabra de la interfaz SPI coincide con el de las muestras, que en este caso se fija en 16 bits. A medida que se reciben, las muestras se almacenan en

un *shift–register* cuya profundidad coincide con la cantidad de canales. Cuando se termina de recibir un set de muestras, el contenido del *shift–register* se copia a un banco de registros y se da comienzo a la compresión, permitiendo comenzar la recepción de un nuevo set de muestras.

El bloque Compresor comprime las muestras de a una y guarda el resultado de la compresión en un FIFO, el cual hace de *buffer* entre el compresor y el bloque SPI. En cada ciclo SPI en el que se solicita una nueva muestra, si hay datos en el FIFO estos son enviados y, si no hay, se envía una palabra reservada. Esta es reconocida en el receptor y descartada. El bloque de Control se encarga de habilitar el bloque SPI si todavía quedan muestras por recibir, dar comienzo a la compresión de un nuevo set de muestras y seleccionar el dato a transmitir (muestra comprimida desde el FIFO o la palabra reservada). Esta última decisión se toma en base al estado de las señal Vacío del FIFO. El diseño implementa además un bloque generador de relojes como se mencionó al comienzo de este capítulo.

Para probar la arquitectura secuencial fue necesario desarrollar un diseño en otro FPGA que transmita las muestras y reciba el resultado de la compresión. Además, debe ser capaz de descartar el dato recibido si coincide con la palabra reservada. El diseño es relativamente simple, consiste de un bloque SPI esclavo, dos memorias y una máquina de estados. Este diseño se implementó en la Cyclone V, que además de poseer una buena cantidad de bits de memoria para almacenar las muestras, permite extraer el contenido de las mismas, desde un PC, para verificar que el compresor funciona correctamente. La herramienta utilizada para extraer el contenido de las memorias es el *In–System Memory Content Editor* del Quartus Prime.

Las modificaciones introducidas en la arquitectura secuencial solamente agregan un período de reloj al tiempo de compresión, que se debe al manejo de la memoria de contexto. Todos los demás bloques funcionan en paralelo y no introducen demoras. El bloque SPI no introduce latencia siempre y cuando la frecuencia del reloj de la transmisión SPI (*spiclk*) sea igual o mayor a la del reloj del resto de la lógica (*sysclk*). Esto se debe a que la latencia del bloque SPI es $16N$ ciclos (N es la cantidad de canales), que es menor a la suma de los tiempos de compresión promedio de todos los canales ($27N$ ciclos). Para lograr que *spiclk* y *sysclk* tengan la misma frecuencia fue necesario generar un reloj que opere al doble de la frecuencia de *sysclk*, ya que, para generar *spiclk*, el bloque SPI divide a la mitad la frecuencia del reloj de entrada. Esto es fácilmente realizable mediante un PLL o un divisor de frecuencia, bloques que proveen los fabricantes de las FPGAs en sus herramientas.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Tecnología utilizada

Este capítulo introduce brevemente a las FPGAs y en especial los aspectos relacionados con su consumo energético. Luego se presentan las diferentes plataformas utilizadas para probar la implementación en hardware del algoritmo de compresión. Por último, se describe la interfaz de comunicación SPI sobre LVDS (del inglés, Low–Voltage Differential Signaling), y se hacen menciones sobre el diseño para bajo consumo de energía.

4.1. Descripción de las FPGAs

Las FPGAs son dispositivos semiconductores compuestos por una matriz de elementos lógicos configurables; bloques dedicados, como memorias y multiplicadores, y una estructura programable de enrutamiento que permite interconectar los distintos bloques. Poseen además numerosas interfaces de entrada salida para la comunicación del dispositivo con el mundo exterior. Las FPGAs pueden ser reprogramadas después de la fabricación, es decir, pueden ser actualizadas una vez que están en funcionamiento (en el "campo") o incluso pueden modificarse para cumplir una tarea diferente. Esta característica los diferencia de los ASICs, que son fabricados para una función específica.

Si bien los FPGAs no alcanzan la performance de los ASICs de celdas estándar en cuanto a velocidad, consumo y área; los costos de diseño y el tiempo de salida al mercado en los FPGAs son considerablemente menores para bajos volúmenes de producción. Además, la reprogramabilidad les otorga a los FPGAs la flexibilidad del software manteniendo las características del hardware, por ejemplo, paralelismo masivo y arquitectura flexible.

Desde su creación, los dispositivos lógicos programables han atravesado diversos cambios tecnológicos y estructurales [39]. Al día de hoy, los FPGAs más utilizados son los basados en RAM estática (SRAM). Estos almacenan su configuración en celdas de SRAM distribuidas a lo largo del FPGA. Muchas de estas celdas se utilizan para seleccionar las líneas de interconexión, mientras que las restantes almacenan datos en lookup-tables (LUTs) que se utilizan para implementar funciones lógicas. Las celdas SRAM no requieren de un refresco periódico como las RAM dinámicas,

razón por la que consumen menos energía. Pero sí son volátiles, por lo que las FPGAs basadas en ellas requieren de hardware adicional capaz de almacenar el diseño y realizar la configuración. La gran ventaja de los FPGAs basados en SRAM es que utilizan el procesos de fabricación CMOS y por lo tanto, se benefician de los más recientes avances de esta tecnología. Además de la volatilidad, otras problemáticas que afronta de esta tecnología son la seguridad, el exceso de área y las fugas en los transistores que causan un alto consumo estático.

Una alternativa son las FPGAs basadas en memoria FLASH, que retienen su configuración cuando el dispositivo es apagado. No solo resuelven el problema de la volatilidad de la tecnología SRAM, sino que eliminan la necesidad de utilizar circuitos externos para almacenar y configurar el diseño. Adicionalmente, esta tecnología es más eficiente en su utilización del área y en cuanto a las pérdidas en los transistores. Una desventaja es que pueden ser reprogramadas una cantidad finita de veces.

Consumo de potencia en FPGAs

Al igual que todos los diseños digitales modernos, las FPGAs en general están basadas en compuertas lógicas CMOS. El consumo de potencia de los circuitos basados en esta tecnología puede dividirse en dinámico y estático.

El consumo estático, que se produce aún cuando los transistores no tienen actividad de cambio, puede calcularse como:

$$P_{est} = VccI_L \tag{4.1}$$

donde Vcc es la tensión de alimentación e I_L representa las corrientes de fuga, que están compuestas principalmente por las corrientes subumbrales (*sub–threshold leakage*) y las pérdidas de compuerta (*gate leakage*) de los transistores CMOS. Las corrientes de fuga, y por lo tanto, el consumo estático, empeoran a medida que la tecnología de fabricación avanza y los transistores se reducen en tamaño. Adicionalmente, las FPGAs requieren de lógica extra para soportar la reconfiguración, lo que causa que su consumo estático sea más elevado que un ASIC digital, para una misma funcionalidad [40].

El consumo dinámico tiene dos grandes componentes: las corrientes de cortocircuito y la carga y descarga de las capacidades de salida de los transistores. La primera es causada por la conducción simultánea de los transistores N y P durante una conmutación, lo que produce un breve flujo de corriente entre fuente y tierra. La segunda es causada por la carga y descarga de las capacidades parásitas de los nodos durante las conmutaciones. Esta potencia disipada se puede modelar como:

$$P_{din} = Vcc^2C(\alpha f) \tag{4.2}$$

donde C es la capacidad del nodo y αf representa la frecuencia de conmutación del nodo, que es un porcentaje de la frecuencia de reloj f. Esta componente es una de las más relevantes desde el punto de vista del diseño en FPGAs, ya que las demás dependen de aspectos de la tecnología de fabricación.

Las FPGAs se pueden dividir en grandes bloques, donde por lo general cada uno tiene su propia fuente de alimentación. Por lo tanto, el consumo total se compone de los siguientes elementos:

- Consumo del *Core*: Elementos lógicos, memorias internas, multiplicadores hardware, etc.
- Consumo de bloques analógicos: PLL, DCM, etc.
- Consumo de los puertos de entrada y salida.

En este trabajo se midió únicamente el consumo del *Core*, ya que es el más representativo de la FPGA. El consumo de los puertos de entrada/salida depende de la frecuencia de conmutación, de la capacidad de salida y del tipo de interfaz utilizada (LVDS, TTL, etc.), ya que esta última determina las tensiones y corrientes que maneja el puerto. Por lo tanto, se puede considerar que dicho consumo es independiente de la FPGA. Esto no implica que no deba ser tenido en cuenta, especialmente en trabajos como el que aquí se presenta, en el cual se apunta a un sistema completo donde el consumo de energía es una de las variables más relevantes. Sin embargo, al ser este trabajo una exploración de técnicas y plataformas que permitan alcanzar ciertos requerimientos de velocidad y consumo energético, el consumo de la entrada/salida no fue medido, ya que este puede ser determinado independientemente de la plataforma utilizada. Adicionalmente, el diseño realizado utiliza relativamente pocos puertos de entrada/salida, por lo tanto, la potencia consumida por estos va a ser menor que la del *Core*.

4.2. Plataformas utilizadas

El diseño se probó en tres plataformas provenientes de dos fabricantes distintos. La mayor parte del desarrollo se realizó en la placa DE0-CV de Terasic, que posee un FPGA Cyclone V 5CEBA4F23C7N de Intel. Este chip se eligió como punto de partida para el diseño, en primer lugar, por la familiaridad del autor con esta familia de FPGAs y, en particular, con las herramientas de diseño relacionadas. En segundo lugar, porque posee suficientes recursos y bloques dedicados que facilitan el diseño. El problema con la Cyclone V es que, debido a su tamaño, tiene un elevado consumo de potencia; razón por la cual se eligieron las FPGAs de Lattice: iCE40HX-8K y MachXO2 (LCMXO2-7000HE-4). Estas FPGAs poseen considerablemente menos recursos y están diseñadas especialmente (en particular la iCE40) para aplicaciones de bajo consumo de energía. En la Tabla 4.1 se resumen las características de las tres FPGAs utilizadas.

Las primeras dos columnas muestran la cantidad de LUTs y registros, componentes básicos de la lógica programable. Las FPGAs de Lattice poseen 10 veces menos elementos lógicos que la Cyclone V y en consecuencia consumen considerablemente menos energía. La tercera columna indica si la programación de la FPGA se pierde al desconectar la alimentación. Si bien todas las FPGAs de la tabla son basadas en memoria RAM estática (volátil), la MachXO2 posee memoria Flash

FPGA	LUTs (miles)	Registros (miles)	Volátil	RAM (kbits)	Pines de E/S	PLLs	DPSs
5CEBA4F23C7N	73.9	73.9	Sí	3383	224	2	66
iCE40HX-8K	7.68	7.68	Sí	128	210	2	0
LCMXO2-7000HE	6.86	6.86	No	234	114	4	0

Tabla 4.1: Características de las FPGAs utilizadas.

interna que permite almacenar la configuración y re–programar la FPGA apenas esta se enciende. Las otras dos requieren adicionar un circuito externo para ser programadas. La cuarta columna indica la cantidad de bits de memoria dedicada disponibles en cada chip. Esta memoria posee baja latencia y es vital en diseños como el de este trabajo, que requieren el trasiego de datos a gran velocidad y necesitan de buffers entre las distintas etapas. La sexta columna indica el número de pines de entrada/salida de propósito general, cantidad que supera con creces los requerimientos de la aplicación de este trabajo en cualquiera de las tres FPGAs. Las últimas dos columnas muestran la cantidad de PLLs y DSPs. La iCE40 y la MachXO2 no poseen bloques DSPs dedicados, lo cual no presenta un problema en este caso ya que no serían aprovechados.

La MachXO2 posee un modo de bajo consumo denominado *stand–by*, que reduce considerablemente el consumo de energía. Este modo es útil siempre que sea posible operar en ciclos de trabajo, despertando la FPGA cada cierto tiempo para realizar una tarea y luego cambiando al modo de bajo consumo el resto del tiempo.

4.3. Comunicación SPI sobre LVDS

Se eligió SPI para la comunicación con dispositivos externos por diversas razones:

- La interfaz es simple y muy flexible en cuanto al ancho de palabra a transmitir.
- Teóricamente no está limitado en frecuencia máxima, lo que permite alcanzar velocidades de comunicación superiores a otros protocolos serie.
- Soporta transmisión y recepción simultáneas (comunicación *full–duplex*).
- El chip RHD2132 (*front–end* analógico del sistema) se comunica mediante SPI.

Las altas frecuencias de muestreo requeridas deben ir acompañadas de una interfaz de comunicación rápida. Además, por las características del algoritmo no puede haber errores en la transmisión, ya que de haberlos no se podría reconstruir la señal sin distorsión. Una forma de alcanzar grandes velocidades y robustez frente a la interferencia es utilizar LVDS. Este es un estándar[1] genérico para interfaces di-

[1]Definido en el estándar ANSI/TIA/EIA-644

ferenciales de alta velocidad, que especifica únicamente las características eléctricas del transmisor y del receptor. Es decir, define la capa física pero no el protocolo, lo que le otorga gran flexibilidad.

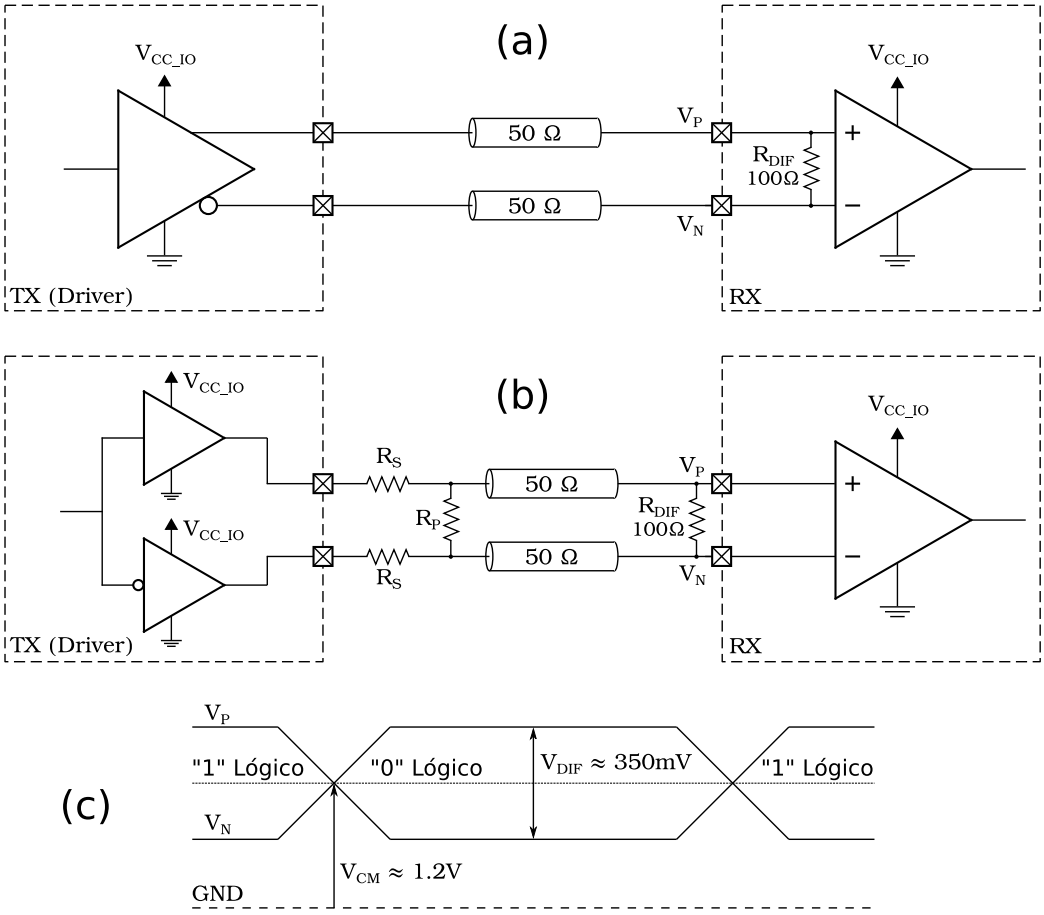

Figura 4.1: Diagramas de operación de una interfaz LVDS. (a) Driver diferencial (*true–LVDS*) y receptor con resistencia de terminación integrada. (b) Driver compuesto de dos salidas *single–ended* y una red de resistencias (*emulated–LVDS*), y receptor con resistencia de terminación externa. (c) Diagrama indicando las tensiones usuales de las señales involucradas en la comunicación LVDS.

En la Figura 4.1a se muestra un diagrama del circuito equivalente de una interfaz LVDS, y en la Figura 4.1c la forma de onda de la señales involucradas. El transmisor hace circular una corriente constante (aproximadamente 3.5 mA) a través de una resistencia de terminación de 100 Ω en el receptor, donde el sentido de las corrientes determina el nivel lógico. La señal es diferencial, es decir, corrientes iguales y opuestas fluyen por las líneas de transmisión. Al ser una señal diferencial, es necesario dedicar dos pines de la FPGA por cada puerto de entrada/salida.

Siempre y cuando las líneas de transmisión estén físicamente juntas (e.g par trenzado), los campos magnéticos producidos por las corrientes estarán fuertemente acoplados, lo que genera una gran reducción del ruido producido por interferencia

electromagnética. Al ser una señal diferencial tiene un fuerte rechazo a variaciones en el voltaje de modo común. Otra importante ventaja, es el bajo consumo de potencia debido a las bajas tensiones que maneja. El consumo del transmisor va a estar dominado por la potencia disipada en el resistor diferencial, esto puede calcularse como:

$$P_{LVDS} = \frac{V_{DIFF}^2}{R_{DIFF}} = 1.23mW \tag{4.3}$$

Por lo general, las FPGAs modernas proveen soporte para LVDS, e incluso permiten conectar el resistor de terminación internamente (e.g. la familia Cyclone V). Sin embargo, en algunos casos (e.g. la FPGA iCE40HX) no disponen de drivers diferenciales para el transmisor, y para implementar el protocolo es necesario emular interfaz. Esto se logra utilizando dos salidas *single–ended* y una red de resistencias que adapta las tensiones y genera las corrientes adecuadas (ver Figura 4.1b).

4.4. Diseño para bajo consumo

El proceso de diseño en FPGAs posee varias etapas o niveles, en cada una de las cuales pueden aplicarse técnicas para reducir consumo de potencia o de energía [41]. A nivel de sistema se pueden definir, por ejemplo, qué tareas se ejecutan en hardware y cuáles en software, los algoritmos a utilizar para resolver determinado problema, las interfaces de entrada/salida, el manejo de memoria, etc. Las definiciones a nivel de sistema van a ser las que mayor repercusión tengan en el consumo, pero en este trabajo no hay demasiado margen para realizar modificaciones, ya que se apuntó a una implementación puramente en hardware del algoritmo tal cual está definido. Solamente se actuó a nivel de interfaces, donde se escogió LVDS por su robustez frente al ruido y por su bajo consumo de potencia.

El siguiente nivel de abstracción, que también tiene un alto impacto en la reducción del consumo, es en la definición de la arquitectura. Aquí se pueden aplicar técnicas como reutilización de recursos, paralelismo, *pipelining* y utilización de modos de bajo consumo, entre otras.

El *pipelining* consiste en añadir barreras de registros entre caminos de lógica combinatoria, y afecta el consumo de potencia por su contribución en la eliminación de *glitches*[2]. Este método se aplica principalmente para aumentar la velocidad de operación de un circuito, dado que al insertar registros se acorta la profundidad de la lógica combinatoria, lo que permite aumentar la frecuencia de reloj. Sin embargo, también ha probado ser útil para reducir el consumo debido a la reducción de *glitches* [42]. Se utilizó el *pipelining* en uno de los bloques del diseño (divisor), pero con el objetivo de aumentar la máxima frecuencia de operación.

Al utilizar modos de bajo consumo, el circuito realiza sus funciones a una velocidad superior a la requerida y luego pasa a un estado de consumo de potencia

[2]Los *glitches* ocurren cuando un simple cambio en una entrada causa múltiples cambios en la salida de una compuerta. Estos incrementan el consumo de potencia, ya que contribuyen en la carga/descarga de las capacidades de los nodos (sin ningún efecto en la operación del circuito)

reducido hasta que llegue el momento de ponerse nuevamente en funcionamiento. Esto implica trabajar en ciclos y, si se cumple que el tiempo de funcionamiento en modo activo es menor que el tiempo en modo de bajo consumo, se pueden lograr importantes reducciones en el consumo total de energía. El problema de este método es que, por lo general, las FPGAs no cuentan con estos modos y tienen un alto consumo de potencia aún cuando no están realizando la función para la que fueron programadas. Esto se debe a que deben permanecer encendidas para no perder la configuración.

Una alternativa a los modos de bajo consumo es mantener el dispositivo sin energía (apagado) mientras no se utiliza y luego encenderlo cuando sea necesario. Dado que los FPGAs son en su mayoría basados en memoria volátil, al apagarlos pierden la configuración y es necesario reprogramarlas. Este método puede ser de utilidad en algunas aplicaciones, pero hay que considerar cuidadosamente la relación entre el consumo en operación y el consumo y el tiempo de configuración. En el presente trabajo este método quedó descartado, ya que los tiempos de configuración (del orden de milisegundos) limitarían la máximas frecuencias de muestreo. Una alternativa sería almacenar muchas muestras en una memoria externa, lo que permitiría mantener la FPGA apagada por más tiempo, pero esto requeriría hardware adicional que no solo encarece el diseño sino que aumenta considerablemente el consumo de energía.

Luego existen técnias a nivel de circuito, como por ejemplo el *clock gating* [43]. Esta involucra desactivar las líneas que llevan la señal de reloj, que por lo general contribuyen de forma importante al consumo dinámico total, a porciones inactivas del circuito. Técnicas de esta naturaleza no fueron utilizadas en este trabajo, pero sería interesante evaluar que efecto tienen en el diseño realizado, por lo que se agendan como trabajo a futuro.

Otros aspectos que afectan al consumo son la tensión de alimentación y la frecuencia de operación. En el primer caso no hay demasiado margen para variaciones, por lo que el ahorro en consumo sería bajo. La segunda variable por lo general se fija de antemano en base a los requerimientos del circuito, en este caso lo fija la frecuencia de muestreo.

En este trabajo, la técnica que obtuvo mejores resultados fue diseñar una arquitectura lo más reducida posible mediante la reutilización de recursos, de modo que el diseño pudiera ser implementado en un FPGA de bajo consumo (iCE40HX). La diferencia en consumo entre esta clase de FPGAs y las demás es importante, pero debido a su reducido tamaño no son siempre adecuadas para todos los desarrollos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Resultados Experimentales

En este capítulo se presenta la caracterización del diseño en las distintas plataformas. En primer lugar se describe la metodología de medida y luego se presentan los resultados de área, velocidad y potencia. Se analiza la performance del algoritmo en relación al consumo de potencia para las diferentes plataformas, y se comparan los resultados contra la implementación en el microcontrolador.

La implementación del algoritmo en el microcontrolador fue caracterizada para 21, 31 y 59 canales, por lo tanto, se utiliza esta misma cantidad de canales, y las mismas bases de datos, para evaluar los diseños en las FPGAs. La arquitectura paralela solamente se midió para 14 canales en la FPGA Cyclone V 5CEBA4, ya que es la cantidad máxima que la herramienta logra implementar con los recursos disponibles en dicho dispositivo. El diseño con 21, 31 y 59 canales se implementó en chips más grandes de la familia Cyclone V, siempre seleccionando el más chico posible en el que entrara el diseño. Para estos casos se estimó el consumo de potencia utilizando las herramientas del fabricante y se realizaron extrapolaciones tomando los datos de estimación y medidas reales del diseño con 14 canales.

5.1. Metodología para determinar el consumo

Desde el comienzo se planteó realizar medidas reales de consumo, ya que las herramientas de estimación en FPGAs suelen tener importantes errores [44, 45]. Aún así, en algunos casos se realizaron estimaciones utilizando las herramientas de los fabricantes, con el fin de obtener resultados preliminares. También fue necesario a la hora de evaluar la performance de la arquitectura paralela en los FPGAs con mayores recursos de la familia Cyclone V, ya que no se disponía de ejemplares para realizar medidas reales.

5.1.1. Estimación de consumo

Para minimizar el error en las estimaciones se deben ajustar correctamente los parámetros de las herramientas, de modo que el escenario considerado en la estimación se asemeje lo mejor posible a la realidad. En el caso de Intel, la herramienta

permite utilizar la actividad de conmutación de los nodos como insumo para realizar la estimación. Esto permitiría, en teoría, obtener una estimación de consumo dinámico del circuito cercana al valor en operación real. Para el caso de Intel se siguieron los siguientes pasos:

1. En primer lugar, se realizó la simulación funcional[1] a nivel de compuertas[2] del diseño completo. Para la simulación se utilizó el software ModelSim–Intel version 10.5b.

2. Durante la simulación se registró la actividad de todos los nodos del circuito en un archivo Value Change Dump (VCD).

3. Por último, se ingresaron estos datos en la herramienta de estimación *Power Analyzer Tool* (incluída en Quartus Prime) junto con otros parámetros como temperatura ambiente, características del chip (grado Industrial o Comercial) y datos sobre la disipación de la placa.

Siempre y cuando la simulación sea representativa de la realidad, esta metodología de trabajo permite a la herramienta de estimación utilizar la información de la actividad de cada nodo en las mismas condiciones que en la operación real. La gran diferencia radica en la ventana de tiempo considerada, que en la simulación es varios órdenes de magnitud más chica que en la realidad, debido a limitaciones en las capacidades del simulador y de la estación de trabajo.

Para el caso de la FPGA iCE40HX se utilizó la herramienta *Power Estimator* disponible en el entorno de desarrollo icecube2 v2017.08. Esta herramienta no permite la utilización de información de la actividad de los nodos, lo que produce estimaciones de consumo dinámico con un gran error. Para la estimación de consumo en la FPGA MachXO2 se utilizó la herramienta *Power Calculator* incluida en el software Lattice Diamond v3.10. Esta tampoco permite la utilización de la actividad de los nodos.

5.1.2. Medidas de consumo

Las FPGAs utilizadas poseen fuentes de alimentación reguladas y estables, por lo tanto, para obtener la potencia alcanza con medir la corriente. Para ello se utilizó una resistencia en serie (*shunt*) entre el regulador de tensión y la entrada de alimentación del chip. La caída de tensión se midió con un multímetro FLUKE 8846A que posee seis dígitos y medio de precisión. En la Figura 5.1 se muestra un diagrama de la configuración utilizada para la medida de consumo en la arquitectura secuencial. A todos los diseños se agregaron dos entradas, conectadas a botones externos, que permiten resetear el sistema y comenzar la compresión. Como se explicó en el capítulo 3, se utiliza otro FPGA (Cyclone V) para enviar

[1] Mejores resultados hubieran sido posibles realizando simulaciones que consideren el retardo real de la lógica (*Timing Simulation*), pero esta opción no está disponible para la familia Cyclone V [46].

[2] Simulación sobre circuito resultante luego de la etapa de Mapping y Place and Route.

las muestras y recibir la salida comprimida a través de una interfaz SPI. Dichas muestras son enviadas luego a un PC para verificar el correcto funcionamiento del algoritmo.

Las placas de desarrollo que contienen FPGAs de Lattice incluyen un resistor *shunt* que fue utilizado para medir el consumo de corriente. En los cálculos se consideró su valor nominal, reportado en la hoja de datos (1 Ω), ya que para medirlos correctamente deberían ser quitados del PCB. En los casos en que la placa no fue diseñada para permitir medidas de consumo se utilizó un *shunt* externo. En todas las plataformas fue necesario realizar modificaciones; en las de Lattice (iCE40HX, MachXO2) se soldaron cables a los bornes del *shunt* para conectar el medidor, y en la placa con un FPGA de Intel se retiró un resistor de cero ohm y se soldaron en su lugar dos pines, a los cuales se conectó el *shunt* externo (DE0-CV).

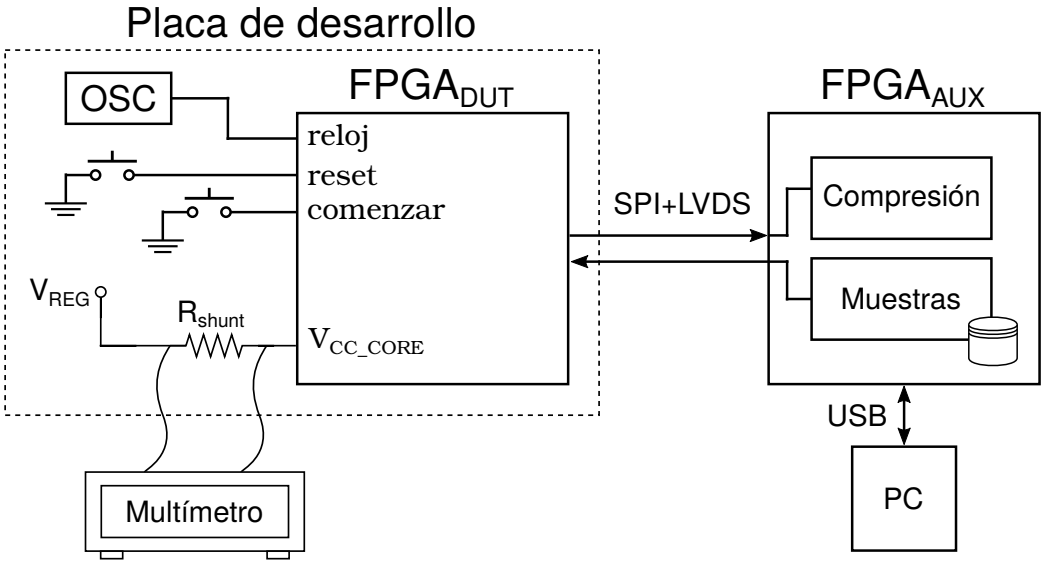

Figura 5.1: Diagrama del *setup* de medidas de consumo. Se midió solamente la corriente consumida por el *Core* de la FPGA que implementa el compresor.

Para cada plataforma y para cada diseño, se realizaron medidas de consumo en dos escenarios. El primero es la operación normal del circuito, de donde se obtiene el consumo total. En el segundo se dejan fijas todas las entradas, en particular la señal de reloj, y lo que se obtiene al medir es el consumo estático. Al bloquear todas las entradas del sistema es imposible que ocurran cambios de estado en los transistores, por lo tanto, el consumo dinámico se anula. Luego, este último puede calcularse restando el consumo estático al total.

En general, los FPGAs poseen entradas de alimentación separadas para el *Core*, que incluye a los elementos lógicos, memorias internas y multiplicadores de hardware; para los bancos de entrada salida, y para los bloques auxiliares (PLLs, DCMs, etc). Como ya se mencionó, en este trabajo se midió únicamente el consumo del *Core* (ver Figura 5.1).

Durante todas las medidas se mantuvo controlada la temperatura del laboratorio entre 22°C y 24°C, ya que el consumo estático aumenta de forma exponencial

con la temperatura, lo que produciría diferencias en las medidas que dificultarían el análisis. En todos los escenarios, previo a realizar la medida se mantiene la operación del chip durante al menos cinco minutos para permitir la estabilización de la temperatura.

5.2. Área y frecuencia máxima

5.2.1. Arquitectura Paralela

En la Tabla 5.1 se muestran la utilización de recursos y la máxima frecuencia de operación de la arquitectura paralela para distintas cantidades de canales. En la FPGA 5CEBA4 entra un máximo de 14 canales utilizando un 84 % de los recursos del chip. En general no se logra una utilización mucho mayor al 80 % debido a la imposibilidad de interconectar todos los elementos (*Routing*). Para sintetizar 21, 31 y 59 canales se debió elegir miembros más grande de la familia. La frecuencia máxima (F_{MAX}) ronda los 70 MHz para todas las implementaciones, lo que permitiría alcanzar tiempos de compresión en el entorno de 0.4 μs.

Nro. Canales	FPGA	ALUTs	Registros	BRAM	Util Lógica	F_{MAX}
14	5CEBA4	26259	11212	7	84 %	72
21	5CEBA5	39503	16841	9	81 %	72
31	5CEBA7	58147	24703	12	61 %	73
59	5CEBA9	110828	46816	17	58 %	77
Por Canal	5CEBA4	1886	802	0	6 %	N/A

Tabla 5.1: Recursos totales y por canal de la arquitectura paralela para distintos FPGA de la familia Cyclone V.

El diseño de la arquitectura paralela no contiene memorias. La utilización de Bloques de memoria RAM (BRAM) en la implementación en las Cyclone V se debe a que la herramienta infiere un *shift–register* en el bloque divisor del canal raíz. Este bloque está basado en bloques de memoria RAM (*altshift–taps*).

5.2.2. Arquitectura Secuencial

En la Tabla 5.2 se muestra la utilización de recursos y la frecuencia máxima de la arquitectura secuencial en las diferentes FPGAs. Esta arquitectura reduce considerablemente los recursos, lo que puede observarse en el bajo porcentaje de utilización obtenido en la 5CEBA4.

Las FPGAs de Lattice tienen un tamaño adecuado para el diseño, por ejemplo, se puede observar que la iCE40HX se acerca al 100 % de utilización para 59 canales. Las máximas frecuencias de operación permiten alcanzar las frecuencias de muestreo deseadas en todos los casos.

A diferencia de la arquitectura paralela, la secuencial sí utiliza memorias en su diseño; estas se encuentran en el FIFO y en la RAM de contexto.

FPGA	Nro. Canales	LUTs	Registros	Bloques BRAM	Tamaño BRAM	Util. Lógica (%)	F_{MAX} (MHz)
5CEBA4	21	2413	911	7	10K	8 %	64.8
5CEBA4	31	2595	899	7	10K	9 %	61.5
5CEBA4	59	2923	935	7	10K	11 %	65.5
iCE40HX	21	4945	1615	18	4K	65 %	45.8
iCE40HX	31	5387	1935	18	4K	71 %	44.5
iCE40HX	59	7092	2854	18	4K	93 %	39.1
MachXO2	21	3341	1652	16	9K	50 %	27.6
MachXO2	31	3458	1971	16	9K	51 %	29.1
MachXO2	59	4424	2946	15	9K	68 %	29.5

Tabla 5.2: Utilización de recursos de la arquitectura secuencial en las tres plataformas para distintas cantidades de canales.

5.3. Consumo de Potencia

5.3.1. Arquitectura Paralela

En la Tabla 5.3 se presentan los resultados de consumo de potencia en función de cantidad de canales para la implementación de la arquitectura paralela en FPGAs de la familia Cyclone V. En todos los casos el consumo es excesivamente alto para los requerimientos planteados, lo que era esperable debido al tamaño del diseño y de la FPGA.

Debido a la naturaleza de esta arquitectura, la máxima frecuencia de muestreo la determina el tiempo de compresión promedio del canal más lento.

FPGA	Nro. Canales	F_{clk} (MHz)	SR_{MAX} (kSps)	P_{TOT} (mW)	P_{DIN} (mW)	P_{EST} (mW)
5CENA4	14	50	1724	104.2	71.2	33.0
5CENA5	21	50	1724	150.1	103.6	46.5
5CENA7	31	50	1923	204.2	153.5	50.7
5CENA9	59	50	1785	316.8	203.3	113.5

Tabla 5.3: Consumo estático, dinámico y total y máxima frecuencia de muestreo de la arquitectura paralela en FPGAs de al familia Cyclone V. Los resultados para 14 canales corresponden a medidas reales, pero el resto fueron estimados utilizando las herramientas del fabricante y luego el resultado fue ajustado en base a la diferencia entre la estimación y la medida de la implementación de 14 canales en la FPGA 5CEBA4.

5.3.2. Arquitectura Secuencial

En la Tabla 5.4 se presentan los resultados de consumo de potencia en función de cantidad de canales para la implementación de la arquitectura secuencial en todas las FPGAs. En la Cyclone V 5CENA4 se logra reducir el consumo dinámico en un

38 % respecto a la arquitectura paralela, pero dado que el estático prácticamente no varía, el consumo total se mantiene alto.

Los FPGAs de Lattice tienen un menor consumo que la Cyclone V, en particular la iCE40HX. Esta última logra incluso resultados que compiten con los del microcontrolador de bajo consumo, comparación que se detalla en la Sección 5.3.3. La MachXO2 tiene un consumo superior a la iCE40HX, no obstante, posee un modo de *stand–by* que le permite trabajar en ciclo de trabajo y reducir considerablemente el consumo, este aspecto también se analiza en la Sección 5.3.3.

FPGA	Nro. Canales	F_{clk} (MHz)	SR_{MAX} (kSps)	P_{TOT} (mW)	P_{DIN} (mW)	P_{EST} (mW)
5CENA4	21	50	88.18	59.9	27.1	32.7
5CENA4	31	50	64.52	65.1	32.3	32.7
5CENA4	59	50	31.39	63.2	30.4	32.7
MachXO2	21	6.05	10.7	21.6	13.4	8.2
MachXO2	21	12.09	21.3	26.6	18.5	8.2
MachXO2	21	22.17	39.1	35.0	26.8	8.2
MachXO2	31	6.05	7.8	20.8	12.7	8.2
MachXO2	31	12.09	15.6	26.6	18.4	8.2
MachXO2	31	22.17	28.6	35.4	27.2	8.2
MachXO2	59	6.05	3.8	22.1	13.9	8.2
MachXO2	59	12.09	7.6	27.8	19.7	8.2
MachXO2	59	22.17	13.9	37.6	29.5	8.2
iCE40HX	21	6	10.6	2.99	1.67	1.32
iCE40HX	21	12	21.2	4.60	3.29	1.31
iCE40HX	31	6	7.7	3.24	1.92	1.32
iCE40HX	31	12	15.5	4.90	3.56	1.33
iCE40HX	59	6	3.8	3.43	2.05	1.38
iCE40HX	59	12	7.5	4.98	3.62	1.36

Tabla 5.4: Consumo estático, dinámico y total y máxima frecuencia de muestreo de la arquitectura secuencial en todas las FPGAs utilizadas. En cada FPGA se midió el consumo para 21, 31 y 59 canales y diferentes frecuencias de operación.

En esta arquitectura, el tiempo de compresión de un *set* de muestras se obtiene multiplicado el tiempo de compresión promedio por la cantidad de canales. Las máximas frecuencias de muestreo (SR_{MAX}) alcanzables en las FPGAs superan con creces las del microcontrolador. Por ejemplo, para los tres números de canales estudiados, la MachXO2 alcanza velocidades de compresión entre 11x y 12x veces más rápidas. La iCE40HX no logra los mismos tiempos de compresión que la MachXO2, pero igualmente es más de 6x veces más rápida que el microcontrolador en todos los casos. Adicionalmente, esta FPGA logra las mejoras en velocidad manteniendo los niveles de consumo de potencia similares a los del microcontrolador, lo que la hace notoriamente más eficiente.

No se presentan resultados en la iCE40HX para 22 MHz de frecuencia, debido a que en dicho caso el circuito no funcionó como se esperaba, es decir, la salida comprimida no era correcta. Según los datos de frecuencia máxima provistos por la

herramienta (ver Tabla 5.2), el diseño debería funcionar a 22 MHz. Es de suponer que en realidad el circuito no puede funcionar a tan alta frecuencia, y la herramienta estima incorrectamente el valor. Por ejemplo, la MachXO2 es una FPGA con similares características de velocidad que la iCE40HX y la frecuencia máxima reportada por la herramienta, para este diseño, es bastante menor. No se descarta que el problema venga de un error de diseño, por ejemplo, por una definición insuficiente de las restricciones (*constraints*) o una ineficiente configuración de los parámetros de la herramienta. Sin embargo, esto no se pudo determinar, mayoritariamente porque el entorno de desarrollo *icecube2* es realmente muy pobre y poco amigable, lo que dificulta el diseño y, en especial, el *debugging*.

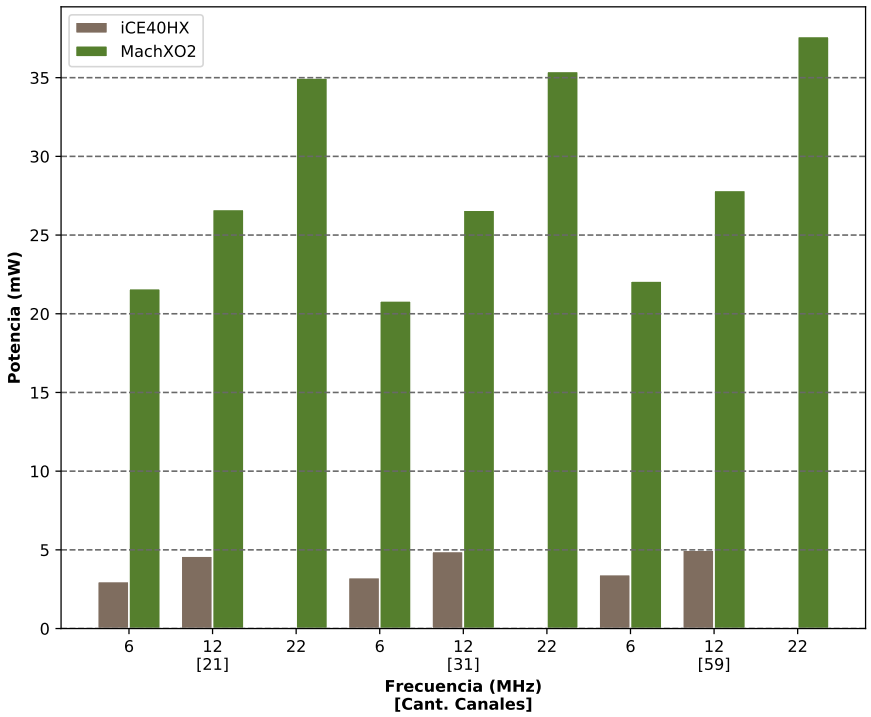

Figura 5.2: Consumo total de la arquitectura secuencial en las FPGAs iCE40HX y MachXO2 para distintas frecuencias de reloj y cantidades de canales.

En la Figura 5.2 se comparan los resultados de consumo de la arquitectura secuencial en las FPGAs de Lattice. A primera vista llama la atención la notoria diferencia entre ambas y, en particular, el alto consumo dinámico de la MachXO2 (ve tabla 5.4). Si bien esta FPGA tiene una mayor orientación a la alta performance que la iCE40HX, ambas son FPGAs catalogadas como de bajo consumo. Una parte importante del elevado consumo dinámico se debe a que la MachXO2 utiliza como fuente de reloj un oscilador integrado, a diferencia de las demás FPGAs que reciben la señal de un cristal externo alimentado con una fuente independiente. No se hallaron datos de consumo del oscilador en los documentos provistos por el fabricante. Lo único que se encontró fue el trabajo publicado en [47], donde se reporta un consumo del oscilador de 8 mA (a $V_{CC} = 1.2V$) para otra FPGA de la

familia MachXO2.

5.3.3. Consumo de potencia vs Throughput

Salvo la MachXO2, las demás FPGAs utilizados no poseen modos de bajo consumo. Esto implica que deben permanecer siempre encendidas aunque no estén procesando datos. Por lo tanto, para que la comparación sea justa se midió el consumo de las plataformas con los FPGAs operando a su máxima velocidad de compresión, la cual varía según la frecuencia de reloj. En el caso del microcontrolador, que permite pasar a modos de bajo consumo, se utilizan las medidas tomadas operando en ciclo de trabajo. En la Figura 5.3 se muestra dicha comparación, teniendo en el eje y la potencia total consumida y en el eje x el *throughput* a la entrada, calculado como la cantidad de canales por la cantidad de bits por muestra dividido el tiempo de compresión promedio. Es interesante observar como la FPGA iCE40HX logra consumos similares al microcontrolador (en la grafica se ve que es menor, pero sumando el consumo de la entrada/salida al del *Core*, lo superaría en algunos casos). Por otro lado, para los mismos consumos la iCE40HX logra *throughputs* entre dos y tres órdenes de magnitud superiores.

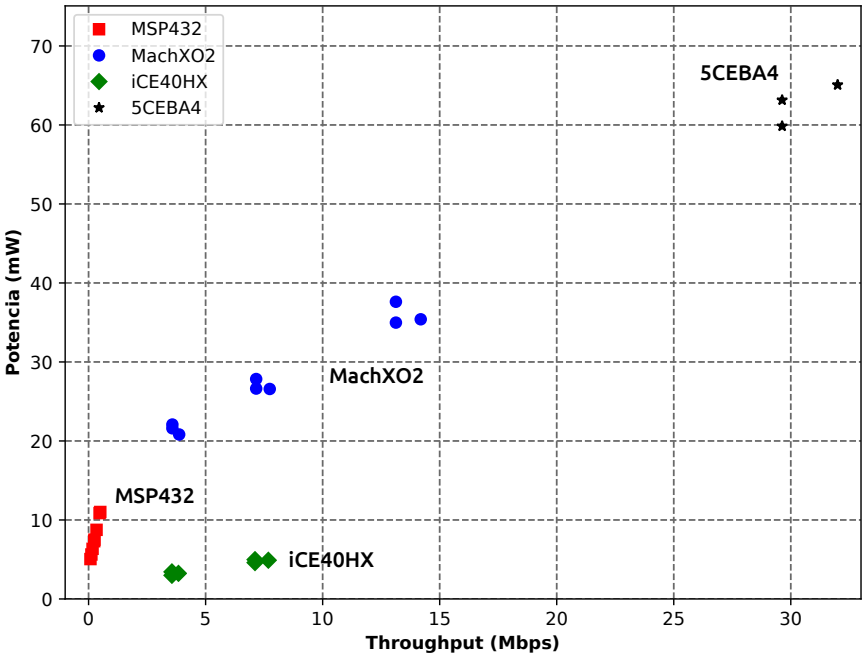

Figura 5.3: Potencia en función de throughput para todas las plataformas.

La implementación en microcontrolador alcanza frecuencias de muestreo considerablemente menores que las FPGAs, por otro lado, estas tienen un consumo de base más alto que el microcontrolador. Esto dificulta la comparación entre ambas plataformas, ya que, en el rango de frecuencias de muestreo que permite el microcontrolador, la FPGA tiene un consumo mayor. Una alternativa para evaluar

las implementaciones es utilizar una medida de eficiencia en el consumo, definida como el cociente entre la performance y la potencia consumida. En este caso se fija la cantidad de canales y se establece como performance la máxima frecuencia de muestreo. Esto es:

$$\text{Eficiencia} = \frac{\text{Máxima Frecuencia de Muestreo}}{\text{Potencia}} \quad \left[\frac{kSps}{W}\right] \tag{5.1}$$

donde valores más grandes son mejores.

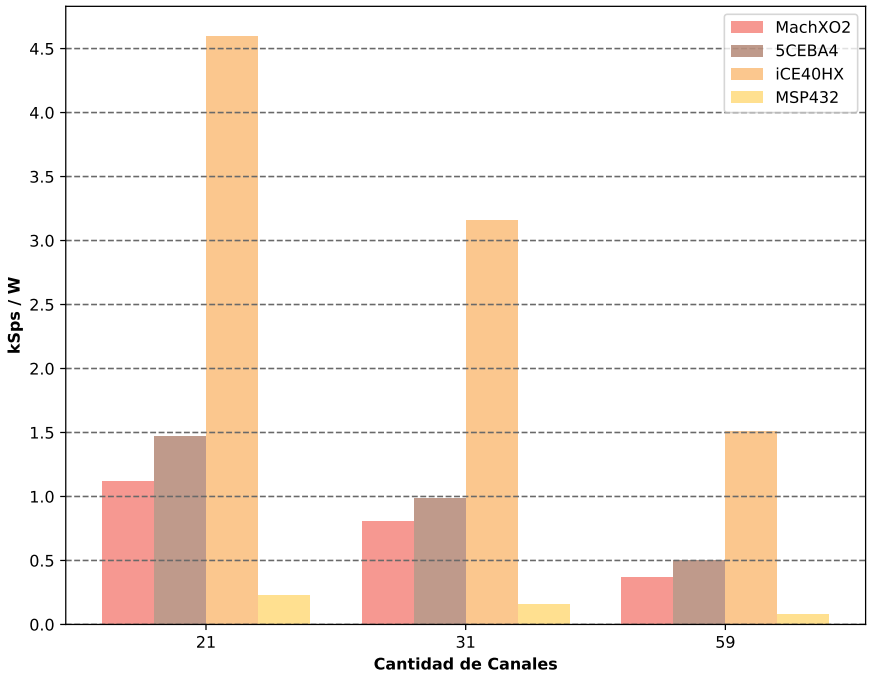

Figura 5.4: Comparación de eficiencia de consumo entre MachXO2, iCE40HX, MSP432 y 5CEBA4. Valores más grandes son mejores.

Como se puede observar en la Figura 5.4, bajo esta métrica las FPGAs obtienen una clara ventaja sobre el microcontrolador. Esto significa que la implementación en FPGAs es preferible cuando el sistema requiere altas frecuencias de muestreo. Incluso la Cyclone V 5CEBA4 posee mejor eficiencia de potencia que el MSP432 si se opera a altas frecuencias (50 MHz). Pero por más eficiente que sea el consumo, está un orden de magnitud por encima y, por lo tanto, deja de ser adecuado para un dispositivo de bajo consumo. La iCE40HX logra excelentes resultados, ya que permite operar a altas frecuencias de muestreo pero su consumo es muy cercano al obtenido con el microcontrolador.

En el caso de la MachXO2 es posible realizar una la comparación para bajas frecuencias de muestreo, ya que dicha FPGA posee un modo de *stand–by* en el que reduce de forma importante su consumo. Esto permite operar en un ciclo de trabajo, donde la FPGA se activa para procesar los datos lo más rápido posible y luego se pasa al modo de bajo consumo. La Tabla 5.5 muestra la potencia consumida por

la MachXO2 en los dos modos y el ahorro que puede obtenerse si se opera a bajas frecuencias de muestreo usando un ciclo de trabajo. En la Figura 5.5 se muestra el consumo del microcontrolador y de la FPGA operando en ciclo de trabajo para distintos *throughputs*. Puede observarse que a partir de aproximadamente 450 kbps el FPGA pasa a consumir menos que el microcontrolador. Esta comparación no incluye el consumo de la entrada/salida en la FPGA, por lo tanto, para los valores medidos el microcontrolador tendría menor consumo total. Esto no invalida la reflexión anterior, simplemente aumenta el umbral de *throughput* para el cual se cumple.

Nro. Canales	$P_{Standby}$ (mW)	P_{ON} (mW)	t_{comp} (us)	SR (Sps)	Duty Cycle (%)	P_{TOT} (mW)	Reduc. (%)
21	9.14	21.6	93.8	250	2.34	9.4	56
21	9.14	26.6	46.9	500	2.34	9.6	64
21	9.14	35.0	25.6	1000	2.56	9.8	72
31	9.25	20.8	128.2	250	3.21	9.6	54
31	9.25	26.6	64.1	500	3.21	9.8	63
31	9.25	35.4	35.0	1000	3.50	10.2	71
59	9.36	22.1	263.5	250	6.59	10.2	54
59	9.36	27.8	131.8	500	6.59	10.6	62
59	9.36	37.6	71.9	1000	7.19	11.4	70

Tabla 5.5: Ahorro de energía obtenido en la FPGA MachXO2 utilizando el modo Stand–by para operar en ciclos de trabajo.

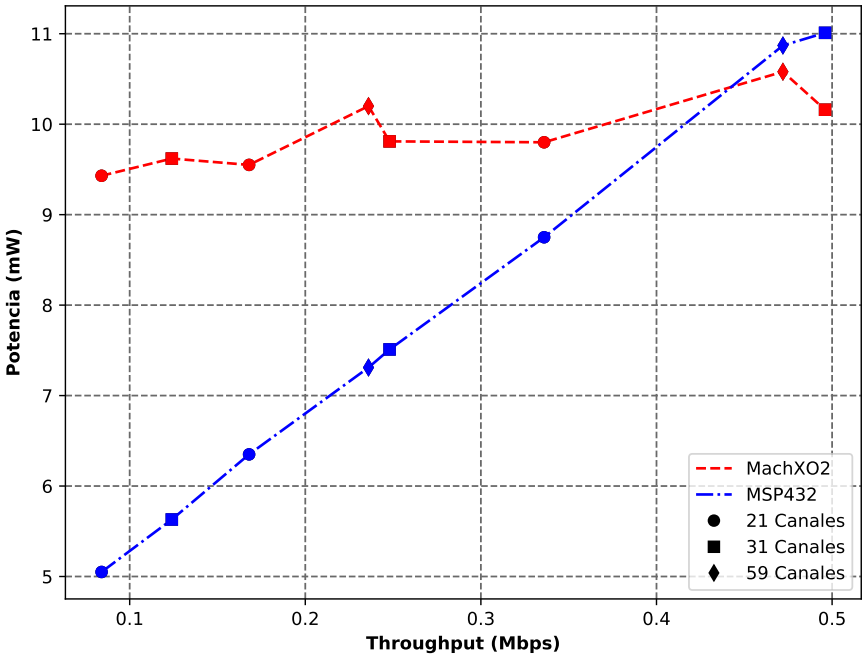

Figura 5.5: Potencia consumida en función del throughput para el MSP432 y la MachXO2 a frecuencia de muestreo fija y operando con un ciclo de trabajo. El símbolo en cada punto del gráfico indica la cantidad de canales.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Sistema Completo

Los capítulos anteriores se centraron en investigar si, para la implementación del algoritmo de compresión propuesto en [15], las FPGAs logran superar en performance al desarrollo en microcontrolador, sin aumentar el consumo de potencia. Habiendo concluido que bajo determinadas circunstancias lo anterior se cumple, se aborda la siguiente etapa del diseño, que involucra el desarrollo de un sistema completo. Esta es una de las principales motivaciones del presente trabajo, donde se busca avanzar hacia la construcción de un sistema vestible para la adquisición en tiempo real de señales de EEG, provenientes de múltiples canales y adquiridas a altas frecuencias de muestreo.

Se busca que el diseño completo pueda ser implementado completamente en un único chip, dejando fuera los dos componentes que no son fácilmente integrables: el AFE y ADC y la Radio. En este trabajo, la plataforma tecnológica elegida es las FPGA, pero dado que el sistema fue diseñado casi en su totalidad en HDL genérico, este podría ser implementado en un ASIC digital. Esta alternativa resulta interesante si se desea mejorar aún más la performance y el consumo de energía.

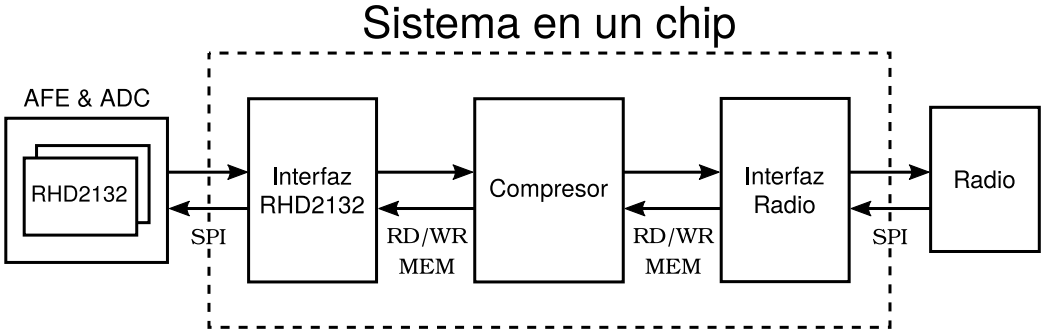

Figura 6.1: Diagrama de los tres bloques que conforman el sistema completo. Entre líneas punteadas se muestran los diseños realizados en este trabajo.

El sistema completo, representado en la Figura 6.1, se basa en tres bloques. Por un lado, el AFE y ADC se encarga de filtrar, amplificar y adquirir (convertir a digital) las señales de EEG provenientes de los electrodos. Luego, el bloque central del diseño (sistema en un chip) realiza el procesamiento de datos y el control

general del sistema. Este consta de tres partes: el Compresor, que fue detallado extensamente a lo largo del documento; la Interfaz RHD2132, que se encarga de la comunicación y el control del AFE y ADC, y la interfaz con la Radio. El tercer bloque es la Radio, encargada de la transmisión de los datos comprimidos a un receptor externo.

Este capítulo tiene como objetivo describir los restantes componentes del sistema que, junto al compresor, conforman el diseño completo.

6.1. Placa AFE y ADC

En el proyecto de fin de carrera wEEG [31], realizado en el marco del proyecto CSIC I+D "Electroencefalógrafo inalámbrico de bajo consumo de energía", se diseñó y fabricó un PCB que incorpora dos integrados RHD2132 de Intan Technologies. Estos son dispositivos de ultra bajo consumo, específicamente diseñados para la adquisición de señales biológicas. Presenta una amplificación de bajo ruido, ancho de banda programable, filtros analógicos y digitales, y un ADC de 16 bits. Cada chip presenta 32 canales analógicos más tres canales auxiliares (que se utilizan principalmente para inyectar señales de sincronismo, o como canales analógicos adicionales). El ADC tiene una frecuencia máxima de 1.05 MSps, lo que le permite adquirir 35 canales a 30 kSps/s cada uno. La interfaz de comunicación del chip es SPI esclavo. Un detalle interesante es que el dispositivo no posee reloj propio y opera en función del reloj SPI maestro, siendo esta una de las causas de su reducido consumo de energía. Esta forma de funcionamiento implica que la frecuencia de conversión (muestreo) está dada por la velocidad de la transmisión SPI.

6.2. Interfaz para RHD2132

Para hacer uso del desarrollo mencionado en la Sección 6.1, se diseñó un circuito que implementa la comunicación y control de dos integrados RHD2132 y el manejo de las muestras adquiridas. El sistema permite modificar la configuración interna de los chips y enviar comandos de conversión (adquirir muestras) según la cantidad de canales elegida, a una frecuencia de muestreo configurable. Es capaz de adquirir las muestras a aproximadamente 1 MSps/s, que es la máxima frecuencia permitida por el chip. Este valor puede ser alcanzado para cualquier cantidad de canales, por ejemplo, se pueden muestrear 32 canales a 30 kSps/s o cuatro canales a 250 kSps/s. El diseño se implementó en la FPGA Cyclone V 5CEBA4 de Intel.

El sistema, representado en el diagrama de la Figura 6.2, está constituido por los siguientes bloques:

- Registros que almacenan la configuración del diseño (registros del sistema).
- Registros que almacenan la configuración interna de los chips RHD2132 (registros de parámetros).
- Dos cores SPI maestro (interfaz SPI).

- Una máquina de control que selecciona los comandos a enviar en función de la configuración y realiza el manejo de las muestras adquiridas.
- Dos memorias tipo FIFO para almacenar el resultado de la conversión.
- Interfaz Wishbone para el manejo del sistema.

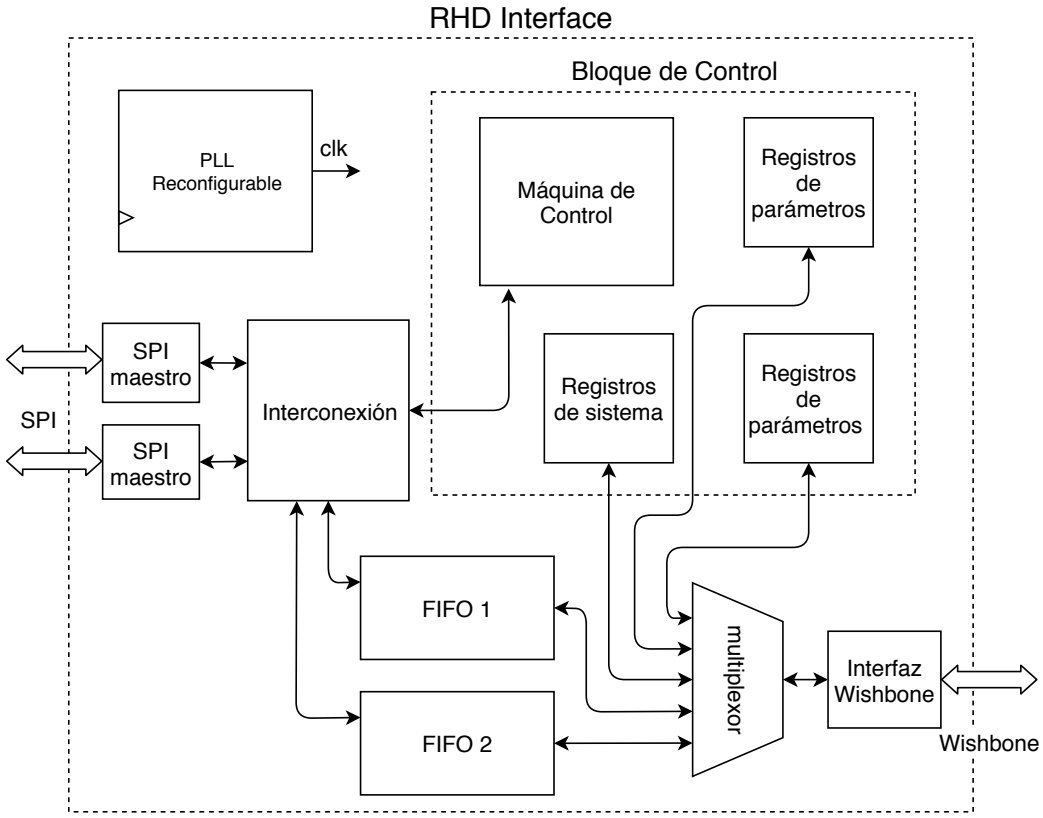

Figura 6.2: Diagrama del sistema para el manejo de dos integrados RHD2132.

Respecto a la configuración, se implementó en el diseño un espejo de todos los registros de configuración de cada integrado (registros de parámetros). De esta forma, el usuario puede configurar a elección todos los parámetros disponibles, de forma independiente para cada chip. Se cuenta con un mecanismo que sobreescribe todos los registros internos del chip con el contenido de los registros de parámetros (que deben ser previamente cargados por el operador). Si se desea realizar una operación puntual, como la escritura o lectura de un único registro interno, el sistema es capaz de enviar un único comando ingresado manualmente.

En la carga de los parámetros y en la adquisición de las muestras, el sistema funciona de forma simétrica para los dos integrados, esto implica que los parámetros de configuración (e.g. frecuencias de corte de los filtros) se cargan en ambos chips en simultáneo y la cantidad de canales a convertir es la misma para los dos. Esto no es así para la funcionalidad de enviar un único comando, la cual puede realizarse de manera independiente para uno.

En el modo de adquisición de muestras se envían los comandos de conversión uno a continuación del otro, con un tiempo de espera entre cada uno. La cantidad de períodos de reloj que toma cada ciclo SPI es fija, por lo tanto, variando la frecuencia de operación del sistema es que se logran las distintas frecuencias de muestreo. Para esto el diseño posee un bloque PLL reconfigurable en operación, el cual es dependiente del fabricante del FPGA. Por lo tanto, si se cambia la plataforma es necesario adaptarlo este bloque. Si bien la mayoría de los FPGAs poseen bloques PLL o similares, no todos permiten la modificación de sus parámetros en tiempo de ejecución.

El diseño incluye una interfaz Wishbone [48] que permite configurar los registros del sistema, enviar comandos y leer las muestras adquiridas. Se optó por utilizar esta interfaz durante las etapas de verificación del diseño porque facilitaba las pruebas, ya que permite abrir una conexión entre el PC y la placa a través de una interfaz JTAG–USB. Para implementar este diseño junto al compresor se deben quitar los bloques Wishbone y utilizar los FIFOs como interfaz entre ambos diseños. Del lado del compresor es necesario adaptar los mecanismos de control para manejar la Interfaz RHD2132 y la lectura de los FIFOs.

FPGA	LUTs	Registros	Bloques BRAM	Tamaño BRAM	Util. Lógica (%)	F_{MAX} (MHz)
5CEBA4	1192	812	65	10K	4 %	127

Tabla 6.1: Resultados de utilización de recursos y frecuencia máxima del diseño para manejar los integrados RHD2132 implementado en la FPGA 5CEBA4.

En la Tabla 6.1 se presentan los resultados de utilización de recursos y frecuencia máxima del diseño implementado en la FPGA 5CEBA4. En términos de elementos lógicos es un diseño pequeño, que entraría sin problemas junto al compresor en todas las implementaciones estudiadas. La gran cantidad de bloques de memoria utilizados se debe a la profundidad de los FIFOs, que fueron agregados para verificar el funcionamiento del diseño en la placa, que se realizó mediante la lectura de las muestras adquiridas desde un PC a través de una interfaz JTAG–USB. Dado que dicha interfaz es lenta, es necesario almacenar las muestras en los FIFOs para no perderlas. En el funcionamiento junto al compresor, estos FIFOs podrían ser mucho menores, ya que las muestras se consumen igual o más rápido de lo que se adquieren.

6.3. Comunicación inalámbrica

En la Sección 2.3.2 se mostró que el algoritmo logra excelentes ratios de compresión en señales de EEG multicanal. Pero, tanto los pesos que ponderan los predictores como el parámetro de Golomb son adaptivos y su valor depende de las muestras anteriores. Por lo que un cambio repentino en las señales de entrada produciría un aumento en el error de predicción y causaría que la codificación no

sea óptima. Estos casos, que en promedio no deberían tener un efecto considerable el ratio de compresión, por momentos pueden producir que el algoritmo aumente el tamaño de las muestras en lugar de reducirlo. La solución al problema sería dejar de utilizar el algoritmo y enviar directamente las muestras sin procesar hasta que se estabilicen los parámetros. Esto implica que la comunicación inalámbrica tiene que ser capaz de soportar rachas de transmisión de muestras sin comprimir, lo que puede resolverse aumentando el tamaño de los *buffers* o el ancho de banda de la transmisión. Considerando las altas tasas de datos, parece más razonable la segunda opción.

Considerando solamente los resultados obtenidos con las FPGAs de bajo consumo, la mayor tasa de datos a la entrada se da en la configuración de 31 canales, 16 bits por muestra y una frecuencia de muestreo de 28.6 kSps. En este caso el *throughput* resultante es:

$$31 \times 16 \text{ bits} \times 28.6 \text{ kSps} = 14.2 \text{ Mbps} \tag{6.1}$$

Manejar estos niveles de *throughput* significa descartar los protocolos de comunicación inalámbrica de bajo consumo, como Bluetooth Low Energy (algunos Mbps) o los basados en el estándar IEEE 802.15.4 (cientos de kbps).

Asumiendo que la utilización del dispositivo se restringe a espacios limitados, como podría ser el hogar de un paciente que se esté realizando un estudio, el alcance de la comunicación estaría limitado a decenas de metros cuadrados. Sumando esto a los requerimientos de velocidad de transmisión, una opción viable es utilizar Wi–Fi (802.11x). Este protocolo tiene además la ventaja de ser de uso masivo, lo que abarata los costos de equipamiento y facilita la implantación.

Una alternativa podría ser utilizar redes celulares (3G/4G/5G), pero esto incurriría en costos adicionales. También existen versiones de Bluetooth que ofrecen velocidades de transferencia del orden de decenas de Mbps, pero en realidad utilizan protocolos Bluetooth para establecer la conexión y luego la transferencia de datos la resuelven mediante el protocolo 802.11.

Existen múltiples versiones del protocolo 802.11 con diferentes características, así como también existe una inmensa cantidad de dispositivos que los implementan. De todas formas, no es el objetivo de este trabajo evaluarlos al detalle ni determinar la mejor alternativa, sino simplemente presentar algunos ejemplos para dar una idea de la viabilidad de la implementación y del consumo de potencia que añadiría al sistema.

A modo de ejemplo, en la Tabla 6.2 se presentan algunos módulos Wi–Fi orientados a aplicaciones de bajo consumo. En la cuarta columna se muestra el consumo del transmisor en modo activo para situaciones de alto throughput (mostrado en la sexta columna), y en la quinta columna la corriente en modo de bajo consumo (*sleep*).

El consumo de Wi–Fi va a depender de múltiples factores, como el esquema de modulación y codificación utilizado, o el ancho del canal [49]. Pero un factor predominante es el throughput de la transmisión/recepción [50]. Considerando el alto consumo energético que posee este protocolo, se pueden lograr importantes reducciones en la energía de transmisión aplicando compresión de datos. Especial-

Fabricante	Dispositivo	Protocolo	I_{TX}@3.3V (mA)	I_{Sleep}@3.3V (μA)	Bit Rate (Mbps)
Espressif	ESP8266	b/g/n	135	10	65
Microchip	RN171	b/g	240	4	24
Microchip	ATWILC1000	b/g/n	186	380	65
Maxchip	EMW3161	b/g/n	260	200	65
Texas Instruments	CC3100	b/g/n	160	4	54
Silicon Labs	AMW007	b/g/n	150	10	65

Tabla 6.2: Ejemplo de modulos Wi–Fi orientados a aplicaciones de bajo consumo alimentadas a batería.

mente considerando en sacar provecho de la gran diferencia entre el consumo activo y en *sleep* de los módulos Wi–Fi presentados.

Respecto a la integración con el compresor, en general los módulos como los que se presentan en la Tabla 6.2 poseen inteligencia propia (microcontrolador y memorias embebidas) e implementan la mayoría de las capas de los protocolos que utilizan. Esto significa que el bloque Interfaz Radio pasa a ser meramente una interfaz serie (SPI, UART, etc) entre el compresor y la Radio. Sería necesario además dotar al control general del sistema la capacidad de configurar el módulo de Radio.

Capítulo 7

Conclusiones

Se presentó una implementación en hardware del algoritmo de compresión propuesto en [15], como parte central de un sistema de EEG inalámbrico de bajo consumo. Se propusieron dos arquitecturas, una que explota de forma directa el paralelismo de las FPGAs y otra que hace uso de la reutilización de recursos para reducir el área y el consumo de potencia. Resultados preliminares del diseño fueron presentados en la conferencia *SPL 2019* en Buenos Aires, Argentina [51]. Los diseños fueron implementados en tres FPGAs: la Cyclone V 5CEBA4 (Intel), la iCE40HX y la MachXO2 (ambas de Lattice). Los resultados obtenidos fueron comparados con la implementación del algoritmo en una plataforma de bajo consumo basada en el microcontrolador MSP432 de Texas Instruments.

Los diseños fueron caracterizados para 21, 31 y 59 canales y distintas frecuencias de operación. La verificación se llevó a cabo, primero mediante simulaciones y luego programando el diseño en los FPGAs. En todos los casos se utilizaron muestras reales de EEG provenientes de bases de datos públicas, y se obtuvieron los mismos ratios de compresión que en la versión de software del algoritmo. En la arquitectura secuencial, que es la más atractiva en términos de consumo energético, todos los FPGAs logran importantes mejoras en la velocidad de compresión. La MachXO2 es entre 11 y 12 veces más rápida que la implementación en microcontrolador, alcanzando frecuencias de muestreo en el entorno de 13 kSps para 59 canales. La iCE40 permite utilizar frecuencias de muestreo alrededor de 7 kSps, y logra incrementos respecto al MSP432 de aproximadamente 6×. La 5CEBA4 es aún más rápida que las otras dos, pero debido al alto consumo de potencia no resulta demasiado adecuada para la aplicación.

Respecto a las medidas de potencia, en todos los casos se midió únicamente el consumo del *Core* de las FPGAs. Los mejores resultados se obtuvieron con la FPGA iCE40HX, los cuales se encuentran entre 3 mW y 5 mW. El consumo de la entrada/salida no fue medido, pero se estima que va a ser del orden del consumo del *Core* debido al protocolo (LVDS) y a la cantidad puertos utilizados. Por lo tanto, sumando ambas potencias aún se está en valores similares al consumo del microcontrolador, que varía entre 5 mW y 11 mW. En base a los resultados de performance y consumo obtenidos para esta plataforma, se puede concluir que es adecuada para su utilización en sistemas de bajo consumo que manejen altas tasas

de datos, como es el caso de un EEG inalámbrico.

La potencia consumida en la FPGA MachXO2 oscila entre 21 mW y 38 mW, números bastante altos para un FPGA de bajo consumo, en particular la componente de consumo dinámico, que en promedio es un 70 % del total. Sin embargo, al poseer un modo *stand–by* permite la operación en ciclos de trabajo y logra reducciones en consumo que lo acercan a los niveles del microcontrolador.

El diseño con la arquitectura paralela no es recomendable para un sistema inalámbrico, ya que posee un alto consumo de potencia, producto de la necesidad de utilizar FPGAs de gran tamaño. Sin embargo, sería interesante explorar este diseño en otro tipo de aplicaciones que requieran la transmisión en tiempo real de señales biológicas multicanal con altas tasas de datos (orden de Mbps), y que se beneficien de la compresión de datos.

Avanzando hacia un sistema completo, se diseña en HDL una interfaz para el manejo del AFE, el cual está basado en dos integrados RHD2132 de Intan. Se verificó el diseño mediante simulaciones y pruebas reales con la FPGA 5CEBA4, logrando manejar los chips a su máxima frecuencia de muestreo (alrededor de 1 MSps). Los bajos recursos utilizados por el diseño permiten que este sea implementado junto al compresor en cualquiera de los FPGAs utilizados.

A simple vista, la mejora en performance que ofrecen los FPGAs resulta muy atractiva. Más aún considerando que en uno de los casos estudiados, el consumo de potencia se mantuvo en valores similares a los obtenidos en el microcontrolador. Sin embargo, cabe señalar que el esfuerzo requerido en el desarrollo con FPGAs, y hardware en general, es enormemente superior al de un microcontrolador. Para un único diseñador no experto, un desarrollo pequeño como el aquí presentado requiere varios meses de trabajo. El aspecto que probablemente más tiempo consuma es la verificación y corrección de errores, especialmente cuando estos requieren cambios de arquitectura que implican grandes modificaciones en todo el diseño. Se resalta la importancia de definir de la forma más completa posible la arquitectura del sistema, incluyendo todas las tareas o bloques y sus interfaces.

A modo de resumen, se listan los aportes realizados en el marco de esta tesis:

- Se diseñó un hardware en HDL genérico que realiza la compresión de señales de EEG utilizando el algoritmo descrito en [15]. Se optimizó el diseño en consumo de potencia pero respetando las restricciones de *throughput* de entrada.

- Se caracterizaron la performance y el consumo de potencia de la implementación en tres FPGAs. En el caso de la iCE40HX se logran resultados que claramente superan a la implementación del algoritmo en un microcontrolador, debido a que se alcanzan considerables aumentos en la velocidad, sin incrementar el consumo de potencia.

- Se demostró mediante un ejemplo la ventaja de las FPGAs en aplicaciones de bajo consumo de energía, con altos requerimientos de procesamiento y manejo de datos.

- Se diseñó un circuito para el manejo de dos integrados RDH2132 de Intan,

que junto al compresor y una radio inalámbrica completan el sistema de adquisición de señales de EEG inalámbrico.

7.1. Publicaciones asociadas a la tesis

La siguiente publicación fue presentada en el congreso *SPL 2019* en Buenos Aires, Argentina:

- **F. Favaro** and J. P. Oliver, "Hardware implementation of a multi-channel eeg lossless compression algorithm," in *2019 X Southern Programable Logic (SPL)*. IEEE, 2019, pp. 69-73 [51].

Los siguientes trabajos, que fueron citados a lo largo del documento, están relacionados con el trabajo de tesis:

- G. Dufort, **F. Favaro**, F. Lecumberry, Á. Martín, J. P. Oliver, J. Oreggioni, I. Ramírez, G. Seroussi, and L. Steinfeld, "Wearable eeg via lossless compression," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2016, pp. 1995–1998 [14].

- G.D.y Alvarez, **F. Favaro**, F. Lecumberry, A. Martín, J. P. Oliver, J. Oreggioni, I. Ramírez, G. Seroussi, and L. Steinfeld, "Wireless EEG system achieving high throughput and reduced energy consumption through lossless and near–lossless compression," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 231–241, Feb. 2018 [15].

7.2. Trabajo futuro

En primer lugar se desea agregar una reflexión adicional a lo expresado sobre el esfuerzo requerido en el diseño de hardware. Al comienzo del trabajo de tesis, el autor no contaba con mayor experiencia que la obtenida en un curso básico de diseño digital en FPGAs. Durante el transcurso de la tesis se fue adquiriendo experiencia y conocimiento sobre el tema, lo cual mejoró progresivamente la calidad de los diseño realizados. Naturalmente, ya sobre el final del trabajo se identificaron múltiples aspectos de los diseños, especialmente en las partes elaboradas en etapas iniciales, en las cuales hay bastante lugar para mejorar. Por lo tanto, sería deseable como trabajo a futuro mejorar los diseños y, en particular, aplicar más técnicas para optimizar el consumo de energía.

El siguiente paso es finalizar la integración de todas las partes que conforman el sistema completo. Esto implica adaptar las interfaces entre el compresor y los restantes bloques, y adquirir un módulo de comunicación inalámbrica adecuado. Una vez logrado el prototipo y verificado su funcionamiento, sería deseable probarlo con señales de FPGA reales.

Respecto a las plataformas utilizadas, ambas FPGAs de Lattice están orientadas a aplicaciones de bajo consumo de energía. Sin embargo, dentro de la familia a

la que pertenece cada una, existen otros modelos de menor consumo. Sería interesante probar el diseño en dichos modelos y ver si se logra reducir dicha variable sin perder en performance.

Aprovechando que el diseño fue realizado en lenguajes de descripción de hardware genéricos, otro aspecto interesante a explorar es la implementación del sistema completo en un ASIC digital usando celdas estándar. Esto permitiría incrementar considerablemente la performance y reducir el consumo de energía.

Referencias

[1] A. J. Casson, D. C. Yates, S. J. M. Smith, J. S. Duncan, and E. Rodriguez-Villegas, "Wearable electroencephalography," *IEEE Engineering in Medicine and Biology Magazine*, vol. 29, no. 3, pp. 44–56, May 2010.

[2] C. E. Elger and C. Hoppe, "Diagnostic challenges in epilepsy: seizure under-reporting and seizure detection," *The Lancet Neurology*, vol. 17, no. 3, pp. 279–288, 2018.

[3] L. Kuhlmann, K. Lehnertz, M. P. Richardson, B. Schelter, and H. P. Zaveri, "Seizure prediction—ready for a new era," *Nature Reviews Neurology*, p. 1, 2018.

[4] A. I. Sburlea and G. R. Müller-Putz, "Exploring representations of human grasping in neural, muscle and kinematic signals," *Scientific reports*, vol. 8, no. 1, p. 16669, 2018.

[5] I. Iturrate, J. M. Antelis, A. Kubler, and J. Minguez, "A noninvasive brain-actuated wheelchair based on a p300 Neurophysiological protocol and automated navigation," *IEEE Transactions on Robotics*, vol. 25, no. 3, pp. 614–627, Jun. 2009.

[6] D. Erdogmus, A. Adami, M. Pavel, T. Lan, S. Mathan, S. Whitlow, and M. Dorneich, "Cognitive state estimation based on eeg for augmented cognition," in *Conference Proceedings. 2nd International IEEE EMBS Conference on Neural Engineering, 2005.* IEEE, 2005, pp. 566–569.

[7] J. van Erp, F. Lotte, and M. Tangermann, "Brain-computer interfaces: Beyond medical applications," *Computer*, vol. 45, no. 4, pp. 26–34, Apr. 2012.

[8] V. Mihajlović, B. Grundlehner, R. Vullers, and J. Penders, "Wearable, wireless EEG solutions in daily life applications: What are we missing?" *IEEE Journal of Biomedical and Health Informatics*, vol. 19, no. 1, pp. 6–21, Jan. 2015.

[9] N. Usui, K. Terada, K. Baba, K. Matsuda, F. Nakamura, K. Usui, T. Tottori, S. Umeoka, S. Fujitani, T. Mihara *et al.*, "Very high frequency oscillations (over 1000 hz) in human epilepsy," *Clinical Neurophysiology*, vol. 121, no. 11, pp. 1825–1831, 2010.

[10] F. Zhang, J. Holleman, and B. P. Otis, "Design of ultra-low power biopotential amplifiers for biosignal acquisition applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 4, pp. 344–355, Aug. 2012.

[11] S. Bluetooth, "Bluetooth core specification v5. 0," *Bluetooth Special Interest Group: Kirkland, WA, USA*, 2016.

[12] A. M. Dixon, E. G. Allstot, D. Gangopadhyay, and D. J. Allstot, "Compressed sensing system considerations for ecg and emg wireless biosensors," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 6, no. 2, pp. 156–166, 2012.

[13] I. Capurro, F. Lecumberry, A. Martín, I. Ramírez, E. Rovira, and G. Seroussi, "Efficient sequential compression of multichannel biomedical signals," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 4, pp. 904–916, Jul. 2017.

[14] G. Dufort, F. Favaro, F. Lecumberry, Á. Martín, J. P. Oliver, J. Oreggioni, I. Ramírez, G. Seroussi, and L. Steinfeld, "Wearable eeg via lossless compression," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2016, pp. 1995–1998.

[15] G. D. y. Álvarez, F. Favaro, F. Lecumberry, A. Martín, J. P. Oliver, J. Oreggioni, I. Ramírez, G. Seroussi, and L. Steinfeld, "Wireless EEG system achieving high throughput and reduced energy consumption through lossless and near-lossless compression," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 231–241, Feb. 2018.

[16] "Information Technology–Coding of Audio-Visual Objects–Part 3: Audio, 3rd Ed. Amendment 2: Audio Lossless Coding (ALS), New Audio Profiles and BSAC Extensions," *ISO/IEC 14496-3:2005/Amd.2:2006*, 2006.

[17] Y. Kamamoto, N. Harada, and T. Moriya, "Interchannel dependency analysis of biomedical signals for efficient lossless compression by mpeg-4 als," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*. IEEE, 2008, pp. 569–572.

[18] Y. Wongsawat, S. Oraintara, T. Tanaka, and K. R. Rao, "Lossless multichannel eeg compression," in *2006 IEEE International Symposium on Circuits and Systems*. IEEE, 2006, pp. 4–pp.

[19] K. Srinivasan, J. Dauwels, and M. R. Reddy, "Multichannel EEG compression: Wavelet-based image and volumetric coding approach," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 1, pp. 113–120, Jan. 2013.

[20] J. Dauwels, K. Srinivasan, M. R. Reddy, and A. Cichocki, "Near-lossless multichannel eeg compression based on matrix and tensor decompositions," *IEEE journal of biomedical and health informatics*, vol. 17, no. 3, pp. 708–714, 2012.

[21] L. Shaw, D. Rahman, and A. Routray, "Highly efficient compression algorithms for multichannel eeg," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 26, no. 5, pp. 957–968, 2018.

[22] Z. Zhang, T.-P. Jung, S. Makeig, and B. D. Rao, "Compressed sensing of eeg for wireless telemonitoring with low energy consumption and inexpensive hardware," *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 1, pp. 221–224, 2012.

[23] M. Elsayed, A. Badawy, M. Mahmuddin, T. Elfouly, A. Mohamed, and K. Abualsaud, "FPGA implementation of DWT EEG data compression for wireless body sensor networks," in *Proc. IEEE Conf. Wireless Sensors (ICWiSE)*, Oct. 2016, pp. 21–25.

[24] C.-A. Chen, C. Wu, P. Abu, and S.-L. Chen, "Vlsi implementation of an efficient lossless eeg compression design for wireless body area network," *Applied Sciences*, vol. 8, no. 9, p. 1474, 2018.

[25] E. Chua and W. Fang, "Mixed bio-signal lossless data compressor for portable brain-heart monitoring systems," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 1, pp. 267–273, Feb. 2011.

[26] B. Carpentieri, M. J. Weinberger, and G. Seroussi, "Lossless compression of continuous-tone images," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1797–1809, Nov. 2000.

[27] A. C. Singer and M. Feder, "Universal linear prediction by model order weighting," *IEEE Transactions on Signal Processing*, vol. 47, no. 10, pp. 2685–2699, 1999.

[28] G.-O. Glentis and N. Kalouptsidis, "Efficient order recursive algorithms for multichannel least squares filtering," *IEEE Transactions on Signal Processing*, vol. 40, no. 6, pp. 1354–1374, 1992.

[29] D. Speck, "Fast robust adaptation of predictor weights from min/max neighboring pixels for minimum conditional entropy," in *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, vol. 1. IEEE, 1995, pp. 234–238.

[30] M. J. Weinberger, G. Seroussi, and G. Sapiro, "Loco-i: A low complexity, context-based, lossless image compression algorithm," in *Proceedings of Data Compression Conference-DCC'96.* IEEE, 1996, pp. 140–149.

[31] M. Causa, F. La Paz, S. Radi, J. P. Oliver, L. Steinfeld, and J. Oreggioni, "A 64-channel wireless eeg recording system for wearable applications," in *2018 IEEE 9th Latin American Symposium on Circuits & Systems (LASCAS).* IEEE, 2018, pp. 1–4.

[32] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[33] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, "Bci2000: a general-purpose brain-computer interface (bci) system," *IEEE Transactions on biomedical engineering*, vol. 51, no. 6, pp. 1034–1043, 2004.

[34] G. Dornhege, B. Blankertz, G. Curio, and K.-. Muller, "Boosting bit rates in noninvasive EEG single-trial classifications by feature combination and multiclass paradigms," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 6, pp. 993–1002, Jun. 2004.

[35] B. Blankertz, G. Dornhege, M. Krauledat, K.-R. Müller, and G. Curio, "The non-invasive berlin brain–computer interface: fast acquisition of effective performance in untrained subjects," *NeuroImage*, vol. 37, no. 2, pp. 539–550, 2007.

[36] A. Delorme, G. A. Rousselet, M. J.-M. Macé, and M. Fabre-Thorpe, "Interaction of top-down and bottom-up processing in the fast visual analysis of natural scenes," *Cognitive Brain Research*, vol. 19, no. 2, pp. 103–113, 2004.

[37] B. Hejrati, A. Fathi, and F. Abdali-Mohammadi, "Efficient lossless multi-channel eeg compression based on channel clustering," *Biomedical Signal Processing and Control*, vol. 31, pp. 295–300, 2017.

[38] S. Golomb, "Run-length encodings (corresp.)," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, Jul. 1966.

[39] I. Kuon, R. Tessier, J. Rose *et al.*, "Fpga architecture: Survey and challenges," *Foundations and Trends in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2008.

[40] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[41] A. Raghunathan, N. K. Jha, and S. Dey, *High-level power analysis and optimization*. Springer Science & Business Media, 2012.

[42] E. Boemo, J. P. Oliver, and G. Caffarena, "Tracking the pipelining-power rule along the fpga technical literature," in *Proceedings of the 10th FPGAworld Conference*, ser. FPGAworld '13. New York, NY, USA: ACM, 2013, pp. 9:1–9:5. [Online]. Available: http://doi.acm.org/10.1145/2513683.2513692

[43] S. Huda, M. Mallick, and J. H. Anderson, "Clock gating architectures for fpga power reduction," in *2009 International Conference on Field Programmable Logic and Applications*, Aug 2009, pp. 112–118.

[44] J. P. Oliver and E. Boemo, "Power estimations vs. power measurements in cyclone iii devices," in *2011 VII Southern Conference on Programmable Logic (SPL)*, April 2011, pp. 87–90.

[45] J. P. Oliver, J. P. Acle, and E. Boemo, "Power estimations vs. power measurements in spartan-6 devices," in *2014 IX Southern Conference on Programmable Logic (SPL)*, Nov 2014, pp. 1–5.

[46] *Intel Quartus Prime Standard Edition User Guide: Third-party Simulation*, Intel, 2018. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qps-tp-simulation.pdf

[47] T. Pitkänen, P. Jamieson, T. Becker, S. Moisio, and J. Takala, "Power consumption benchmarking for reconfigurable platforms," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 2, pp. 649–659, 2012.

[48] R. Herveille, "Wishbone system-on-chip (soc) interconnection architecture for portable ip cores, rev. version: B4," *By Open Cores Organization*, 2010.

[49] L. Sun, H. Deng, R. K. Sheshadri, W. Zheng, and D. Koutsonikolas, "Experimental evaluation of wifi active power/energy consumption models for smartphones," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 115–129, 2016.

[50] S. K. Saha, P. Deshpande, P. P. Inamdar, R. K. Sheshadri, and D. Koutsonikolas, "Power-throughput tradeoffs of 802.11 n/ac in smartphones," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 100–108.

[51] F. Favaro and J. P. Oliver, "Hardware implementation of a multi-channel eeg lossless compression algorithm," in *2019 X Southern Conference on Programmable Logic (SPL)*. IEEE, 2019, pp. 69–73.

Esta página ha sido intencionalmente dejada en blanco.

Glosario

ADC Analog to Digital Converter. 3, 11, 14, 51, 52

AFE Analog Front–End. 3, 11, 51, 52, 58

ALS Audio Lossless Coding. 4, 14

ASIC Application Specific Integrated Circuit. 3, 31, 32, 51, 60

BCI Brain–Computer Interface. 13

BLE Bluetooth Low Energy. 12

BR Basic Rate. 12

BSBL Block Sparse Bayesian Learning. 5

BT Bluetooth. 11–13

CMOS Complementary Metal Oxide Semiconductor. 6, 32

CR Compression Ratio. 14, 69

CS Compressed Sensing. 5

DCM Digital Clock Manager. 33, 41

DPCM Differential Pulse Code Modulation. 6

DSP Digital Signal Processor. 34

DWT Discrete Wavelet Transform. 5

ECG Electrocardiograma. 6

EDR Enhanced Data Rate. 12

EEG Electroencefalógrafo, Electroencefalograma. 1–7, 11–13, 27, 51, 54, 57, 59, 71

FIFO First–In First–Out. 29, 42, 53, 54

FPGA Field Programmable Gate Array. 3, 4, 6, 17, 18, 26, 27, 29, 31–37, 39–48, 51, 52, 54, 57–59, 69

HDL Hardware Description Language. 3, 51, 58

IDE Integrated Development Environment. 16

JTAG Joint Test Action Group. 54

LUT Lookup Table. 31, 33

LVDS Low Voltage Differential Signal. 31, 33–36

MCF Multi–Channel Fixed. 8, 13–15, 69

MCS Multi–Channel Speck. 8, 14, 69

PC Personal Computer. 13, 14, 18, 27, 29, 54

PCB Printed Circuit Board. 41, 52

PLL Phase Locked Loop. 17, 29, 33, 34, 41, 54

RLS Recursive Least Squares. 7, 8, 14

SPI Serial Periferial Interface. 11, 12, 17, 28, 29, 31, 34, 41, 52, 54, 56

SPP Serial Port Profile. 12

SRAM Static RAM (Random Access Memory). 31, 32

TTL Transistor Transistor Logic. 33

UART Universal Asynchronous Receiver-Transmitter. 12, 13, 56

USB Universal Serial Bus. 13, 54

VCD Value Change Dump. 40

VHDL VHSIC (Very High Speed Integrated Circuits) Hardware Description Language. 17, 18

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el miércoles 28 octubre, 2020.
http://iie.fing.edu.uy/