Universidad de la República
Facultad de Ingeniería

EPILOOP: Evaluación e implementación en FPGA de algoritmos para detección de epilepsia

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Santiago Colman, Sofía Duarte, Tamara Martínez

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutores

Santiago Martínez ..... Universidad de la República
Francisco Veirano ..... Universidad de la República

Tribunal

Pablo Cancela ..... Universidad de la República
Santiago Martínez ..... Universidad de la República
Juan Pablo Oliver ..... Universidad de la República
Francisco Veirano ..... Universidad de la República

Montevideo
Martes 15 Octubre, 2024

EPILOOP: Evaluación e implementación en FPGA de algoritmos para detección de epilepsia, Santiago Colman, Sofía Duarte, Tamara Martínez.

Contiene un total de 149 páginas.

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.

Richard Feynman

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Queremos expresar nuestros sinceros agradecimientos a todas aquellas personas que contribuyeron de manera significativa en la realización de este proyecto de grado.

En primer lugar, agradecer a nuestros tutores, Santiago Martínez y Francisco Veirano, por su guía experta, su constante apoyo y sus valiosas sugerencias que fueron fundamentales para el desarrollo y la finalización de este trabajo. Además, queremos expresar nuestro sincero agradecimiento a Juan Pablo Oliver por facilitarnos material imprescindible para una parte importante de nuestro proyecto, así como por su gran disposición y colaboración a lo largo del proceso.

Por otro lado, queremos expresar un profundo agradecimiento a nuestras familias y amigos por su inquebrantable apoyo, comprensión y motivación a lo largo de este viaje académico.

Sin el aporte generoso y el respaldo de todas estas personas, este proyecto no habría sido posible. Estamos sinceramente agradecidos por su contribución y dedicación.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El aumento en la prevalencia de enfermedades neurológicas y el avance en la comprensión de los mecanismos mediante los cuales se producen, ha impulsado la investigación en neuroestimuladores implantables. Estos dispositivos, diseñados para estimular el sistema nervioso, se dividen en dos clases: los estimuladores en lazo abierto y los estimuladores en lazo cerrado. Mientras que los primeros aplican estimulación de manera fija, los últimos interactúan con el sistema nervioso, adaptando la estimulación en respuesta a la actividad nerviosa del paciente.

El procesamiento de señales nerviosas, especialmente la extracción de características, consume una cantidad significativa de energía, lo que afecta la vida útil del dispositivo. En este proyecto, se propone implementar y evaluar en hardware (FPGA) algoritmos clásicos para la identificación de ataques epilépticos, con el objetivo de optimizar el consumo.

Este proyecto toma como antecedente el trabajo presentado por Raghunatan, et al en *A hardware-algorithm co-design approach to optimize seizure detection algorithms for implantable applications* [1]. En dicho trabajo se presenta la evaluación de distintos algoritmos de detección de epilepsia para realizar una optimización de diseño que tiene en cuenta tanto la eficacia en la detección como el costo referido al consumo y área del circuito. Las funciones de detección se evalúan por su capacidad para detectar convulsiones electrográficas a partir de datos de ratas tratadas con kainato, adquiridos por microelectrodos. A diferencia del antecedente, en el presente proyecto la implementación y evaluación se realizarán utilizando herramientas de diseño en FPGA y bases de datos existentes, donde se tienen señales registradas de humanos.

Se implementaron y compararon algoritmos clásicos, utilizando una metodología que incluyó variaciones en el tamaño de ventana, modificaciones en la cantidad de bits para la representación de datos y análisis de las características de las señales utilizadas. A partir de los resultados, se identificaron configuraciones óptimas que maximizan la capacidad de detección mientras minimizan el tiempo de reacción, lo que es de gran importancia para aplicaciones médicas en tiempo real. Este enfoque práctico y experimental permite establecer un punto de comparación sólido entre diferentes algoritmos y su adecuación para su implementación en neuroestimuladores implantables.

Para evaluar el rendimiento de los algoritmos, se llevaron a cabo comparaciones exhaustivas en tres aspectos clave: la capacidad de detección de cada algoritmo, su consumo y la cantidad de recursos lógicos utilizados. Los resultados permitieron identificar algoritmos con un equilibrio adecuado entre eficacia en la detección de crisis epilépticas y un consumo reducido, lo cual es crucial para extender la vida útil de un dispositivo implantable.

Tabla de contenidos

Tabla de contenidos

Esta página ha sido intencionalmente dejada en blanco.

Glosario

API: Application Programming Interface
AUROC: Area Under the ROC curve
DBS: Deep Brain Stimulation
ECoG: Electrocorticography
EEG: Electroencephalography
FDA: Food and Drugs Administration
FN: False Negatives
FP: False Positives
FPGA: Field-Programmable Gate Array
FPR: False Positives Rate
IED: Interictal Epileptiform Discharges
NLA: Non-Linear Autocorrelation
RMS: Root Mean Square
RNS: Responsive Neurostimulation
ROC: Receiver-Operating Characteristic curve
TCL: Transaction Control Language
TN: True Negatives
TP: True Positives
TPR: True Positives Rate
VCD: Value Change Dump
VHDL: VHSIC Hardware Description Language
VHSIC: Very-High-Speed Integrated Circuit
VNS: Vague Nerve Stimulation
VRS: Variance of Rectified Signal

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

1.1. Motivación

La epilepsia es un trastorno neurológico crónico caracterizado por la predisposición recurrente a crisis electrográficas, que son episodios súbitos y transitorios de actividad cerebral excesivamente intensa. Esta condición afecta a personas de todas las edades y puede tener un impacto significativo en la calidad de vida de quienes la padecen.

Las crisis, uno de los síntomas más reconocibles de la epilepsia, son el resultado de una actividad eléctrica anormal en el cerebro. Normalmente, las células nerviosas en el cerebro se comunican entre sí mediante impulsos eléctricos regulados. Sin embargo, en personas con epilepsia, este proceso puede interrumpirse, lo que provoca la generación de señales eléctricas caóticas y sincrónicas que pueden propagarse rápidamente a través de regiones del cerebro.

Las crisis pueden manifestarse de diversas formas, desde movimientos involuntarios repentinos y contracciones musculares hasta cambios en la conciencia y sensaciones extrañas. La naturaleza y la gravedad de las crisis pueden variar ampliamente según el tipo específico de epilepsia y la región del cerebro afectada.

Si bien la epilepsia puede tener múltiples causas, en muchos casos la causa subyacente no se identifica. Sin embargo, algunos factores que pueden contribuir al desarrollo de la epilepsia incluyen lesiones cerebrales traumáticas, accidentes cerebrovasculares, tumores cerebrales, trastornos genéticos y condiciones médicas como la esclerosis temporal mesial [2].

La epilepsia de inicio parcial o focal es el tipo de epilepsia más común. La opción de tratamiento primario para pacientes con dicha enfermedad es la farmacoterapia. Las drogas antiepilépticas son efectivas en controlar crisis epilépticas en aproximadamente un 60 % de las personas con epilepsia de inicio parcial. Sin embargo, queda una gran proporción de pacientes que continúan teniendo crisis

a pesar de la medicación. La neuroestimulación, que consiste en la estimulación del sistema nervioso a partir de impulsos eléctricos, es una opción para aquellos pacientes que tienen epilepsia refractaria (resistente a las drogas) y que no son candidatos a una cirugía resectiva o han fallado en la misma [3]. Tomando esto como principal motivación para el proyecto, se busca implementar y estudiar algunos de los algoritmos clásicos utilizados en dispositivos que pueden administrar estimulación cerebral.

1.2. Señales involucradas

El electroencefalograma (EEG) es una medida valiosa de la función eléctrica del cerebro. Es una representación gráfica de la diferencia de voltaje entre dos sitios de la función cerebral registrada a lo largo del tiempo. La electroencefalografía implica el estudio de la grabación de estas señales eléctricas generadas por el cerebro. El EEG extracraneal brinda una amplia visión de la actividad eléctrica en ambos hemisferios del cerebro, mientras que el EEG intracraneal permite obtener actividad detallada en ciertas regiones del cerebro mediante electrodos implantados quirúrgicamente. Con el EEG se puede revelar información sobre una disfunción cerebral difusa o focal, la presencia de descargas epileptiformes interictales (IED) o patrones de especial significado. La mayoría de las descargas que duran menos de 3 segundos no suelen demostrar signos perceptibles clínicamente, sin embargo, estas alteraciones del comportamiento pueden verse en un EEG [4].

La obtención de señales intracraneales se realiza mediante electrodos invasivos, los cuales se insertan en el cerebro de manera que el extremo quede localizado en el foco epiléptico o lo más próximo al mismo.

Las señales obtenidas tienen la forma de la Figura 1.1, en donde cada línea corresponde a la señal sensada por cada canal del electrodo. En este caso, se presenta una señal de EEG normal, donde el individuo está despierto y realizando tres movimientos oculares horizontales (mirar hacia la izquierda) y dos movimientos oculares verticales (parpadeo), los cuales se pueden apreciar principalmente en las señales de los canales F7 y F8.

Las técnicas no invasivas (como el EEG extracraneal) suelen tener resoluciones espaciales, temporales y relaciones señal a ruido bajas. Por el contrario, la técnica invasiva de electrocorticografía (ECoG), que implica la medición de señales eléctricas cerebrales mediante electrodos que se implantan subduralmente en la superficie del cerebro, proporciona señales que tienen una relación señal a ruido excepcionalmente alta, menos susceptibilidad a artefactos que el EEG y una alta resolución espacial y temporal. Esta técnica permite el estudio de los procesos corticales a gran escala [5]. En la Figura 1.2 se presenta un esquema con la posición de los electrodos según el tipo de técnica utilizada para registrar la actividad cerebral.

Figura 1.1: Ejemplo de señal de EEG, extraído de [4]. En la señal de la figura se tiene un EEG normal, sin actividad epileptiforme. El individuo está despierto y realizando movimientos oculares.

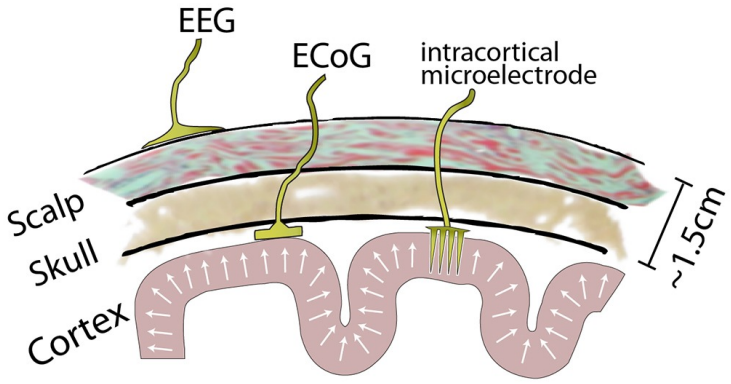


Figura 1.2: Profundidad de los electrodos utilizados para EEG extracraneal, ECoG desde la superficie del cerebro, y microelectrodos intracorticales, extraído de [6].

1.3. Estado del arte

La neuroestimulación es un tratamiento para la epilepsia refractaria relativamente nuevo y que se encuentra en rápido crecimiento [7]. Actualmente hay tres terapias de neuroestimulación aprobadas por la FDA para el tratamiento de la epilepsia de inicio parcial: estimulación del nervio vago (VNS) [8], estimulación cerebral profunda (DBS) [7] y estimulación cortical receptiva (RNS) [3].

El *RNS System* es el primer dispositivo comercial que provee estimulación ce-

rebral en lazo cerrado [3]. Este sistema incluye un neuroestimulador programable (dispositivo *Neuropace*) que es implantado cranealmente y monitorea continuamente el electrocorticograma a través de uno o dos cables profundos que son posicionados en el foco de la epilepsia. Cuando se detectan patrones de actividad electrográfica anormal, el neuroestimulador envía pulsos de estimulación eléctrica a los focos de epilepsia a través de los cables implantados, buscando detener los ataques epilépticos.

Los algoritmos de detección implementados en el neuroestimulador están diseñados para ser computacionalmente eficientes y realizan detección en tiempo real de patrones ECoG. Las herramientas de detección son configurables y pueden ser ajustadas por el médico para cada paciente. Estas herramientas detectan cambios en la señal electrográfica en cuanto a su amplitud, frecuencia y energía [3].

Al igual que la estimulación cortical receptiva (RNS), tanto la estimulación del nervio vago (VNS) así como la estimulación cerebral profunda (DBS) son tratamientos neuroestimuladores utilizados como terapia para pacientes con epilepsia resistente a los medicamentos que no pueden someterse a una cirugía. En el caso del VNS, un dispositivo es implantado debajo de la piel del pecho, y los electrodos de estimulación envían señales eléctricas hacia el nervio vago izquierdo. El dispositivo puede programarse en cuanto a la frecuencia, intensidad y duración de la estimulación, dependiendo de cada paciente [8]. La terapia DBS, por otro lado, puede suprimir las crisis a partir de la inhibición de regiones del cerebro hiperexcitables o alterar regiones específicas responsables de la propagación de la epilepsia [7].

El trabajo tomado como antecedente para este proyecto [1], considera algoritmos para la detección de epilepsia que utilizan herramientas similares a las implementadas en el *RNS System*. En dicho estudio, se presenta la evaluación de los distintos algoritmos de detección de epilepsia para realizar una optimización de diseño que tiene en cuenta tanto la eficacia de la detección como el costo referido al consumo y área del circuito. Las funciones de detección se evalúan por su capacidad para detectar crisis electrográficas a partir de datos de ratas tratadas con kainato, adquiridos por microelectrodos. Para evaluar el consumo energético y la cantidad de compuertas lógicas de cada algoritmo se utilizan herramientas de diseño de circuitos integrados en conjunto con una tecnología particular.

El EEG tiene un papel esencial en el seguimiento y tratamiento de la epilepsia. Esta técnica es extremadamente laboriosa para los médicos para anotar todas las señales registradas, particularmente en el seguimiento a largo plazo. El proceso de anotación a menudo implica identificar segmentos de la señal con características sospechosas de ataque epiléptico u otras anomalías, por lo tanto, la detección automatizada de la epilepsia se convierte en una cuestión clínica de especial importancia. En este sentido, se tienen *Transformadores para la Detección de Convulsiones*, que se tratan de arquitecturas de aprendizaje profundo que son capaces de detectar automáticamente las crisis con un buen rendimiento en el dominio de

entrada de frecuencia-tiempo [9]. Si bien estas arquitecturas son una herramienta de gran utilidad para la detección de crisis epilépticas, este tipo de soluciones no suelen usarse en neuroestimuladores implantables debido a su complejidad y a su alto consumo. En la detección de crisis, los modelos de aprendizaje profundo, especialmente las redes neuronales convolucionales, utilizan espectrogramas o características de Fourier, lo cual aumenta el costo de procesamiento y la ocupación de memoria, en particular a largo plazo [10].

1.4. Desarrollo del documento

En esta sección se presenta la organización del documento y los capítulos en los cuales se desarrolla.

- **Capítulo 1 Introducción:** Se introduce de forma concisa el tema que motiva el proyecto, así como las señales involucradas en el trabajo y el estado del arte.
- **Capítulo 2 Algoritmos:** Se presentan brevemente los algoritmos a implementar y comparar.
- **Capítulo 3 Implementación en Python:** Se detallan los resultados obtenidos a partir de la implementación en Python de los distintos algoritmos, realizando sobre ellos diversas pruebas de forma de elaborar conclusiones en cuanto a la eficacia en la detección de crisis epilépticas.
- **Capítulo 4 Implementación en hardware:** Se describe la implementación en VHDL de los diferentes algoritmos y los sistemas en los que se aplican, así como la simulación de la llegada de datos.
- **Capítulo 5 Medida de consumo:** Se explica la metodología utilizada para medir el consumo de los diferentes algoritmos implementados en hardware, presentando los resultados obtenidos.
- **Capítulo 6 Comparación de algoritmos:** Se realiza la comparación de los resultados para los diferentes algoritmos en términos de capacidad de detección, consumo y cantidad de elementos lógicos utilizados.
- **Capítulo 7 Conclusiones:** Se presentan las conclusiones del proyecto, discutiendo según los resultados obtenidos. Se plantean además directrices sobre posibilidades para continuar con el trabajo a futuro.

Al final del documento se encuentran los apéndices, en los cuales se presentaron: la API utilizada para la descarga de señales de la base de datos, la lista de señales utilizadas, gráficas obtenidas para los distintos algoritmos en las diversas pruebas realizadas en Python, el código TCL para la escritura de la ROM, los diagramas de bloques completos de los sistemas y las tablas con las distintas medidas de consumo registradas.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Algoritmos

Se presentan a continuación los algoritmos a implementar y evaluar en el desarrollo del Proyecto. Estos seis algoritmos son los mismos que fueron estudiados en el trabajo considerado como antecedente principal [1].

Los algoritmos hacen uso de distintas características de la señal de EEG, de forma de poder detectar si se está frente al comportamiento normal de la señal o frente a una crisis epiléptica.

Para todos los algoritmos, $x[i]$ representa la i-ésima muestra de la señal x. Para los algoritmos *Energy*, *Hjorth variance parameter*, *RMS Amplitude* y *Variance of rectified signal*, "N" representa la cantidad de muestras consideradas (tamaño de la ventana). Por otro lado, para el algoritmo *Non-linear Autocorrelation*, "N" representa la cantidad de grupos de muestras considerados.

2.1. Energy

Esta característica refleja la potencia promedio de la señal usando la energía acumulada. Se utiliza una ventana rectangular deslizante para calcular la energía según se describe en la Ecuación 2.1 [11].

$$E = \frac{1}{N} \sum_{i=1}^{N} x[i]^2 \tag{2.1}$$

Este promedio presentado en la Ecuación 2.1 se realiza para cada ventana de manera de recorrer toda la señal.

Las mediciones de energía en la señal de EEG conforman uno de los métodos cuantitativos utilizados para analizar la actividad relacionada con las crisis epilépticas. Esta medición se caracteriza no solo por la simplicidad y rapidez del cálculo, sino también por la sensibilidad a las formas de onda del EEG asociadas con las crisis, los focos epilépticos, las espigas epileptiformes, las ondas rítmicas de

contorno agudo y la actividad de alta amplitud y baja frecuencia (0.5–4 Hz) que se observa después de ráfagas de actividad epileptiforme [11].

Estudios realizados a un grupo de pacientes indica que las crisis epilépticas pueden comenzar como una cascada de eventos electrofisiológicos que evolucionan a lo largo de horas y que las medidas cuantitativas de la actividad eléctrica previa a la crisis podrían utilizarse para predecir las crisis electrográficas mucho antes de su inicio clínico. Específicamente, se descubrió que 50 minutos previos al inicio de la crisis, la energía acumulada de la señal aumenta en comparación con la línea base [11].

2.2. Coastline

La característica representa la suma del valor absoluto de las diferencias de amplitud entre muestras de EEG consecutivas [1].

$$CL = \sum_{i=1}^{N} |x[i] - x[i-1]| \tag{2.2}$$

Este algoritmo utiliza la variabilidad en la longitud de línea de la señal para identificar áreas donde se producen cambios significativos, los cuales pueden suponer eventos epileptiformes, como espigas o descargas. Esto se debe a que dicha cantidad aumenta en presencia de oscilaciones neurales de amplitud alta o frecuencia alta, lo cual puede marcar convulsiones [12].

Se caracteriza por su simplicidad y rapidez en el cálculo, y la sensibilidad a la variabilidad de la señal. Sin embargo, puede presentar limitaciones si la señal presenta ruido de gran amplitud [13].

2.3. Hjorth variance parameter

Los parámetros de Hjorth se han utilizado ampliamente en cálculos estadísticos basados en EEG. Estos parámetros son tres, “actividad” (ACT), “movilidad” (MOB) y “complejidad” (COM), y miden la varianza, la frecuencia promedio y los cambios de frecuencia de una señal, respectivamente [12].

El primer parámetro, que es igual a la varianza de la amplitud de la señal [1] y se puede expresar según la Ecuación 2.3, es el que se implementa en el presente proyecto.

$$H = \frac{1}{N} \sum_{i=1}^{N} x[i]^2 - \mu^2 \tag{2.3}$$

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x[i] \tag{2.4}$$

Este parámetro identifica las variaciones en la amplitud de la señal a través de su dispersión.

El parámetro de Hjorth puede ser difícil de computar eficientemente, debido a que utiliza operaciones de elevar al cuadrado [12].

2.4. RMS Amplitude

La amplitud cuadrática media representa la raíz cuadrada de la potencia de la señal [1].

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x[i]^2} \tag{2.5}$$

Por definición, la amplitud RMS es similar a la estimación de energía, y se utiliza para estudiar el efecto de combinar características de detección matemáticamente similares en la eficacia general del algoritmo.

2.5. Non-linear autocorrelation

La mecánica de este método consiste en considerar un conjunto de puntos, definir grupos con una cantidad específica de dichos puntos y calcularles los máximos y mínimos. Estos se denominan $max(X[i])$ y $min(X[i])$, donde $X[i]$ representa un grupo [14].

$$\begin{aligned} HV &= min\{max\{X[i]\}, max\{X[i+1], X[i+2]\}\} \\ LV &= max\{min\{X[i]\}, min\{X[i+1], X[i+2]\}\} \end{aligned} \tag{2.6}$$

$$NLA = \sum_{i=1}^{N} HV[i] - LV[i] \tag{2.7}$$

Este algoritmo se basa en la detección de patrones repetitivos en el EEG durante una crisis electrográfica, específicamente en la correlación entre los máximos y mínimos de las señales de EEG a lo largo de intervalos de tiempo cortos. En lugar de analizar la forma completa de las ondas cerebrales, el algoritmo simplifica el procesamiento al reducir los datos a dos puntos clave: el máximo y el mínimo dentro de cada intervalo. Esta simplificación permite detectar rápidamente crisis electrográficas, basándose en la consistencia de estos patrones repetitivos, sin necesidad de procesar toda la información morfológica de las ondas. Aunque este enfoque acelera el procesamiento, sigue siendo eficaz para identificar crisis, incluso en entornos con ruido, ya que las ondas de ruido tienden a mostrar menor correlación entre intervalos consecutivos [13].

2.6. Variance of rectified signal

Se utiliza la potencia del EEG como una característica de la crisis y se obtiene a partir de la suma de las desviaciones absolutas de la media de la señal [14].

$$VRS = \sum_{i=1}^{N} (x[i] - \mu)^2 \tag{2.8}$$

$$\mu = \frac{1}{N} \sum_{i=1}^{N} |x[i]| \tag{2.9}$$

En las Ecuaciones 2.8 y 2.9 μ representa la media de los valores absolutos de los puntos de datos en el período utilizado para la evaluación. Este algoritmo funciona de forma similar al Hjorth, buscando obtener información de la señal a partir de la varianza.

Sin embargo, puede fallar si la magnitud del ruido inherente a la señal es mayor en magnitud a las espigas que se generan durante una crisis electrográfica. Además, no parece obtener información del aumento de la autocorrelación temporal de la señal durante una crisis o de la presencia de espigas.

Capítulo 3

Implementación en Python

3.1. Señales

Las señales utilizadas para el estudio de los diferentes algoritmos son obtenidas de una base de datos que contiene registros de la actividad cerebral de diversos pacientes humanos que presentan episodios de epilepsia. La misma pertenece al portal IEEG.ORG, una iniciativa llevada a cabo por el NINDS (*National Institutes of Neurological Disorders and Stroke*). [15]

Las señales están catalogadas por estudios, donde se indica el individuo del cual se obtuvieron las mismas, el centro de salud donde fue realizado el registro, entre otra información. Además, las señales se encuentran etiquetadas por profesionales médicos, indicando el comienzo y el término de las diferentes crisis epilépticas.

En este trabajo se utilizaron señales que presentan focos epilépticos parciales, es decir cuando la actividad cerebral anormal afecta una pequeña área del cerebro, y simples, cuando la crisis no afecta la conciencia. Este filtro acota la cantidad de estudios que pueden utilizarse. Adicionalmente, este trabajo se limitó a estudios que presentan documentación en la que se incluyen los reportes clínicos, con sus respectivas fechas. Estos detallan los canales en los cuales se detectó inicialmente la crisis, la zona en la cual se generó la misma, interpretaciones clínicas, entre otras.

Además, el portal tiene una funcionalidad que permite visualizar el EEG, indicando las zonas donde ocurren crisis. Sumado a la documentación, esto facilita la elección de las señales a descargar, proceso que implica: elección de estudio, identificación de canales donde se presenta una crisis y determinación de intervalos de tiempo de la señal. Simplificar este proceso es vital dado que las señales almacenadas en la base de datos pueden tener una duración de varios días.

Los estudios de donde finalmente se obtuvieron las señales fueron: *Study 005*, *Study 006*, *Study 019* y *Study 030*. Los mismos fueron realizados por la institución *Mayo Clinic*.

Se utilizó una API especialmente diseñada para interactuar con el portal IEEG.org, adjunta en el Apéndice A. Además de permitir descargar las señales con intervalos de tiempo deseados, esta herramienta acondiciona las señales de forma de tengan frecuencia de muestreo de $250Hz$. Esto permite simplificar el procesamiento de las mismas.

Se listan todas las señales utilizadas en el presente trabajo en el Apéndice B.

3.2. Implementación de algoritmos

En primera instancia se implementaron en Python los algoritmos descritos en el Capítulo 2.

Para utilizar los algoritmos, se debió seleccionar inicialmente el tamaño de ventana y cuantas muestras se deben superponer entre las mismas, es decir, los algoritmos permiten un funcionamiento de ventana móvil. Este mecanismo no fue incorporado en el algoritmo NLA, debido a que su implementación sería diferente a los demás algoritmos y por lo tanto, comparar resultados al desplazar la ventana móvil no sería posible. En el caso de este algoritmo, se seleccionó la cantidad de grupos que constituyen una ventana y la cantidad de muestras que constituyen un grupo.

A partir de la implementación en Python, una de las salidas que devuelve cada algoritmo es el resultado de las operaciones realizadas en cada ventana. Se muestra un ejemplo en la Figura 3.2, donde se presenta el resultado de las operaciones del algoritmo Energy al alimentarlo con la señal de la Figura 3.1.

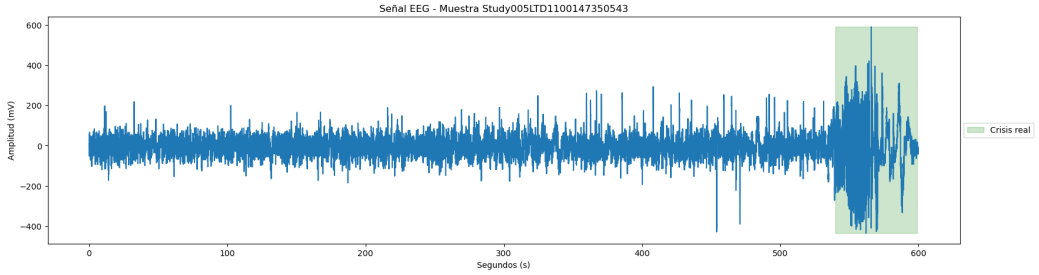


Figura 3.1: Señal EEG - Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

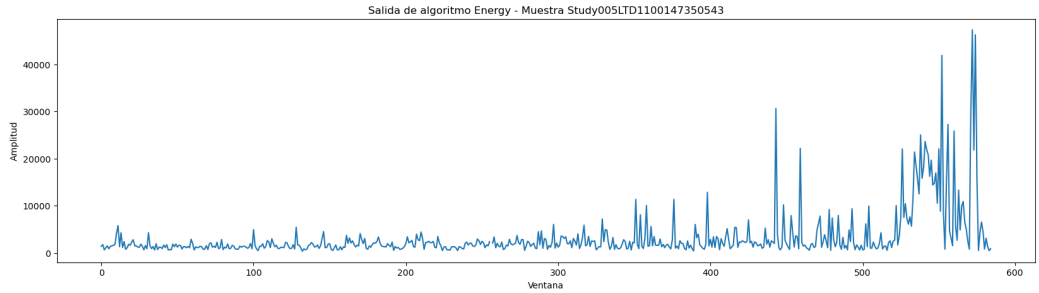


Figura 3.2: Salida de algoritmo Energy con ventana de 256 muestras, sin superposición de ventanas - Señal inyectada: Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

Esta implementación es una herramienta con gran potencial, debido a que permite observar en la salida del algoritmo el impacto de los cambios de la señal biológica, causados por crisis electrográficas. Al mismo tiempo, facilita la visualización del efecto que tiene modificar los parámetros del algoritmo en la resolución de la señal de salida. Por ejemplo, aumentar el tamaño de la ventana de 256 a 1024 en el algoritmo Energy genera una disminución en la resolución de la señal de salida, lo cual se vuelve evidente en la Figura 3.3.

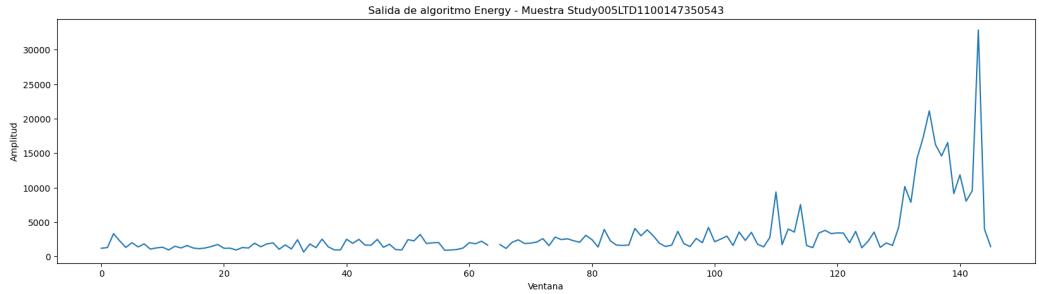


Figura 3.3: Salida de algoritmo Energy con ventana de 1024 muestras, sin superposición de ventanas - Señal inyectada: Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

Por otro lado, se generó otra salida de los algoritmos, la cual resultó de comparar el resultado de las operaciones con un valor de umbral determinado. El umbral se eligió según los valores que toma el resultado de la operación en los intervalos de tiempo en los que se sabe que se está frente a una crisis epiléptica, gracias al etiquetado de las señales utilizadas. Luego, si un resultado es mayor al umbral elegido, se considera que el algoritmo detectó una crisis y se guarda el índice de la primera muestra de la ventana junto a un booleano, que indica la presencia de dicha crisis.

De esta forma se tiene que la implementación permite observar como puede cambiar la capacidad de detección de crisis epilépticas de un algoritmo al cambiar el valor del umbral.

Luego, fue necesario diseñar una metodología para evaluar la eficacia de detección de los algoritmos utilizando la implementación descrita. Un posible procedimiento comenzó por utilizar las señales del Apéndice B, por ejemplo la presentada en la Figura 3.1, e inyectarla en los algoritmos. Se observó la salida que representa el resultado de las operaciones (Figura 3.2) y contrastando con el etiquetado de crisis, se determinó un umbral. En este caso, se fijó el umbral en 20000. Luego, utilizando otra señal del mismo estudio, se la inyectó en los algoritmos y se comparó con el umbral previamente seleccionado. Lo que se espera es que, cuando se le aplique el algoritmo a la señal, si se está frente a una crisis, el resultado de la operación que arroja el algoritmo sea mayor a este valor de umbral. Cabe destacar que las diferencias entre señales biológicas al cambiar de individuo son considerables, por lo que, para evaluar la capacidad de detección con un cierto umbral elegido, es de vital importancia utilizar una señal del mismo estudio, preferentemente del mismo canal. El resultado del proceso de detección se observa en la Figura 3.4.

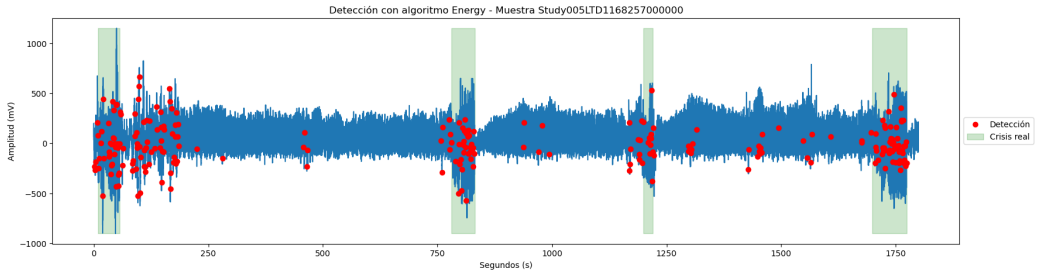


Figura 3.4: Detección con algoritmo Energy - Estudio 005 - Canal LTD1 - Tiempo de inicio 168257000000 μs - Umbral: 20000. Puntos rojos: Detección de crisis por algoritmo.

Se visualiza que el algoritmo Energy identifica repetidamente las crisis electrográficas presentes en la señal de forma correcta, no obstante, aparecen múltiples falsos positivos. La selección del umbral juega un papel crucial en este fenómeno.

La utilización de otro algoritmo también puede mejorar la detección, como se observa en la Figura 3.5 , donde se utilizó el algoritmo Coastline con un umbral de 4000. Este caso parece tener un mejor nivel de detección, debido a la ausencia de falsos positivos.

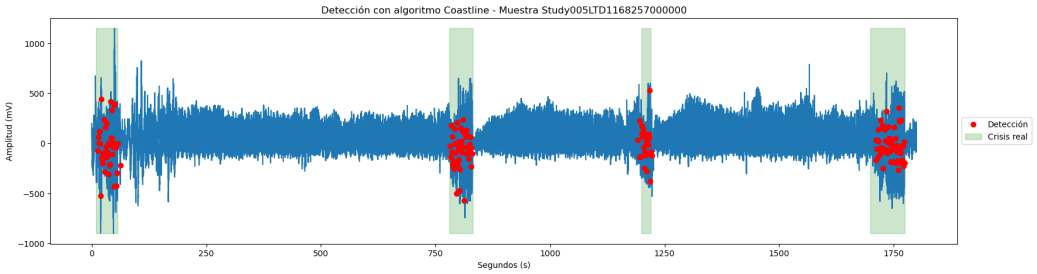


Figura 3.5: Detección con algoritmo Coastline - Estudio 005 - Canal LTD1 - Tiempo de inicio 168257000000 μs - Umbral: 4000. Puntos rojos: Detección de crisis por algoritmo.

Sin embargo, este no es un método eficaz para determinar la capacidad de detección de los algoritmos, debido a que está condicionado por las elecciones manuales que se realicen de los umbrales, lo que lo hace inadecuado para análisis que impliquen múltiples repeticiones y ajustes frecuentes de estos umbrales.

Adicionalmente, la evaluación del algoritmo bajo este proceso es cualitativa y poco precisa, por lo que se necesita un método que arroje resultados cuantitativos, más apropiados para análisis comparativos entre los algoritmos.

3.3. Curvas ROC y AUROC

La curva ROC es una técnica para seleccionar clasificadores en base a su rendimiento. Este método es ampliamente utilizado en el campo médico para el análisis del comportamiento de sistemas diagnósticos. [16]

El proceso de clasificación, en este caso los algoritmos de detección, arroja un resultado binario para cada ventana a un umbral determinado: positivo si detecta una crisis y negativo en caso contrario. Dada esta clasificación y el etiquetado de las bases de datos, hay cuatro resultados posibles:

- Falso positivo (FP)
- Falso negativo (FN)
- Verdadero positivo (TP)
- Verdadero negativo (TN)

Con estos valores se determinan dos métricas con las cuales se construye la curva ROC. Estas son la tasa de verdaderos positivos (TPR), también conocida como tasa de aciertos; y la tasa de verdaderos negativos (FPR), conocida como tasa de falsas alarmas, las cuales se calculan como se detalla a continuación:

$$TPR = \frac{TP}{TP+FN} = \frac{\text{Positivos correctamente clasificados}}{\text{Positivos reales totales}}$$
$$FPR = \frac{FP}{FP+TN} = \frac{\text{Negativos incorrectamente clasificados}}{\text{Negativos reales totales}} \tag{3.1}$$

Iterando el proceso de clasificación para diferentes umbrales, se construye una curva en un plano bidimensional, donde el eje de las abscisas está conformado por los valores de FPR y el eje de ordenadas por los de TPR. Esta es la curva ROC. Se demuestra un ejemplo en la Figura 3.6, donde se procesó la señal de la Figura 3.1 utilizando el algoritmo Energy, iterando 10 veces con distintos umbrales.

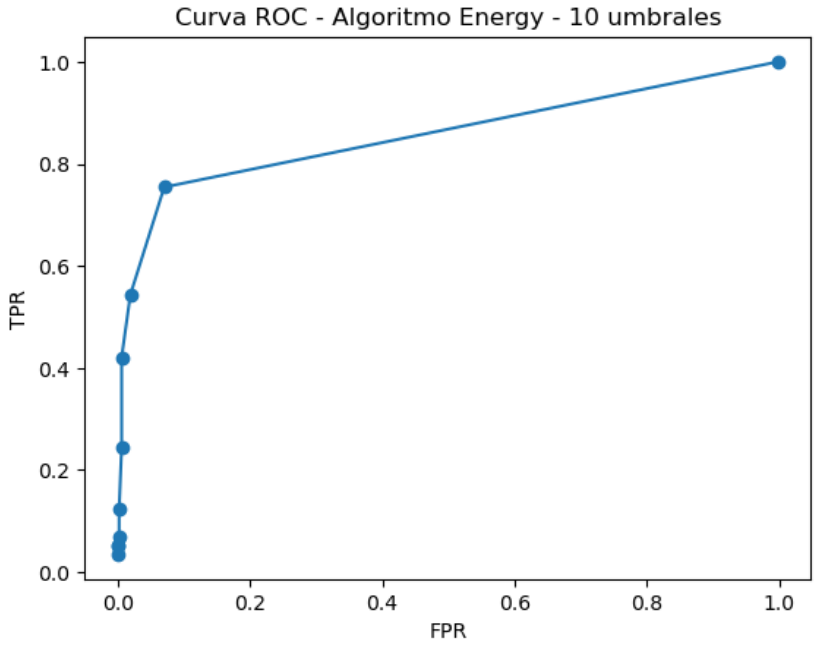


Figura 3.6: Curva ROC - Algoritmo Energy - Ventana de 256 muestras - Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

Para obtener un valor que determine la capacidad de detección del algoritmo utilizado en la señal inyectada, se calculó el área debajo de la curva, la cual se denomina *Area under the ROC curve* (AUROC). Este método permite una evaluación cuantitativa de los algoritmos y es particularmente idóneo si se busca realizar un estudio comparativo, como es el caso del presente trabajo.

La AUROC puede tomar valores comprendidos entre 0 y 1. Cuanto mayor es su valor, mayor es la capacidad de detección del algoritmo. Además, un valor de 0.5 implica que el algoritmo funciona de la misma forma que el azar. Un clasificador que utilice correctamente la información en los datos presentará una curva ROC cuyos puntos se encuentren en la zona triangular superior izquierda del plano. [17] [16]

Considerando este método como base de nuestro estudio comparativo, se formuló un procedimiento para calcular la AUROC para cada señal en la que se pretende aplicar los algoritmos y comparar su capacidad de detección. Luego de procesar la señal a través de un algoritmo, se obtuvo la señal de salida. A esta se le calculó el máximo y se dividió entre la cantidad de puntos con la que se desea construir la curva ROC. Estos valores serán los umbrales contra los que se comparará la señal de salida del algoritmo al realizar la iteración en la cual se determinan los puntos de la curva, siendo estos los pares de valores *TPR* y *FPR*. Una vez construida la curva, se calculó el área debajo de la misma para obtener la AUROC, como fue mencionado previamente.

3.4. Comparación inicial de algoritmos

Habiéndose determinado el método de evaluación de la capacidad de detección de los algoritmos, se procedió a realizar un análisis comparativo inicial, con una cantidad de muestras limitada. Las señales que se utilizaron para la comparación inicial son las detalladas en el Apéndice B, particularmente las listadas en la Tabla B.1. En esta instancia, se utilizaron los algoritmos variando el tamaño de la ventana sin usar la funcionalidad de ventana móvil. Adicionalmente, se buscó que la cantidad de muestras de cada ventana corresponda a una potencia de 2. Esta selección prevé mejorar la eficiencia de la implementación de los algoritmos en Hardware, como se explica posteriormente en la Subsección 4.2.2. Para cada tamaño de ventana considerado, se construyeron las curvas ROC con umbrales diferentes.

Los resultados correspondientes a los algoritmos Energy, Coastline, Hjorth, RMS Amplitude y VRS se presentan en la Figura 3.7. En la misma, se observa el promedio de los valores de AUROC obtenidos de cada señal para cada implementación del algoritmo correspondiente.

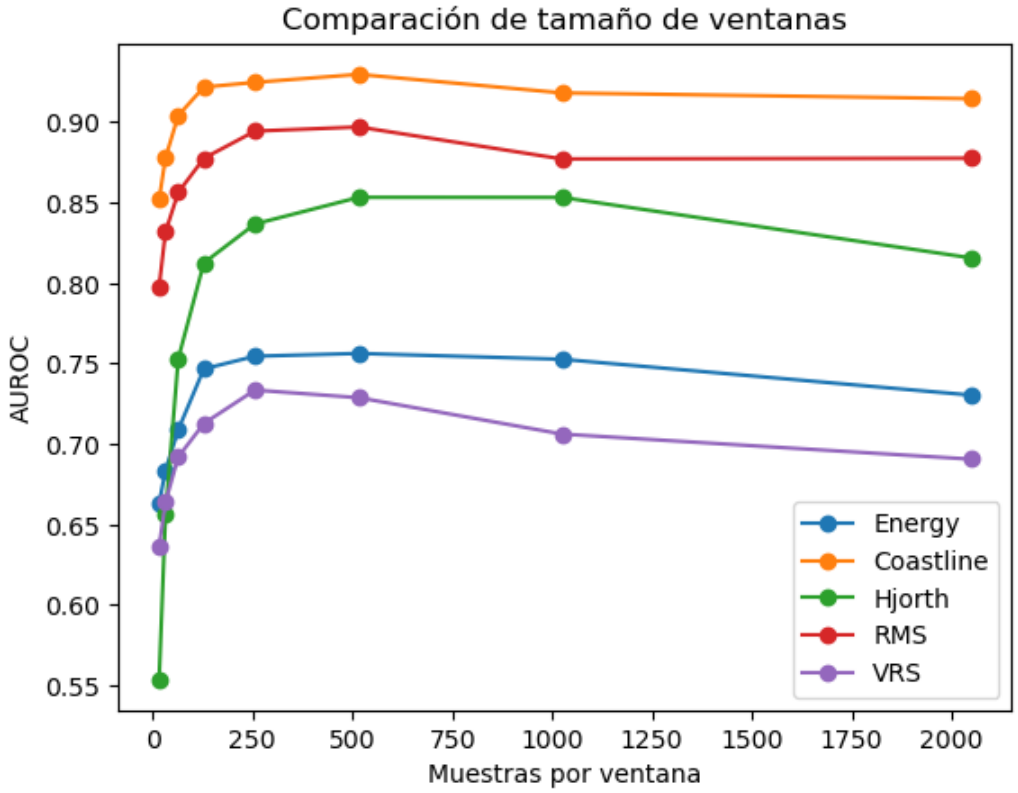


Figura 3.7: Comparación de capacidad de detección de algoritmos para diferentes tamaños de ventana.

Dado que el algoritmo NLA utiliza una ventana de muestras diferente a la de los otros algoritmos, se llevó a cabo un estudio específico. En este análisis, se varía la cantidad de muestras que componen cada grupo y la cantidad de grupos que conforman una ventana. Los resultados obtenidos de cada señal se promediaron y se presentan en un mapa de calor en la Figura 3.8.

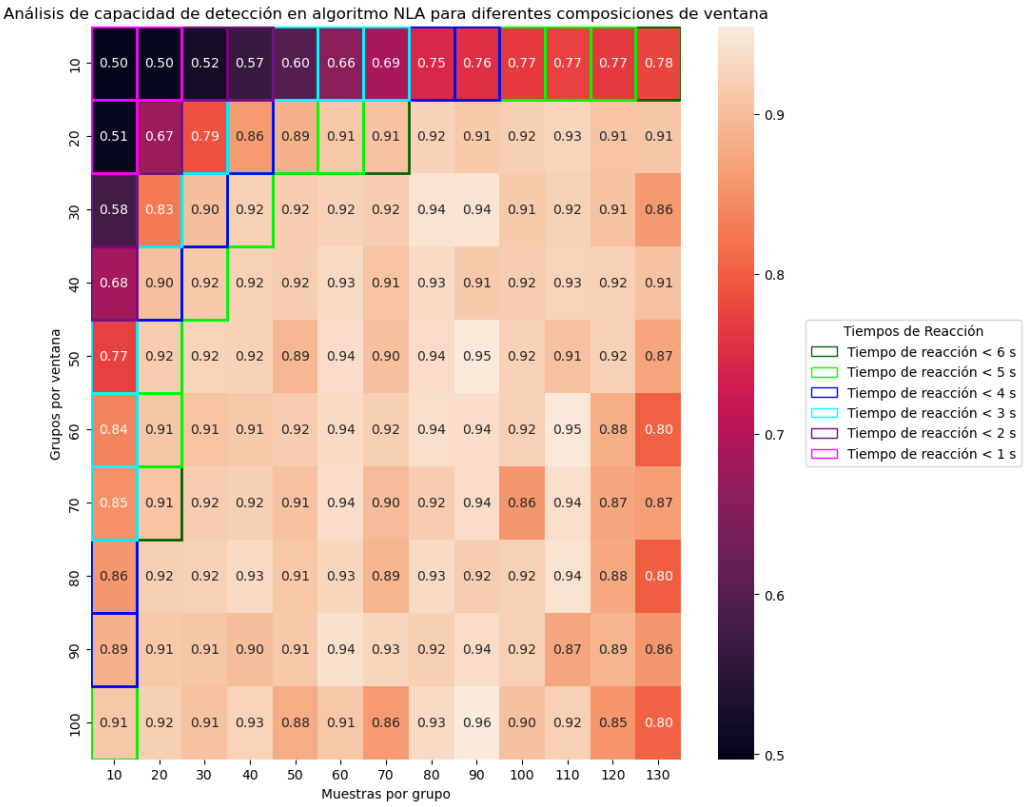


Figura 3.8: Mapa de calor de promedios de AUROC. La capacidad de detección se contrasta con tiempos de reacción del algoritmo de acuerdo a la cantidad de muestras en una ventana.

A partir de estos análisis, se pueden extraer conclusiones preliminares aunque no definitivas. La Figura 3.7 sugiere una tendencia de los algoritmos a maximizar su eficacia en detección cuando el tamaño de la ventana es de 256 o 512 muestras. Esto resulta conveniente para su aplicación en la detección de crisis epilépticas, ya que el tiempo de detección correspondiente sería de 1 y 2 segundos (considerando la cantidad de muestras por ventana y la frecuencia de muestreo especificada en 3.1), respectivamente. Además, se destaca una ventaja de la capacidad de detección del algoritmo Coastline en comparación con los otros algoritmos analizados.

Los resultados presentados en la Figura 3.8 indican que, para el algoritmo NLA, el uso de una ventana más grande mejora la capacidad de detección, ya que permite capturar mejor las características relevantes de las señales. Sin embargo, este incremento en el tamaño de la ventana también conlleva un aumento en el tiempo de procesamiento del algoritmo, lo que puede ser un inconveniente en aplicaciones que requieren respuestas rápidas. Además, se observa que el aumento de la AUROC no es continuo; a medida que el tamaño de la ventana se vuelve demasiado grande, la capacidad de detección del algoritmo comienza a disminuir, sugiriendo la existencia de un punto óptimo de tamaño de ventana.

A partir de estos resultados, se definieron tamaños de ventana específicos para los análisis posteriores, permitiendo establecer un punto de comparación adecuado para todos los algoritmos. En los análisis donde el tamaño de la ventana es fijo, los algoritmos Energy, Coastline, Hjorth, RMS Amplitude y VRS se implementarán utilizando una ventana de 256 muestras. Esto permite una detección adecuada y, al mismo tiempo, minimiza el tiempo de respuesta del sistema. De manera análoga, el algoritmo NLA se implementará con una ventana de 40 grupos, cada uno compuesto por 10 muestras, con el fin de mantener un tiempo de procesamiento similar al de los otros algoritmos, mientras se maximiza la eficiencia en la detección.

3.5. Cálculo de AUROC

El análisis inicial realizado en la Sección 3.4 utilizó 10 umbrales para construir las curvas ROC de cada algoritmo, lo cual podría poner en duda si el valor de AUROC obtenido con tan pocos puntos refleja de manera precisa la capacidad de detección de cada algoritmo. Sin embargo, al repetir el análisis con una mayor cantidad de puntos, como se muestra en la Figura 3.9, se observó un cambio en la forma de la curva ROC, aunque la diferencia en el valor de la AUROC no es significativa (85,4 % para 10 umbrales frente a 86,9 % para 110 umbrales). Esto sugiere que, aunque aumentar el número de puntos en la curva ROC puede proporcionar una representación más detallada, la cantidad inicial de 10 umbrales es suficiente para obtener una estimación aproximada de la capacidad de detección del algoritmo.

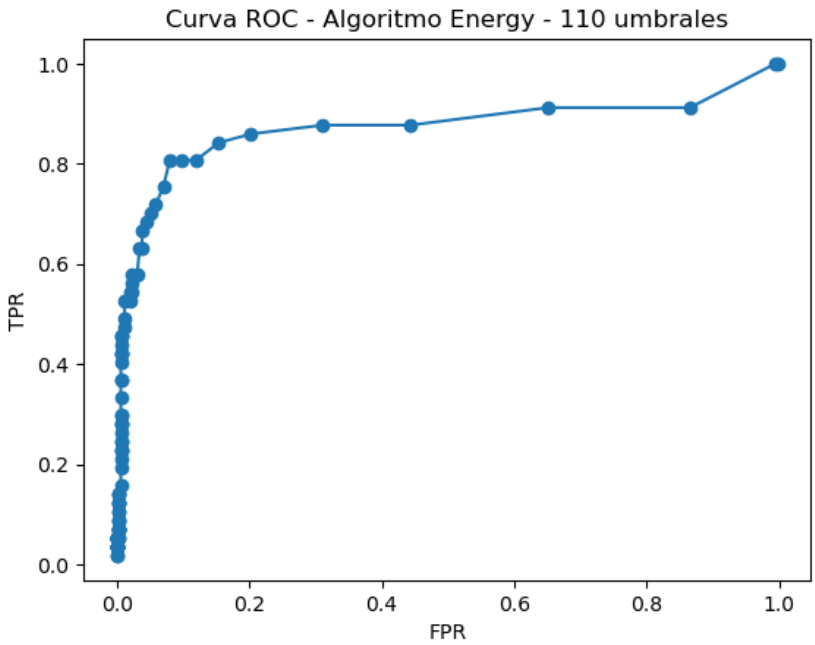


Figura 3.9: Curva ROC - Algoritmo Energy - Ventana de 256 muestras - Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

Al realizar la misma prueba con una señal diferente, los resultados muestraron una situación distinta. Las curvas ROC obtenidas utilizando 10 y 110 umbrales, mostradas en la Figura 3.10 y la Figura 3.11, respectivamente, presentaron variaciones más significativas en la forma de la curva. Esto sugiere que, en ciertos casos, el uso de un mayor número de umbrales puede impactar más notablemente en la precisión de la AUROC, reflejando mejor la capacidad de detección del algoritmo para esa señal específica.

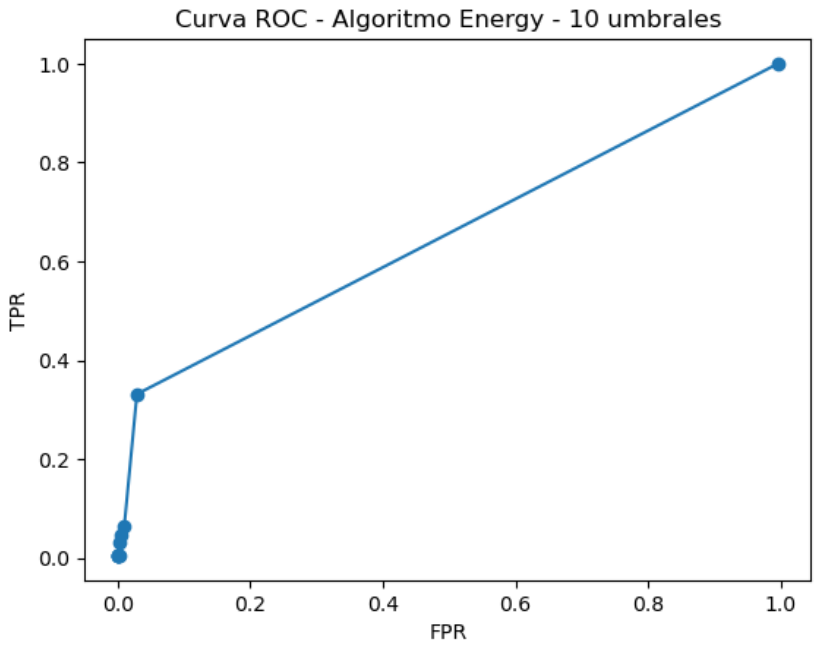


Figura 3.10: Curva ROC - Algoritmo Energy - Ventana de 256 muestras - Estudio 005 - Canal LTD1 - Tiempo de inicio 168257000000 μs.

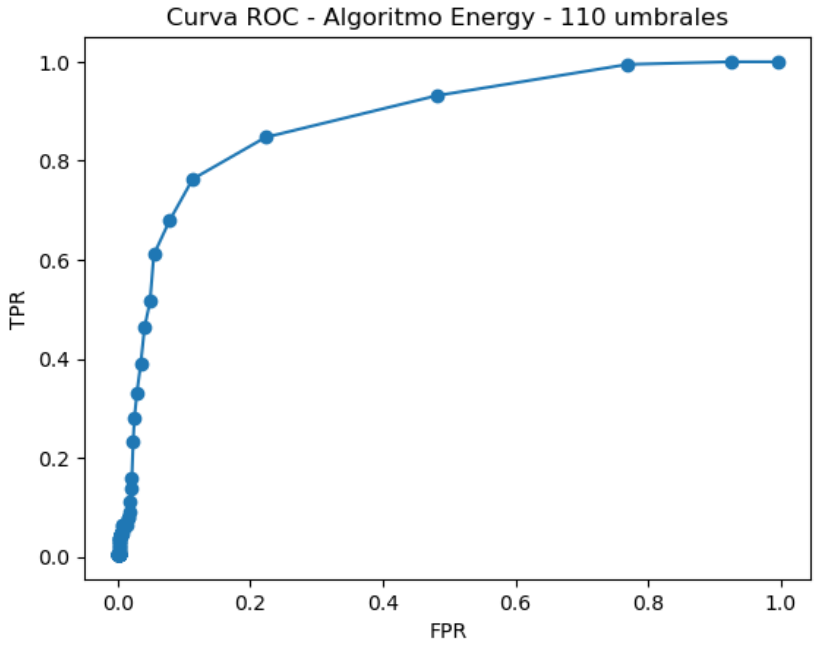


Figura 3.11: Curva ROC - Algoritmo Energy - Ventana de 256 muestras - Estudio 005 - Canal LTD1 - Tiempo de inicio 168257000000 μs.

En esta instancia, el cambio en la curva ROC es más pronunciado, lo cual

se refleja en la diferencia entre los valores de AUROC: 64,8 % para 10 umbrales y 87,8 % para 110 umbrales. La primera evaluación, con menos umbrales, sugirió que el algoritmo no es efectivo en la detección de la crisis para la señal inyectada. Sin embargo, al aumentar la cantidad de umbrales utilizados, la eficacia del algoritmo mejoró notablemente, demostrando una mayor capacidad de detección. Esto pone de manifiesto la importancia de seleccionar adecuadamente la cantidad de umbrales al evaluar la efectividad de un algoritmo en la detección de eventos.

Con esto en mente, se llevó a cabo un análisis para determinar cuantos puntos son suficientes para construir curvas ROC cuyas AUROC sean fehacientes. Los algoritmos procesan las señales de las listas B.1 y B.2 con una ventana de muestras fija, aumentando la cantidad de puntos que se utilizan para elaborar las curvas ROC y calcular la AUROC. Con este análisis no se buscó que los algoritmos alcanzaran su mejor aptitud de detección, sino que determinaran cual era la cantidad de puntos en la que el valor de AUROC se estabiliza.

De acuerdo a las ventanas determinadas como óptimas en la Sección 3.4, se utiliza para este análisis una ventana de 256 muestras para los algoritmos Energy, Coastline, Hjorth, RMS Amplitude y VRS; mientras que para el algoritmo NLA se determina una ventana conformada por 40 grupos, de 10 muestras cada uno.

Un ejemplo del resultado del análisis se muestra en la Figura 3.12, en este caso para el algoritmo Energy. Las gráficas correspondientes a todos los estudios se anexan en el Apéndice C.

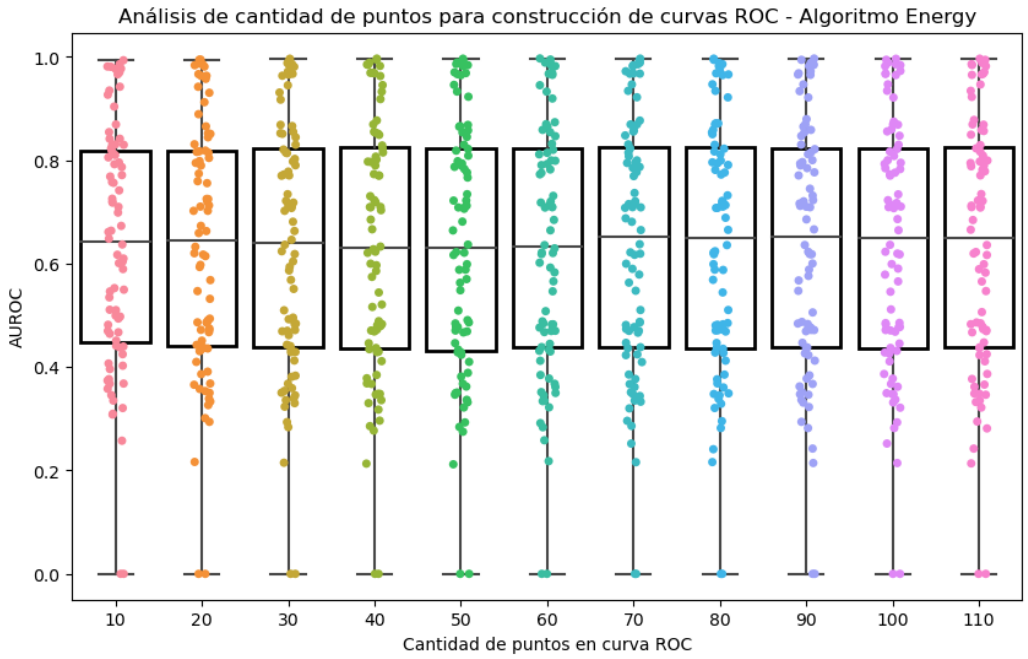
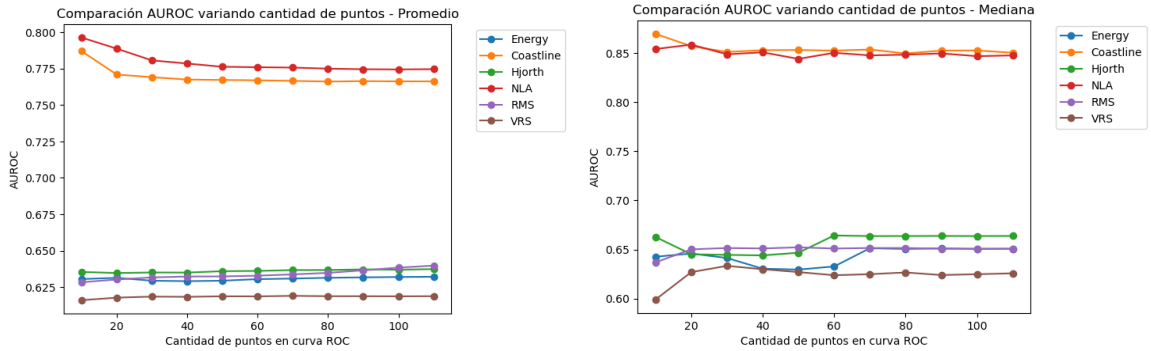


Figura 3.12: Análisis de cantidad de puntos para construcción de curvas ROC - Algoritmo Energy - Ventana de 256 muestras.

Al analizar los resultados, se observó una ligera variación en los valores de AUROC en función de la cantidad de puntos. Sin embargo, esta variación no es lo suficientemente clara, por lo que, para facilitar la visualización de los resultados, se calculan y presentan el promedio y la mediana de los datos para cada cantidad de puntos en las Figuras 3.13a y 3.13b, respectivamente. La mediana, al ser más robusta frente a la presencia de *outliers*, se utilizó junto al promedio para proporcionar una interpretación más precisa y completa de la información.



(a) Promedios (b) Medianas

Figura 3.13: Análisis de cantidad de puntos para construcción de curvas ROC.

Inmediatamente la utilización de la mediana probó ser útil, dado que reveló

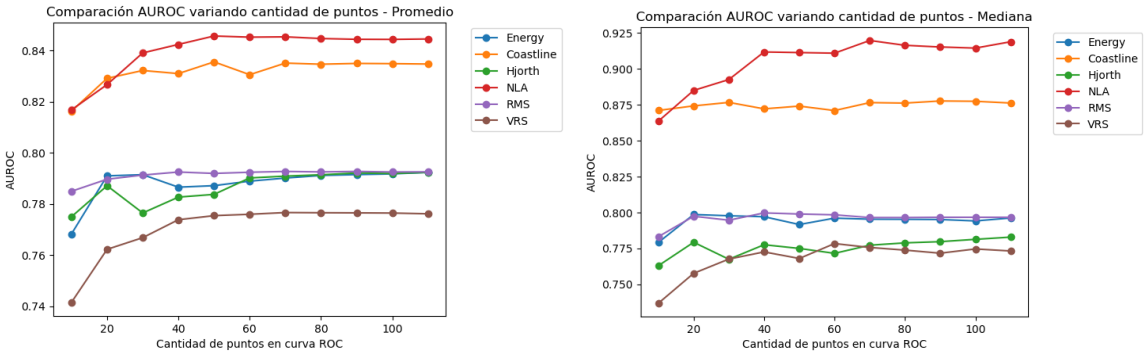
un comportamiento diferente al promedio. Guiándose por este último, el resultado de los algoritmos Energy, Hjorth, RMS Amplitude y VRS es independiente de la cantidad de puntos que se utilizan para la construcción de las curvas ROC. La mediana por el contrario, muestra que el valor de AUROC no es estable hasta los 70 puntos aproximadamente.

En ambos casos se observó que el valor de AUROC para los algoritmos Coastline y NLA comienza a estabilizarse en el entorno de los 30 puntos. Si bien se observó una disminución de la AUROC con una mayor cantidad de puntos, lo que se debe destacar es que se mantenga constante, de forma que sea una fiel representación de la eficacia del algoritmo.

Los cambios que se observaron en las AUROC fueron pequeños, por lo que podría considerarse que estas son aproximadamente constantes en función de la cantidad de umbrales para todos los algoritmos, y no parecerían exigir una gran cantidad de puntos para la construcción de las ROC. Esto podría ser una ventaja, en particular si utilizar una mayor cantidad de puntos para la construcción de las curvas ROC significa aumentar considerablemente el tiempo de procesamiento y disminuir la cantidad de señales que se pueden analizar, ya que el procesamiento de cada señal se volvería más costoso computacionalmente.

De forma de poder verificar el resultado anterior, se buscó filtrar los datos analizados para estudiar si este comportamiento se repitió para los casos en que la detección resultó en una AUROC mayor a 50 %. Con esto, se evaluó la eficacia de los algoritmos únicamente cuando lograron un nivel de detección mejor al azar.

Nuevamente, se expresaron los resultados en términos de su promedio y mediana, en las Figuras 3.14a y 3.14b, esta vez sin considerar valores de AUROC menores a 50 %.



(a) Promedios. (b) Medianas.

Figura 3.14: Análisis de cantidad de puntos para construcción de curvas ROC - Restricción de datos con AUROC > 0.5.

En este caso el comportamiento de los datos fue enteramente diferente. Por un lado, todos los algoritmos muestraron una mejora de AUROC a medida que aumentaron los puntos, mientras que en el análisis anterior, presentaron una disminución o no hubo cambio considerable.

Más significativamente, tanto en el caso del promedio como la mediana, los datos parecen converger a un valor luego de cierta cantidad de puntos. Este no fue el caso al usar la totalidad de los datos para los algoritmos Energy, Hjorth, RMS Amplitude y VRS. Esto verificó que los valores de AUROC no son independientes de la cantidad de puntos con los que se construye la curva ROC en dichos algoritmos.

A partir de estos análisis, se determinó utilizar 80 puntos al construir las curvas ROC para todos los estudios posteriores, de forma de representar la capacidad de detección de los algoritmos de forma fehaciente, particularmente en los casos en que AUROC $> 50\,\%$.

Se aplicó este requerimiento en el estudio realizado en la Sección 3.4 para observar su efecto en el cálculo de la AUROC. Los resultados se presentan en la Figura 3.15.

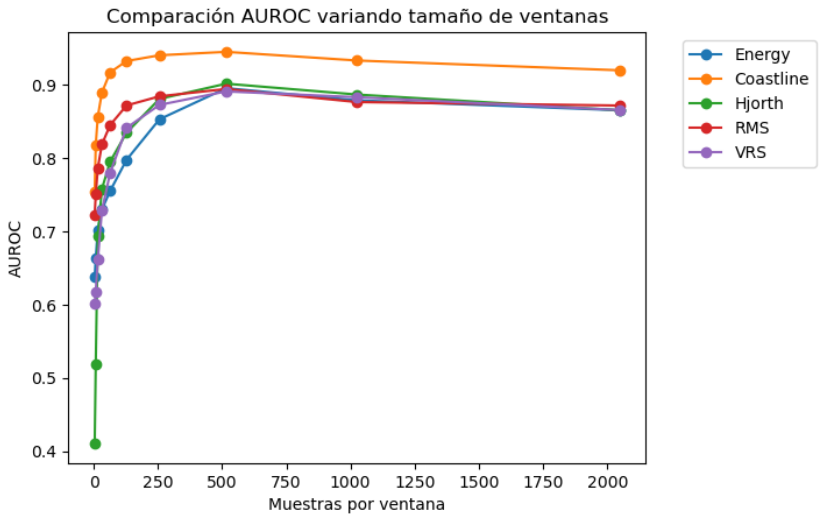


Figura 3.15: Comparación de ventanas para algoritmos - Utilización de 80 puntos para construcción de curvas ROC.

El impacto de construir correctamente las curvas ROC se hizo evidente. Se registraron aumentos importantes en los valores de AUROC para gran parte de los algoritmos, lo cual no significó que su eficacia haya mejorado, sino que ahora se representaron correctamente.

3.6. Comparación de algoritmos originales

Una vez determinado el procedimiento de construcción de curvas ROC, se procedió a comparar la capacidad de detección de los algoritmos bajo diferentes condiciones.

La primera instancia de comparación se realizó considerando la implementación original de los algoritmos. Los tamaños de ventana utilizados son los determinados al final de la Sección 3.4. Adicionalmente, las señales utilizadas se encuentran en las listas B.1 y B.2 y no fueron modificadas de ninguna forma previo a su procesamiento.

Los resultados se muestran en la Figura 3.16.

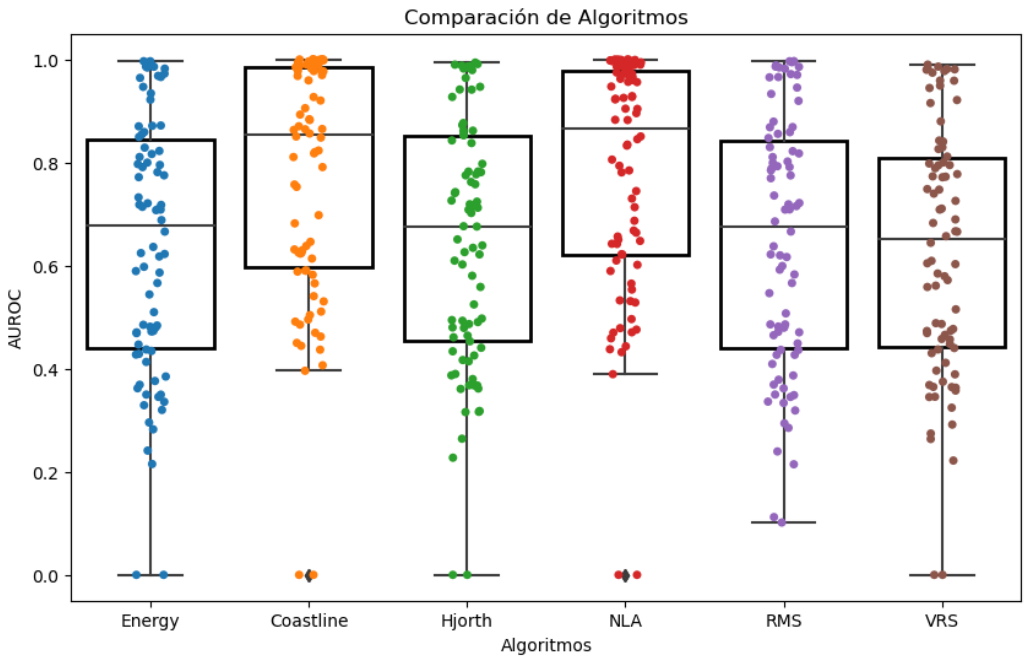


Figura 3.16: Comparación de algoritmos originales sin modificación a las señales biológicas.

Lo primero que salta a la vista es la superioridad en capacidad de detección de los algoritmos Coastline y NLA frente a los demás. Mientras tanto, los algoritmos Energy, Hjorth, RMS Amplitude y VRS presentaron similares valores de AUROC entre ellos.

Estos resultados parecen indicar que ciertos mecanismos de procesamiento son más adecuados para trabajar con señales que presentan crisis electrográficas. Tanto Coastline como NLA funcionan detectando diferencias entre valores dentro de una misma ventana, lo que los hace sensibles a cambios de alta frecuencia en la señal. Mientras tanto, el resto de los algoritmos responden a cambios en la amplitud de

la señal.

Otro aspecto a resaltar es que todos los algoritmos presentaron datos de AUROC por debajo de 50 %. Es decir, que ninguno es infalible. Sin embargo, los cuatro algoritmos con peor capacidad de detección tuvieron más puntos por debajo de 50 %, implicando que estos fallaron para una mayor cantidad de señales.

De las últimas dos observaciones se infiere que la eficacia de los algoritmos depende fuertemente de la señal estudiada.

3.7. Truncado de algoritmos y señales

Debido a que los algoritmos deben ser implementados en Hardware, es de vital importancia preguntarse en cada paso del análisis si se pueden realizar cambios a los algoritmos o a los datos de entrada, de forma de facilitar el diseño o mejorar la eficiencia energética del sistema.

Con esto en mente, trabajar con números enteros permite facilitar el diseño del sistema, al mismo tiempo de disminuir la cantidad de bits con la que se representan los datos, disminuyendo el área de FPGA a utilizarse. El primer estudio que se realizó fue determinar si las muestras de las señales pueden convertirse de números decimales a números enteros sin causar una disminución en la eficacia de los algoritmos. Al mismo tiempo, los algoritmos se modificaron para que las operaciones se realicen únicamente con números enteros, simplificándolas.

Para este estudio se utilizaron las señales de las listas B.1 y B.2. Los algoritmos se configuraron con los tamaños de ventana definidos en la Sección 3.4. La cantidad de bits con la que se representaron los datos es suficiente para que no haya necesidad de modificar el valor de las muestras, por fuera del redondeo.

El resultado de la comparación se observa en la Figura 3.17. En esta, se puede ver que no hay diferencias significativas entre los valores de AUROC obtenidos para los datos de entrada originales y los truncados. Por lo tanto, podemos concluir que es viable trabajar con datos truncados, lo que simplifica la posterior implementación en hardware.

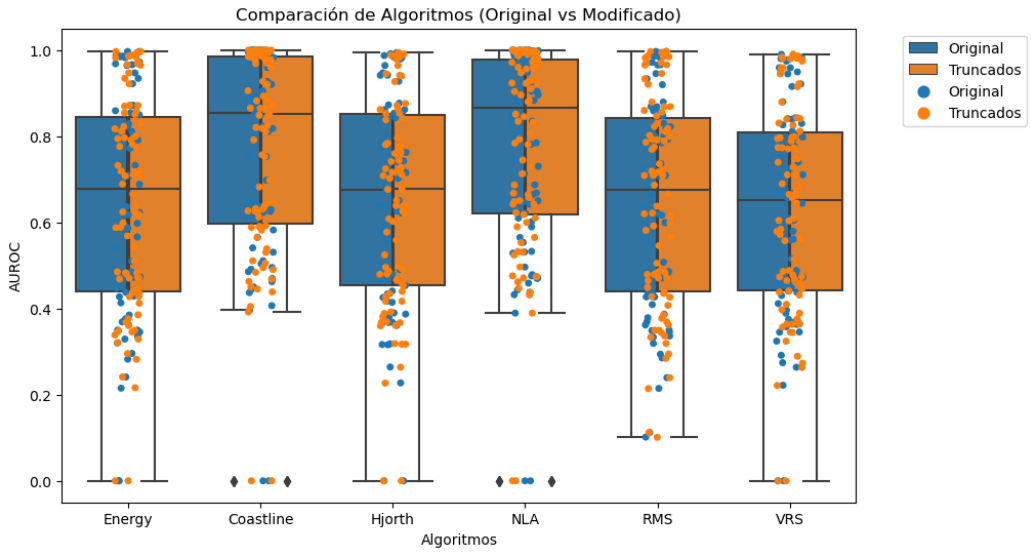


Figura 3.17: Comparación de algoritmos originales contra algoritmos truncados.

3.8. Respuesta de algoritmos a diferentes señales

En análisis anteriores se utilizaron señales de las listas B.1 y B.2, que se obtuvieron de los estudios 005, 019 y 030 de la base de datos. Estas señales tienen en común que no presentan *offset*, es decir que están centradas en 0.

Por otro lado, las señales del estudio 006, listadas en la Tabla B.3, no comparten esta característica. En la Figura 3.18 se muestra un ejemplo de estas señales, donde se visualiza un *offset* negativo.

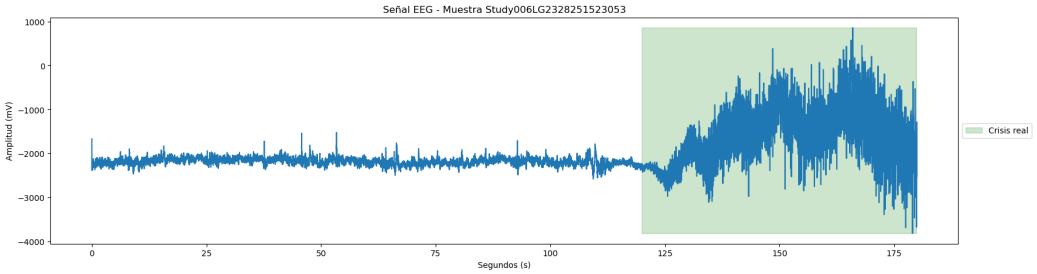


Figura 3.18: Señal EEG - Estudio 006 - Canal LG23 - Tiempo de inicio 28251523053 μs.

El comportamiento de las señales no necesariamente permanece constante, por lo que evaluar los algoritmos en condiciones como las del estudio 006 permite distinguir cuáles de ellos son robustos ante estos cambios. Consecuentemente, se realizó un estudio de la efectividad de detección de los algoritmos comparando su respuesta ante señales de los estudios 005, 019 y 030 en contraste con señales del estudio 006. Los resultados se presentan en la Figura 3.19.

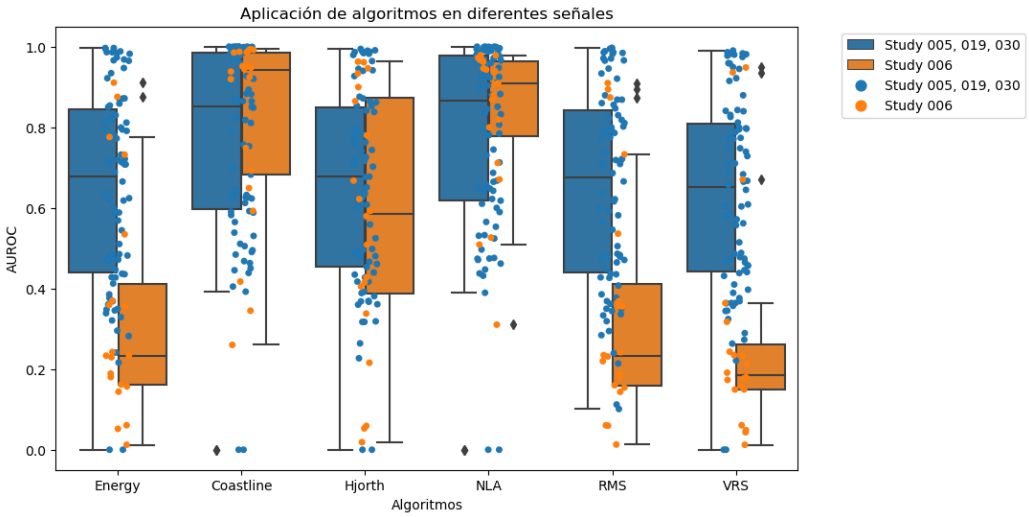


Figura 3.19: Capacidad de detección de algoritmos ante señales de estudios 005, 019 y 030 en comparación con estudio 006.

Se puede observar que para el estudio 006, algoritmos como el Energy, Hjorth, RMS y VRS presentan un valor de AUROC menor a los de los otros estudios. La presencia de un *offset* negativo en las señales analizadas puede influir significativamente en la eficacia de algoritmos que utilizan cálculos basados en el cuadrado de los valores de la señal.

Cuando se aplica un *offset* negativo, el algoritmo produce una salida con mayor magnitud cuando la señal se encuentra en reposo comparado con el momento en el que la señal presenta una crisis electrográfica, como se observa en la Figura 3.20.

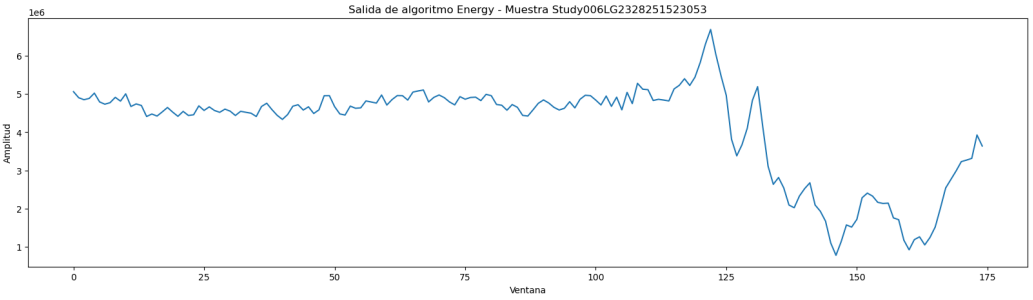


Figura 3.20: Salida del algoritmo Energy con una ventana de 256 muestras - Señal inyectada: Estudio 006 - Canal LG23 - Tiempo de inicio 28251523053 μs.

Al aparecer una crisis, la amplitud de la señal se acerca al 0, invirtiendo el comportamiento que el algoritmo presenta en condiciones normales (sin *offset*). Esta inversión puede dificultar la identificación de patrones relevantes, resultando

en un menor área bajo la curva ROC (AUROC) en comparación con señales centradas en 0 o con un *offset* positivo. Como consecuencia, ciertos algoritmos pueden mostrar menor capacidad de detección, subrayando la importancia de considerar el *offset* de las señales al evaluar su eficacia.

En contraste con esos algoritmos, el algoritmo Coastline, que se basa en la diferencia entre valores consecutivos y acumula el valor absoluto de estas diferencias, mostró una eficacia superior al calcular la AUROC con señales que presentaban un *offset* negativo. Esto se debe a que, al centrarse en las variaciones locales de la señal en lugar de su magnitud absoluta, Coastline es menos sensible a la inversión de la señal provocada por un *offset* negativo, lo que se puede visualizar en la Figura 3.21. Del mismo modo, el algoritmo NLA, que calcula máximos y mínimos y realiza comparaciones entre estos valores, también demostró un desempeño favorable en estas condiciones. Ambos algoritmos pueden identificar patrones significativos en las señales, incluso cuando se ven afectadas por un *offset* negativo, lo que sugiere que su diseño los hace más robustos ante cambios en la posición de la señal. Esto resalta la importancia de considerar la naturaleza específica de cada algoritmo al evaluar su eficacia en diferentes condiciones de señal.

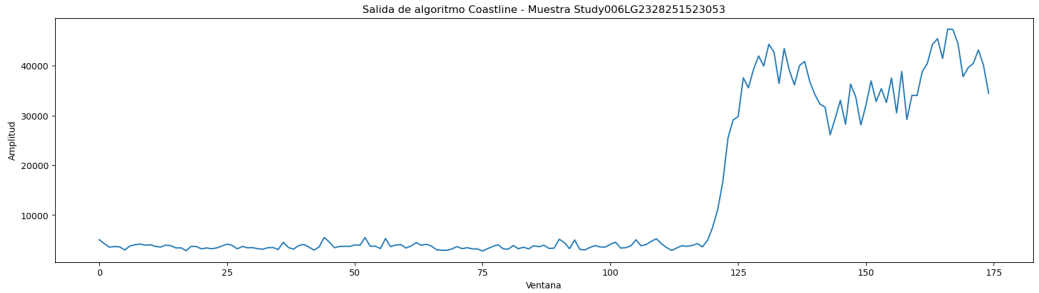


Figura 3.21: Salida del algoritmo Coastline con una ventana de 256 muestras - Señal inyectada: Estudio 006 - Canal LG23 - Tiempo de inicio 28251523053 μs.

Una posible solución consiste en un mecanismo para eliminar el *offset* de la señal antes de que ingrese al algoritmo. Si esto se llevara a cabo de forma digital, implicaría un aumento en el consumo, una consecuencia innecesaria en el caso de Coastline y NLA, donde la capacidad de detección no se ve afectada.

3.9. Operaciones lógicas entre algoritmos

Los algoritmos no necesariamente deben trabajar individualmente para detectar una crisis. Si el sistema admite que múltiples algoritmos funcionen en simultáneo es posible obtener un nuevo resultado de una ventana a través de operaciones lógicas. Las operaciones que se consideraron en este caso son AND y OR.

Previo a presentar los resultados, es ineludible explicar el procedimiento por el cual se obtuvieron, para una mejor interpretación de los mismos. En la Sección 3.5

se finaliza explicando como se construyen las curvas ROC. Se obtuvo una señal de salida para cada algoritmo, a partir de la cual se generó una lista de valores de umbral, que corresponden a cada punto de la curva ROC. En cada paso de la iteración, se procesaron las ventanas de la señal con los algoritmos y el resultado se comparó contra el umbral que determine el índice. Para cada algoritmo, se generó entonces un vector que contiene los resultados de la detección para cada ventana de la señal. A partir del mismo se calcularon los valores de *TPR* y *FPR*. En este estudio, previo a este cálculo, se realizaron las operaciones lógicas AND y OR entre cada valor de los vectores de detección. Esto permitió crear curvas ROC que correspondan a los resultados de las operaciones lógicas.

Este procedimiento tiene la desventaja que los vectores de detección con los que se opera no necesariamente son los resultantes de evaluar la detección con un umbral que optimice todos lo algoritmos simultáneamente. Por lo tanto, es posible que la capacidad de detección resultante de las operaciones lógicas no sea óptima.

Los resultados de este análisis se presentan en la Figura 3.22 y Figura 3.23, para las operaciones AND y OR respectivamente. Además se contrastan con las medias obtenidas en 3.17, para el caso de los algoritmos truncados.

Asimismo, en este análisis no se incluye el algoritmo NLA, ya que se busca comparar aquellos algoritmos que operan con el mismo tamaño de ventana. Esta elección garantiza que los vectores que contienen los resultados de detección correspondan a la misma ventana.

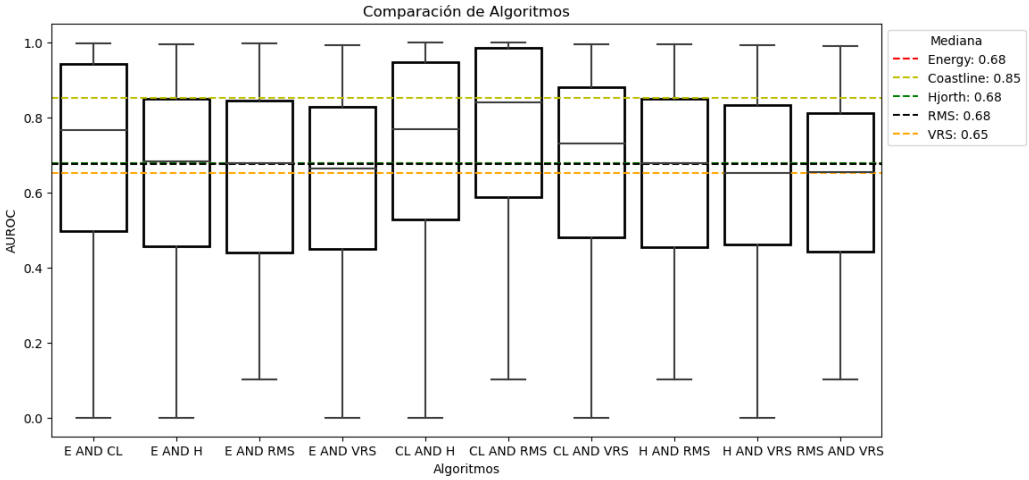


Figura 3.22: Resultados de operación lógica AND entre algoritmos comparado con medianas de resultados de algoritmos individuales.

El análisis de las combinaciones AND que se muestra en la Figura 3.22 revela que, al evaluar los valores de mediana de AUROC, el algoritmo Coastline sobresale al situarse en 85 %, mientras que Energy, Hjorth, RMS y VRS presentan valores

más bajos, todos en torno a 68 %. La AUROC resultante de las combinaciones de Energy con otros algoritmos tienden a estar cercanas a la eficacia de Energy, indicando una limitada mejora. Por ejemplo, la combinación Energy AND Coastline se sitúa en el punto medio entre las medianas de ambos, sugiriendo que Coastline aporta un aumento significativo, pero no suficiente para superar su propia mediana. En el caso de la combinación entre Coastline y RMS, el valor obtenido es casi idéntico al Coastline, lo que refuerza la robustez del RMS. En el caso de la combinación del Coastline y Hjorth, el valor de AUROC aumenta respecto de la mediana del Hjorth pero no lo suficiente para igualar o superar el Coastline. Finalmente, las combinaciones entre Hjorth y VRS también se alinean en torno a sus respectivos valores, sugiriendo que su capacidad de detección es similar cuando se consideran conjuntamente.

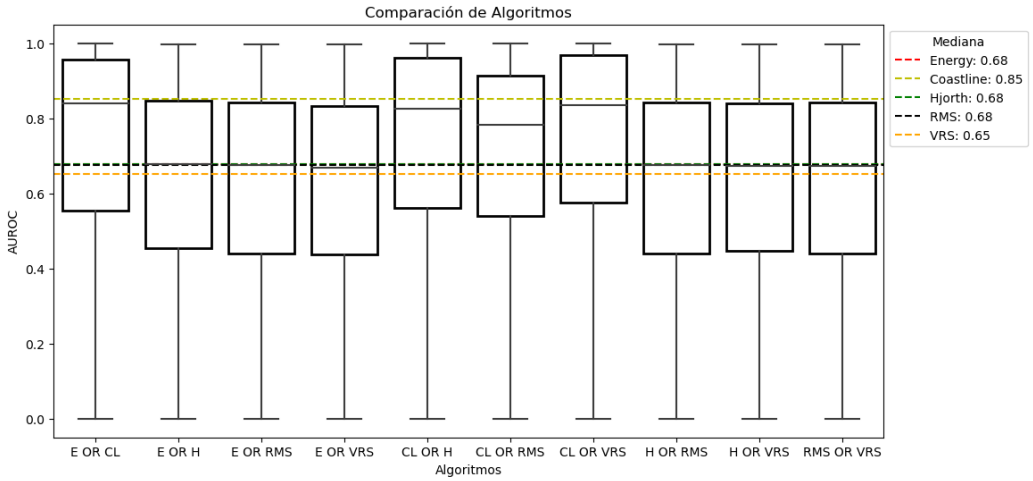


Figura 3.23: Resultados de operación lógica OR entre algoritmos comparado con medianas de resultados de algoritmos individuales.

Los resultados que se observan en la Figura 3.23 reflejan la efectividad del algoritmo Coastline, que, al combinarse con otros algoritmos, logra mantener una alta eficacia en términos de AUROC. Por ejemplo, la combinación Energy OR Coastline arroja un valor cercano a 85 %, indicando que la presencia de Coastline mejora significativamente el desempeño en comparación con Energy, que se sitúa en 68 %. De manera similar, las combinaciones de Coastline con Hjorth y RMS presentan valores de AUROC también cercanos a 78 % y 84 %, respectivamente, lo que sugiere que Coastline potencia la eficacia de los algoritmos a los que se asocia. Sin embargo, las combinaciones que involucran Energy tienden a mantenerse cerca del valor de Energy, reflejando su limitada contribución en comparación con Coastline. En resumen, las combinaciones OR parecen beneficiarse de la inclusión de Coastline, lo que resalta su capacidad para mejorar la detección en escenarios donde se requieren operaciones lógicas de este tipo.

En todos los casos, es claro que el algoritmo Coastline individualmente es superior en capacidad de detección a cualquier combinación que se realice. Buscamos

realizar un análisis mas profundo, analizando como se comporta la detección, utilizando la combinación con el algoritmo Energy como ejemplo. El resultado se presenta en la Figura 3.24.

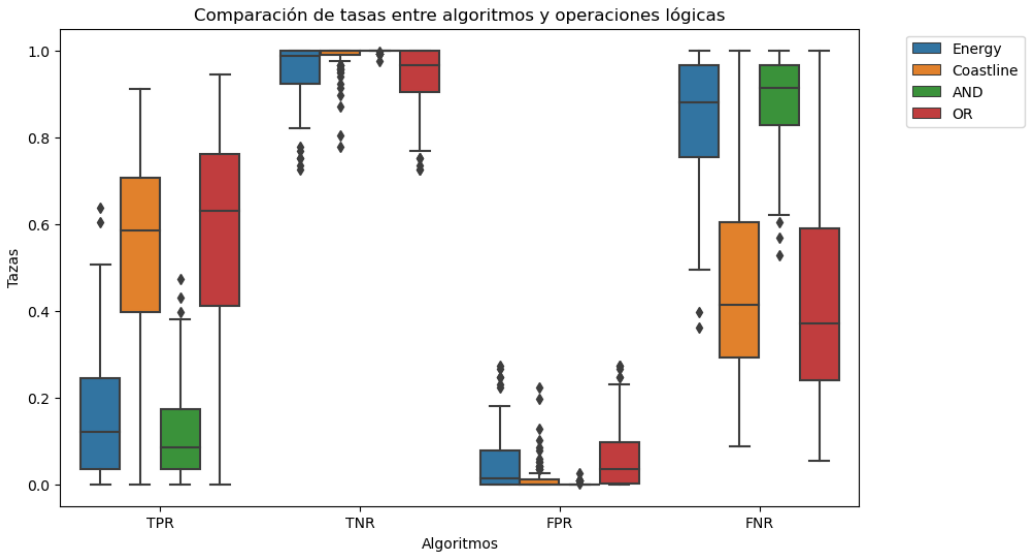


Figura 3.24: Tasas de *TP*, *TN*, *FP*, *FN* para algoritmos Energy, Coastline y sus posibles combinaciones lógicas.

La figura evidencia que las operaciones lógicas tienen los efectos esperados. Por el lado de la operación OR, se espera que detecte mayor cantidad de crisis. Esto genera un claro aumento en *TPR*, del cual ambos algoritmos se ven beneficiados, el Energy en mayor medida. Esto sin embargo se ve compensado por un aumento en el *FPR*. Esto parece indicar que para el algoritmo Energy, el aumento drástico en verdaderos positivos resulta en una mejor detección. No obstante, para el algoritmo Coastline, el pequeño aumento en *TPR* se ve eclipsado por el aumento, en similar medida, del *FPR*. Esto, según los resultados observados en la Figura 3.23, resulta en que la operación lógica OR no despliega una mejor capacidad de detección que el algoritmo Coastline trabajando de forma aislada.

Por otro lado, el aumento de *TPR* por si solo puede ser beneficioso, a pesar de la disminución en AUROC. En una aplicación de detección de crisis epilépticas, es importante detectar la mayor cantidad de episodios posibles, aunque esto signifique un aumento en falsos positivos. En este contexto, la operación OR puede resultar útil.

En el caso de la operación AND se espera que esta detecte una menor cantidad de crisis. En este caso el efecto es aún mas claro. El bajo *TPR* del algoritmo Energy genera que la operación resulte en una disminución significativa de verdaderos positivos. Mientras tanto, la cantidad de falsos positivos es baja para ambos algoritmos, por lo que el efecto de la operación es despreciable. Esto genera que la

capacidad de detección al utilizar la operación AND disminuya en comparación al algoritmo Coastline trabajando individualmente, evidenciado en la Figura 3.22.

En conclusión, al analizar los resultados de las operaciones AND y OR, se evidencia que las combinaciones que incluyen el algoritmo Coastline, especialmente en la operación OR, ofrecen mayor capacidad de detección en términos de AUROC. Aunque las combinaciones AND proporcionan ciertos beneficios, su eficacia se encuentra limitada en comparación con el Coastline individual, que destaca por su robustez y eficacia en la detección. Por lo tanto, se puede afirmar que, para mejorar el la AUROC, es preferible utilizar la operación OR con Coastline, ya que este algoritmo, tanto por sí solo como en combinación, supera a los demás en todas las configuraciones evaluadas. Finalmente, la operación OR parece aumentar el valor de *TPR*, lo cual le da mas relevancia en una aplicación donde interesa detectar la mayor cantidad de episodios posible.

3.10. Comparación de ventanas

Como ya fue mencionado, la cantidad de muestras que conforman una ventana en el algoritmo es un parámetro de vital importancia, ya que no solo determina la cantidad de información a considerar a la hora de detectar, sino el tiempo que demora en reaccionar el sistema ante una crisis.

Por esta razón, el siguiente estudio buscó implementar los algoritmos con diferentes tamaños de ventanas, de forma análoga al estudio realizado en la Sección 3.4, con una mayor cantidad de señales, listadas en la Tabla B.2.

A modo de ejemplo, se presentan los resultados del estudio para el algoritmo Energy en la Figura 3.25; sin embargo, todos los resultados completos pueden consultarse en el Apéndice D.

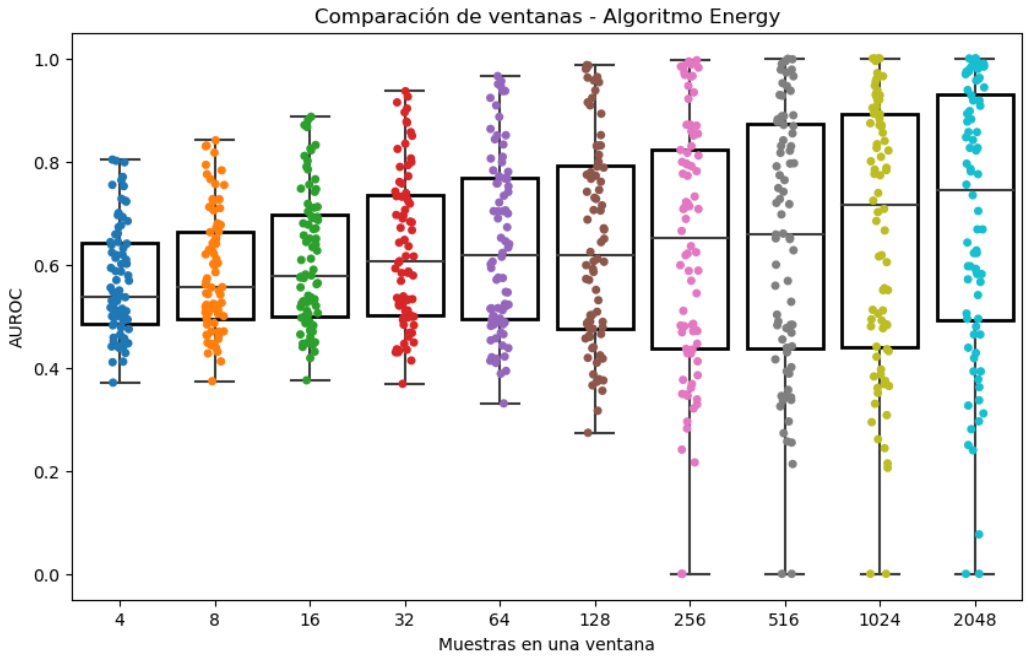
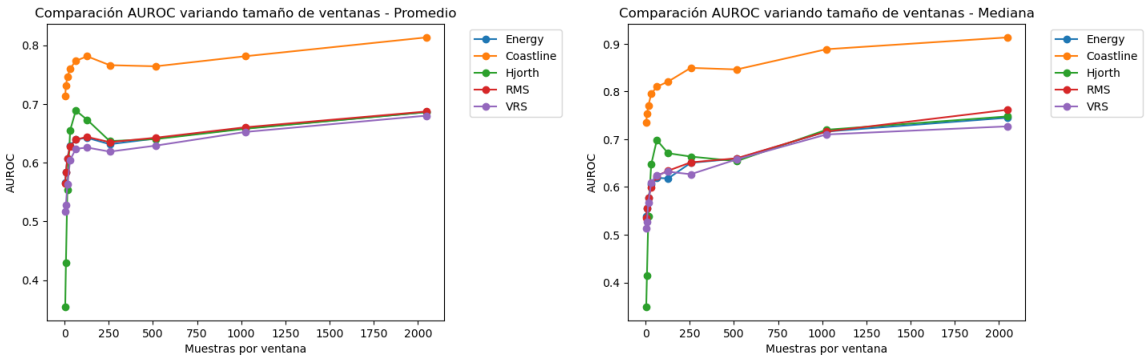


Figura 3.25: Análisis comparativo de tamaño de ventanas para algoritmo Energy.

En todos los algoritmos analizados, se observa que a medida que se incrementa la cantidad de muestras en la ventana, la AUROC obtenida mejora significativamente. Por ejemplo, en el caso del algoritmo Energy, se inicia con un tamaño de ventana de 4 muestras, donde se obtiene un valor de AUROC inferior al 60 %. Sin embargo, al aumentar progresivamente el tamaño de la ventana, se alcanzan valores de AUROC superiores al 60 % a partir de 64 muestras.

De forma análoga al estudio realizado en la Sección 3.5, se calculo el promedio y la mediana de los datos para su mejor visualización, presentados en las figuras 3.26a y 3.26b, respectivamente.



(a) Promedios. (b) Medianas.

Figura 3.26: Análisis comparativo de tamaño de ventanas.

Se puede observar que, tanto para el promedio como para la mediana, a medida que aumenta el tamaño de la ventana la AUROC mejora. En la gráfica de promedio se puede ver que la AUROC es mayor en algunos casos previos a la ventana de 256 muestras para varios algoritmos. Por otra parte, esto no sucede en la gráfica de mediana, exceptuando el algoritmo Hjorth. Como se ha mencionado previamente, la mediana es una mejor forma de representar los datos cuando estos presentan mayor dispersión. Además, la AUROC en el caso de la mediana es mayor. Finalmente, a partir de 256 muestras, la mejora en la AUROC se vuelve gradual. Con el objetivo de optimizar el tiempo de reacción del algoritmo y considerando esta tendencia, se optó por trabajar con un tamaño de ventana de 256 muestras.

El algoritmo Hjorth por otro lado presenta mayor capacidad de detección para ventanas mas pequeñas, tanto en su promedio como en su mediana. Esto permite disminuir el tiempo de reacción y maximizar la capacidad de detección de manera simultánea. En el caso de que fuera a implementarse, la mejor opción es optar por un tamaño de ventana de 64 muestras, que presenta los mejores niveles de detección. De forma de mantener iguales condiciones de comparación con los demás algoritmos, en posteriores estudios se utilizará una ventana de 256 muestras, teniendo en mente que no presentará el mayor nivel de AUROC del que el algoritmo es capaz.

Luego, se presentan los resultados del estudio de ventana para el algoritmo NLA en la Figura 3.27.

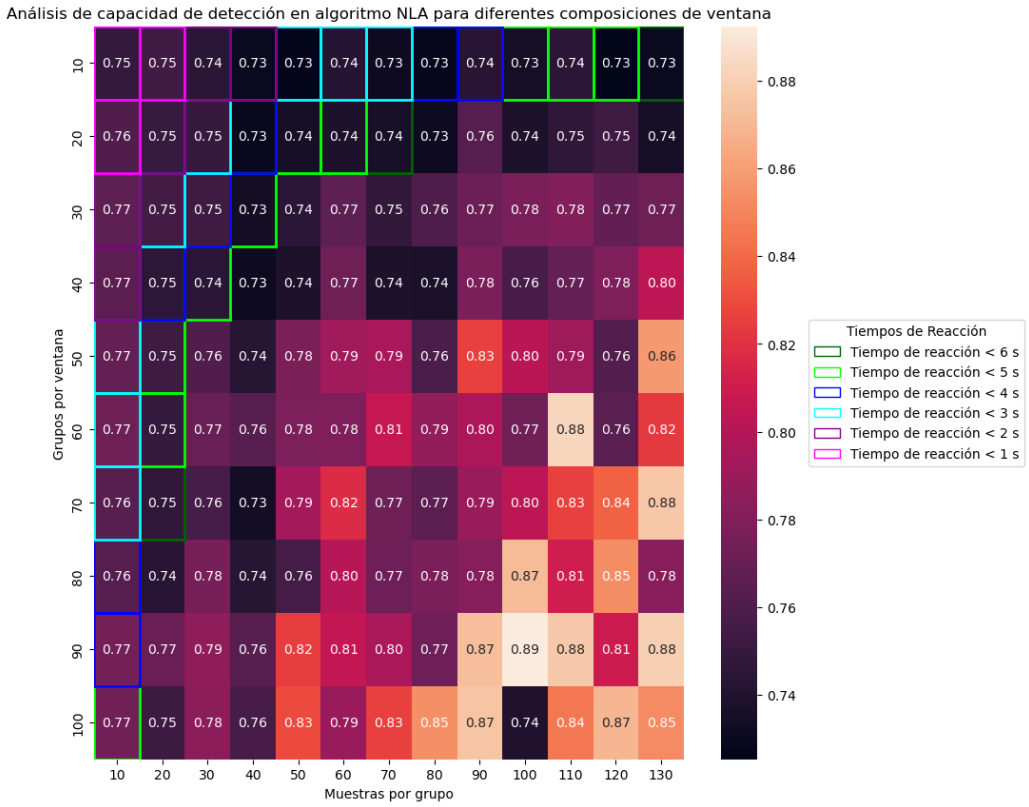


Figura 3.27: Mapa de calor de promedios de AUROC. La capacidad de detección se contrasta con tiempos de reacción del algoritmo de acuerdo a la cantidad de muestras en una ventana.

Los resultados muestran diferencias importantes en comparación con las pruebas iniciales realizadas en 3.8. La tendencia al aumento de la AUROC se visualiza en ambos casos, sin embargo, su tasa de crecimiento es diferente. Se observa que en el nuevo análisis los valores de AUROC no llegan a niveles tan altos en comparación al análisis inicial, y para alcanzar estos valores elevados, el tiempo de reacción debe ser más alto. Por ejemplo, para alcanzar un 80 % de AUROC en el estudio inicial era necesario un tiempo de reacción mínimo de 2.4 s. Al analizar una mayor cantidad de muestras, el tiempo de reacción mínimo alcanza 16.8 s.

Para ventanas de menor tamaño, se observa un aumento en los valores de AUROC, lo que sugiere una mejora en la capacidad de detección de los algoritmos. Por ejemplo, si consideramos el tamaño de ventana seleccionado al final de la Sección 3.4, la capacidad de detección promedio del algoritmo incrementa del 68 % al 77 %. Sin embargo, es importante señalar que aumentos pequeños en el tamaño de la ventana no siempre benefician la capacidad de detección; en algunos casos, incluso pueden reducir la efectividad del algoritmo, según la Figura 3.27. Esto subraya la importancia de encontrar un tamaño de ventana óptimo para maximizar la precisión de detección.

Es importante aclarar que muchas de las señales utilizadas tienen una duración de 3 minutos, lo que implica que, al utilizar tamaños de ventana más grandes, la cantidad de ventanas que pueden ser analizadas es menor, lo cual podría comprometer la precisión de los resultados. Por ejemplo, para el tamaño de ventana más grande, que incluye 100 grupos de 130 muestras cada uno, solo es posible analizar 3 ventanas en una señal de 3 minutos. Además, estos tamaños de ventana grandes resultan en tiempos de reacción demasiado elevados para una aplicación médica. Por esta razón, los análisis posteriores se centrarán en ventanas más pequeñas, que permiten una mejor capacidad de respuesta y una mayor precisión en la detección.

Finalmente, el punto que maximiza la AUROC y disminuye el tiempo de reacción simultáneamente es la ventana conformada por 30 grupos de 10 muestras cada uno.

3.11. Superposición de ventanas

La implementación de los algoritmos Energy, Coastline, Hjorth, RMS Amplitude y VRS permiten la utilización de ventanas móviles, lo cual implica la superposición de muestras entre ventanas.

Esta configuración permite disminuir el tiempo de reacción sin cambiar el tamaño de ventana. Siendo este un parámetro importante en la detección de crisis epilépticas, es necesario identificar si su implementación mejora, o en su defecto disminuye, la capacidad de detección de los algoritmos. A raíz de esto, se realizó un estudio donde el tamaño de ventana del algoritmo se mantiene constante y varía la cantidad de muestras que se superponen en cada ventana.

Se utilizó un tamaño de ventana de 256 muestras, de acuerdo a lo concluido en la Sección 3.10.

Como se ha hecho anteriormente, se presentan los resultados del algoritmo Energy como ejemplo en la Figura 3.28. Todos los gráficos fueron anexados en el Apéndice E.

En el caso del algoritmo Energy, se observa que la AUROC permanece prácticamente constante a medida que se modifica la cantidad de muestras superpuestas en la ventana. Este patrón se repite en los demás algoritmos analizados, que también muestran resultados similares en cuanto a la estabilidad de la AUROC frente a variaciones en la superposición de muestras. Adicionalmente, se calculan los promedios y medianas de los datos y se grafican individualmente por algoritmo para facilitar la visualización de los resultados.

En las gráficas de promedios y medianas que se muestran a continuación, se evidencia que cada algoritmo que emplea ventanas no superpuestas alcanza una

AUROC superior en comparación con aquellos que utilizan ventanas superpuestas. Al mismo tiempo, no disminuye de forma significativa, por lo tanto, puede ser una herramienta útil para disminuir el tiempo de reacción del dispositivo sin modificar el tamaño de ventana.

Sin embargo, este recurso requiere almacenar valores ya procesados de la señal, lo cual en Hardware implica la utilización de memoria. En este punto del análisis, se supone que esto aumenta el consumo de forma considerable, posiblemente dejando sin utilidad esta alternativa. En análisis posteriores se pretende estudiar esta implicancia al medir el consumo del algoritmo RVS, el cual utiliza memoria.

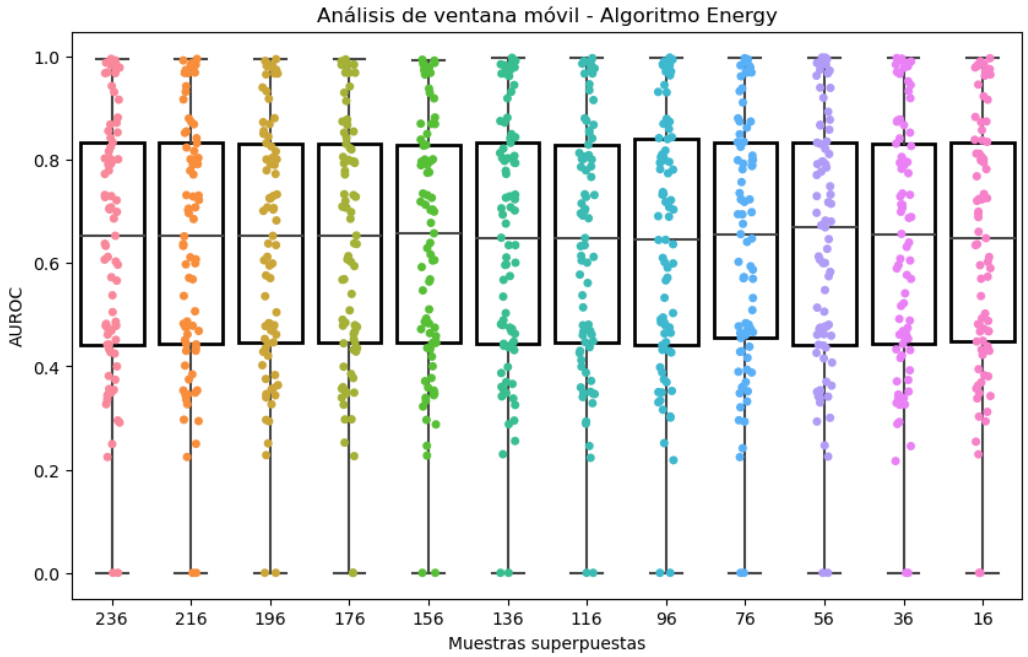
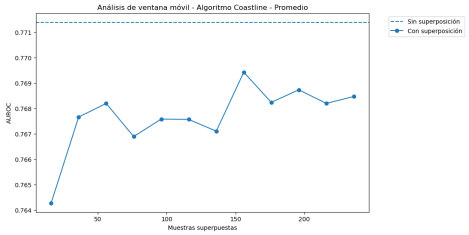
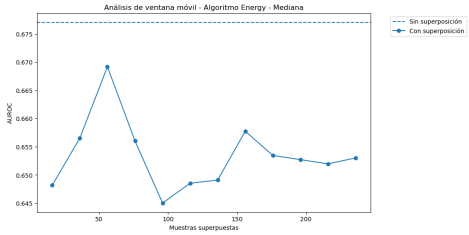


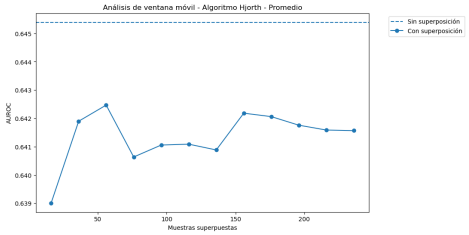
Figura 3.28: Análisis comparativo de superposición de muestras para algoritmo Energy.



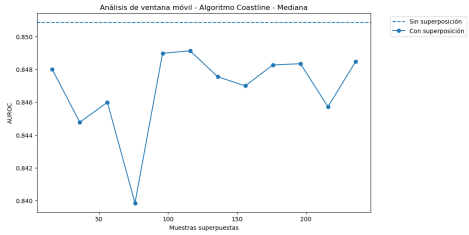
(a) Energy - Promedio.



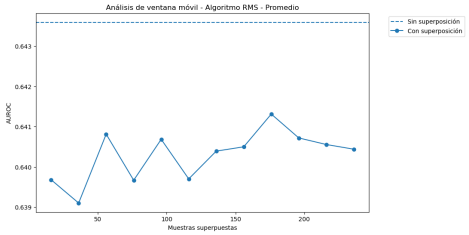
(b) Energy - Mediana.



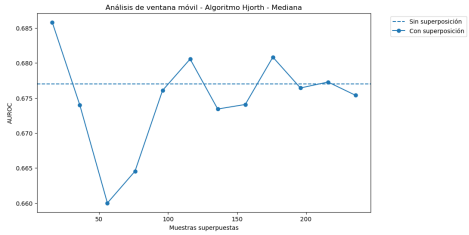
(c) Coastline - Promedio.



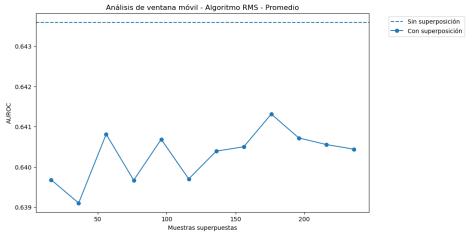
(d) Coastline - Mediana.



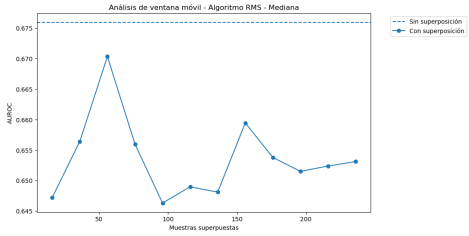
(e) Hjorth - Promedio.



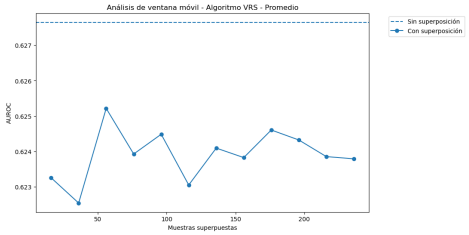
(f) Hjorth - Mediana



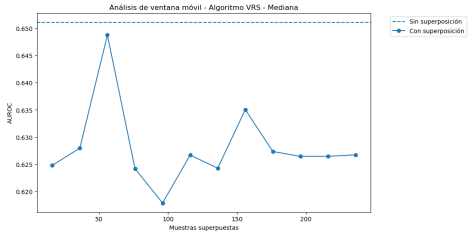
(g) RMS Amplitude - Promedio.



(h) RMS Amplitude - Mediana.



(i) VRS - Promedio.



(j) VRS - Mediana

Figura 3.29: Análisis de superposición de muestras en ventanas.

3.12. Cantidad de bits para representación de datos.

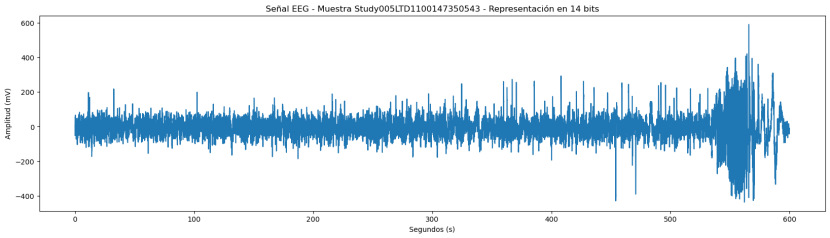
Un aspecto que condiciona fuertemente al sistema al momento de la implementación en Hardware es la cantidad de bits que se requieren para manejar las señales, debido a que el área del FPGA destinada al diseño depende de esto directamente. Luego, si se requiere mayor área, el sistema tenderá a un mayor consumo.

La cantidad de bits que se requiere se vio inicialmente determinada por la amplitud de las señales con las que se trabaja, específicamente las listadas en el Apéndice B. Además, debido a la presencia de valores negativos en las señales, se debió contemplar una representación con signo. Para el presente trabajo, se utilizó la representación en complemento a 2. Considerando estos requisitos, 14 bits fueron necesarios para representar todas las señales sin modificarlas.

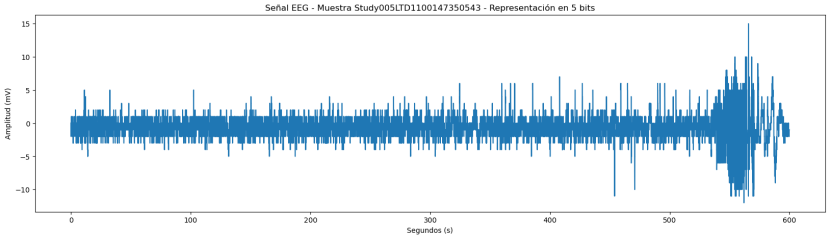
Dicho esto, la señal de entrada puede ser modificada para que se represente en una menor cantidad de bits, efectivamente disminuyendo el espacio de FPGA necesario para el diseño y, en consecuencia, el consumo. Sin embargo, esta modificación disminuye considerablemente la resolución de la señal a medida que se disminuye la cantidad de bits con los que se la representa. El proceso de disminución de bits se aprecia en la Figura 3.30, utilizando la señal de la Figura 3.1.

Una vez obtenidas las señales modificadas de acuerdo a la cantidad de bits que se disponen para su representación, se procedió a realizar la evaluación de capacidad de detección de los algoritmos para las señales en las listas B.1 y B.2. Estos estudios se realizaron en las condiciones determinadas al final de la Sección 3.4. El propósito de este análisis no fue comparar la AUROC entre algoritmos, sino evaluar hasta que punto se pueden disminuir los bits de representación de datos sin perder eficacia.

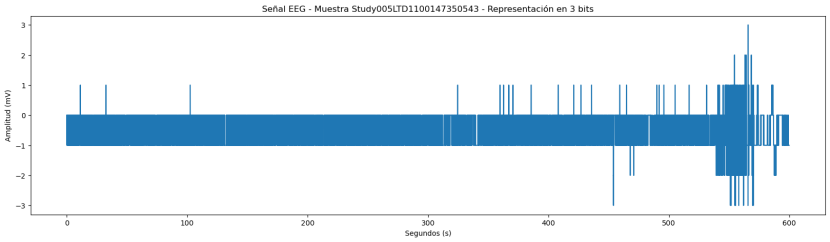
A modo de ejemplo se presenta la gráfica del análisis para el algoritmo Energy en la Figura 3.31. Todas las gráficas se pueden encontrar en el Apéndice F.



(a) EEG de muestra Study005LTD1100147350543 con representación en 14 bits.



(b) EEG de muestra Study005LTD1100147350543 con representación en 5 bits.



(c) EEG de muestra Study005LTD1100147350543 con representación en 3 bits.

Figura 3.30: Visualización de procesos de disminución de bits para representación de datos - Señal EEG - Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs.

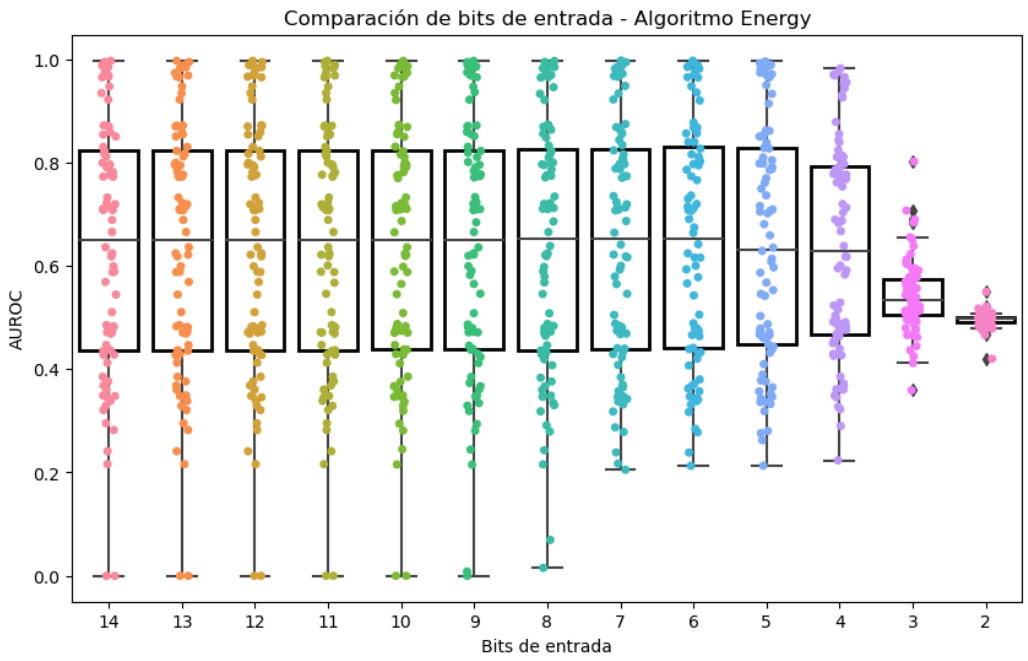
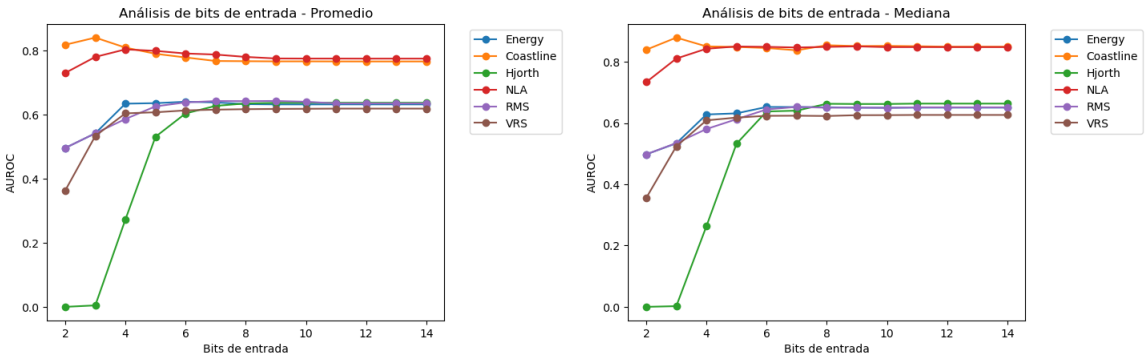


Figura 3.31: Análisis de capacidad de detección de algoritmo Energy al cambiar cantidad de bits para representación de datos.

De forma análoga a secciones anteriores, se presentan los datos calculando su promedio y mediana para cada valor de bits en las figuras 3.32a y 3.32b.



(a) Promedios (b) Medianas

Figura 3.32: Análisis comparativo de capacidad de detección de algoritmos al cambiar cantidad de bits para representación de datos.

En los algoritmos Energy, Hjorth, RMS Amplitude y VRS se observa un comportamiento esperado. La capacidad de detección se mantiene constante hasta que alcanza cierta cantidad de bits. A partir de este momento la AUROC disminuye consistentemente, hasta que alcanza, y en el caso de los algoritmos Hjorth y VRS, atraviesa la barrera de 50 %. Esto evidencia que para este grupo de algoritmos la cantidad de bits con los que se puede representar los datos es limitada, ya que la pérdida de resolución en la señal afecta negativamente la capacidad de detección en el procesamiento.

Por otro lado, tenemos el conjunto de algoritmos Coastline y NLA, para los cuales se observa un comportamiento diferente. En este caso, la capacidad de detección de los algoritmos no disminuye hasta el punto de volverlos inútiles. En el algoritmo NLA se aprecia una caída en la AUROC a partir de los 4 bits, pero no es considerable si se compara con el otro grupo de algoritmos.

El algoritmo Coastline presenta un comportamiento particular. Este no disminuye su eficacia a medida que decae la cantidad de bits. Incluso, se aprecia un aumento de la AUROC para 3 bits de representación de datos en comparación a todos los otros casos. Esto es un resultado importante, ya que si el diseño puede implementarse con solo 2 bits, el área que ocuparía en el FPGA disminuye considerablemente, y en consecuencia, también lo haría el consumo del sistema.

El comportamiento de los algoritmos NLA y Coastline se debe probablemente a su mecanismo, similar a lo explicado en la Sección 3.8, y a la forma de la señal cuando se representa con pocos bits. Si se toma como ejemplo la señal vista en la Figura 3.1, su representación en 2 bits se muestra en la Figura 3.33. Se puede

observar como cambiar la representación de los datos afecta a la señal de salida del algoritmo en la Figura 3.34.

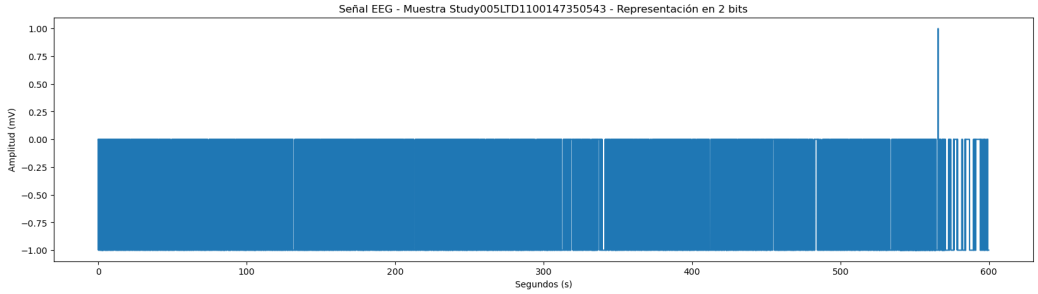
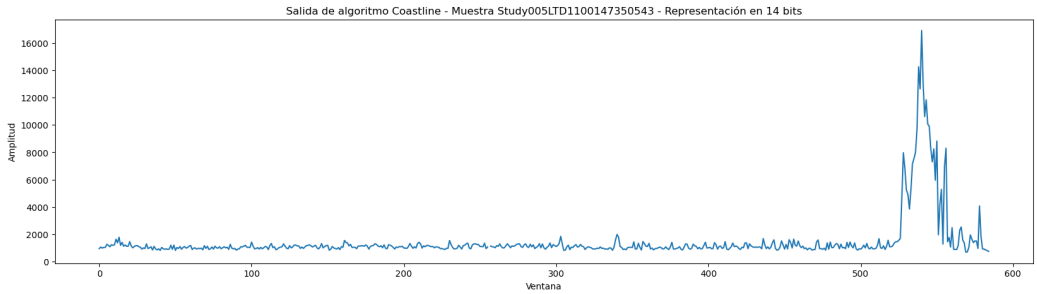
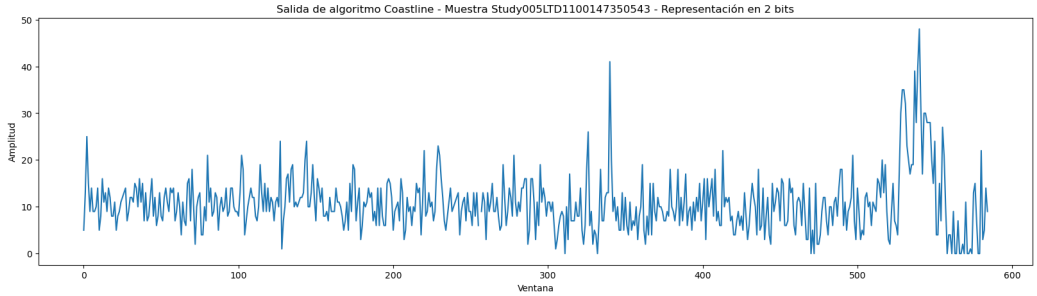


Figura 3.33: EEG de muestra Study005LTD1100147350543 con representación en 2 bits.



(a) Representación en 14 bits.

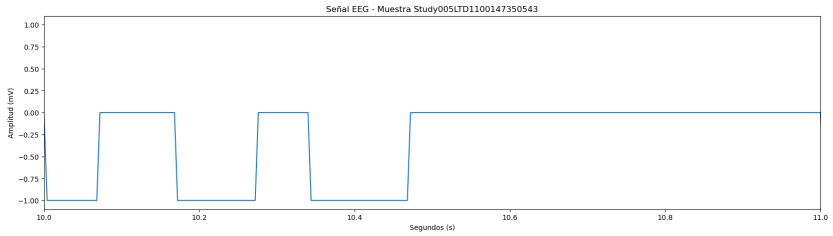


(b) Representación en 2 bits

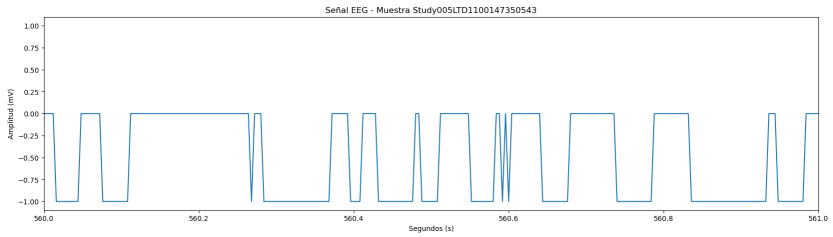
Figura 3.34: Comparación de salida de algoritmo Coastline al inyectar una señal representada con 14 bits contra una señal representada con 2 bits.

A pesar de exhibir una mayor actividad cuando la señal está en reposo, la respuesta del algoritmo ante la crisis se distingue claramente en la 3.34b. Es posible que con una elección correcta de umbral, la evaluación de detección en ambos casos sea similar, con un pequeño aumento en falsos positivos en el caso de la representación de 2 bits.

Además, se muestra un acercamiento de la señal representada con 2 bits en la Figura 3.35, mostrando dos intervalos de tiempo. En uno de ellos se presenta una crisis y en el otro no.



(a) Acercamiento en intervalo de tiempo 10-11 segundos - Sin crisis.



(b) Acercamiento en intervalo de tiempo 560-561 segundos - Con crisis.

Figura 3.35: Visualización de señal con representación en 2 bits - Señal EEG - Estudio 005 - Canal LTD1 - Tiempo de inicio 100147350543 μs. Comparación de intervalos de tiempo donde hay crisis y no hay crisis.

Como fue mencionado anteriormente, los algoritmos Coastline y NLA son sensibles a los cambios de alta frecuencia que ocurren dentro de cada ventana. Si se observan ambos acercamientos, se aprecia que en las zonas de crisis, la frecuencia con la que alterna la señal es mayor a la de la zona sin crisis. Por otro lado, la amplitud de la señal no se ve alterada, por lo que los demás algoritmos no podrán detectar los cambios correctamente. Esto es lo que se observa en la Figura 3.32.

A modo de conclusión, disminuir la cantidad de bits que se utilizan para representar los datos a 6 no parece disminuir la capacidad de detección de ninguno de los algoritmos. A partir de este punto, los algoritmos Energy, Hjorth, RMS Amplitude y VRS empeoran su eficacia, hasta el punto de volverse no funcionales. Por otro lado, los algoritmos Coastline y NLA tienen un comportamiento diferente. El algoritmo NLA disminuye su capacidad de detección como los demás, pero aún así siegue siendo elevada. Esto lo vuelve un opción viable, considerando que si solo se utilizan 2 bits, el consumo sería mínimo. Finalmente, el algoritmo Coastline no pierde capacidad de detección, en cambio, presenta un máximo para 3 bits. Esto implica que el algoritmo permite minimizar el consumo sin perder capacidad de detección, lo cual es un resultado muy positivo.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Implementación en hardware

En todos los pasos descritos a continuación, se utilizó el software Quartus, una herramienta desarrollada por Altera para el análisis y la síntesis de diseños en VHDL. Los sistemas desarrollados se sintetizan e implementan utilizando un FPGA de la familia de Cyclone III que se encuentra dentro de la placa DE0.

4.1. Lectura de datos

De manera de verificar el correcto funcionamiento e implementación del diseño, es prudente comenzar leyendo datos desde una memoria ROM. Este enfoque inicial permite aprovechar la estabilidad y consistencia de los datos predefinidos, facilitando la depuración y el ajuste de los algoritmos. Además, permite realizar pruebas exhaustivas y sistemáticas, comparando los resultados con las simulaciones realizadas en Python para garantizar la precisión y corregir errores de manera eficiente.

La utilización de una memoria ROM con datos estáticos proporciona un entorno controlado que simplifica la identificación de problemas y el ajuste fino de los algoritmos. Esta etapa es crucial para asegurar que los algoritmos funcionan correctamente antes de introducir la variabilidad de los datos reales, es decir, los datos obtenidos a partir de las señales enlistadas en el Apéndice B. Sin embargo, es importante reconocer que los datos predefinidos pueden no cubrir todas las posibles variaciones y escenarios que se presentarán en un entorno real.

Para llevar a cabo esta implementación, dividimos el proceso de simulación de llegada de datos en distintas etapas:

- Lectura desde una memoria ROM con datos ficticios.
- Lectura desde una memoria ROM con datos reales.

4.1.1. Memoria ROM

En la fase de diseño del proyecto, se implementó una memoria ROM utilizando el software Quartus para la lectura de datos. En esta primera instancia se

corroboró el correcto funcionamiento de los algoritmos cargando en la ROM datos ficticios, verificando que los resultados obtenidos fueran los esperados. Posteriormente se cargaron datos reales en la ROM teniendo en cuenta la precisión y la representación de los mismos.

La ROM implementada tiene una capacidad de 4096 palabras, lo que permite cargar múltiples ventanas de datos y aplicar los algoritmos a varias ventanas simultáneamente.

4.1.2. Representación de datos

La representación de los datos y su precisión son características imprescindibles para la correcta aplicación de los algoritmos. Dada la base de datos utilizada fue necesario elegir una cantidad de bits que representara en su totalidad los datos.

La base de datos contiene números decimales, por ende para representarlos en binario necesitaríamos utilizar punto flotante, esta representación puede generar complicaciones en las operaciones. Para simplificar este problema se decidió utilizar números enteros, comprobando con Pyhton si esto afecta la capacidad de detección de los algoritmos. Se tomó la decisión de redondear los datos previo al procesamiento y en la Sección 3.7 se verificó que los resultados obtenidos para la AUROC no difieren. Como consecuencia, se optó por redondear los datos.

Para representar todo el rango de las señales mencionadas en las listas B.1 y B.2 se optó por una selección de 12 bits para la representación de muestras. Para las señales cuyo rango no puede ser representado adecuadamente, por ejemplo las mencionadas en la lista B.3, se utilizará una representación de 14 bits. Sin embargo, independientemente de la cantidad de bits elegida, si alguna señal presenta valores fuera del rango permitido para esos bits, se reemplazarán por los valores máximos o mínimos representables, dependiendo de si se trata de números positivos o negativos. Esta modificación no afectará de manera significativa los resultados de los algoritmos.

Para generalizar el sistema y facilitar su adaptación a diferentes tipos de señales, se definieron variables clave que especifican los tamaños de las señales. Estas incluyen: **bits_in**, que determina el número de bits del dato de entrada al algoritmo; **counter_size**, que define la cantidad de bits necesarios para representar el tamaño de la ventana del algoritmo; **ROM_size**, que establece los bits para el tamaño de la ROM; y **window**, que representa la cantidad de bits en 1 necesarios para definir el tamaño de la ventana en potencias de dos. Al definir estas variables, el sistema se adapta automáticamente a las señales deseadas, sin necesidad de realizar otros cambios en la configuración, lo que garantiza su funcionamiento correcto en distintos contextos.

Dado que varios de los algoritmos realizan operaciones de elevación al cuadra-

do y posteriormente calculan el promedio de esos cuadrados, se determinó que la cantidad mínima de bits necesarios para representar estas operaciones sin desbordamientos es de **bits_in** * 2 + **window**.

Además, para definir la cantidad de bits del límite de la ventana, se utilizan las variables **counter_size** y **window**, especificándose de la siguiente manera: limit [counter_size - 1 .. window] = 0, limit[window-1 .. 0] = 1.

4.1.3. Simulación de llegada de datos

De manera de simular la llegada de datos del diseño, se decidió escribir en tiempo real la memoria ROM utilizando un script en lenguaje TCL. Este lenguaje mostrado en el Apéndice G carga en el sistema archivos **.mif**, los cuales contienen diferentes ventanas de datos. Esto se ejecuta en tiempo real de manera de observar en la placa los diferentes resultados obtenidos de cada ventana.

Se definió un bloque llamado **ME_ROM**, el cual simula la llegada de datos en tiempo real del sistema mediante una máquina de estados, la memoria y un contador, el cual presenta el funcionamiento mostrado en la Figura 4.1. Este bloque cuenta con dos salidas, **ready_data**, que indica que hay un dato disponible para procesar, y **data[11..0]** que contiene el dato en sí. En el futuro, si se utiliza otro sistema para simular la llegada de datos, será suficiente con reemplazar éste por el nuevo bloque, asegurando que las salidas coincidan con las mencionadas.

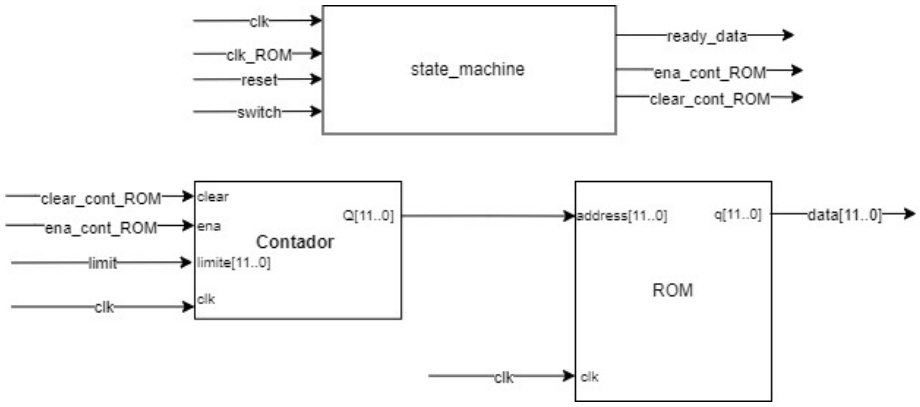


Figura 4.1: Diagrama de bloques de ME_ROM.

Se cuenta con dos relojes: **clk**, que es el reloj principal del sistema, y **clk_ROM**, que marca el tiempo de llegada de cada dato. Este último es más lento que el reloj del sistema.

La máquina de estados opera según el diagrama mostrado en la Figura 4.2. En el estado **rest**, se espera un pulso de **clk_ROM**. Al recibir este pulso, se transita al estado **active**, donde se notifica al sistema que un dato está listo para ser procesado, y luego se avanza al estado **count**, en el cual se habilita el contador

de la ROM para acceder al siguiente dato. Si se alcanza el final de la memoria, se pasa al estado **update**, donde se espera la carga de un nuevo archivo **.mif** con nuevas ventanas de datos para procesar, y se reinicia el contador. Una vez cargado el nuevo archivo, el usuario avisa al sistema mediante un switch en la placa y la máquina de estados se mueve al estado de **rest**. Finalmente, existe un estado **state_reset**, en el que se restablece el contador.

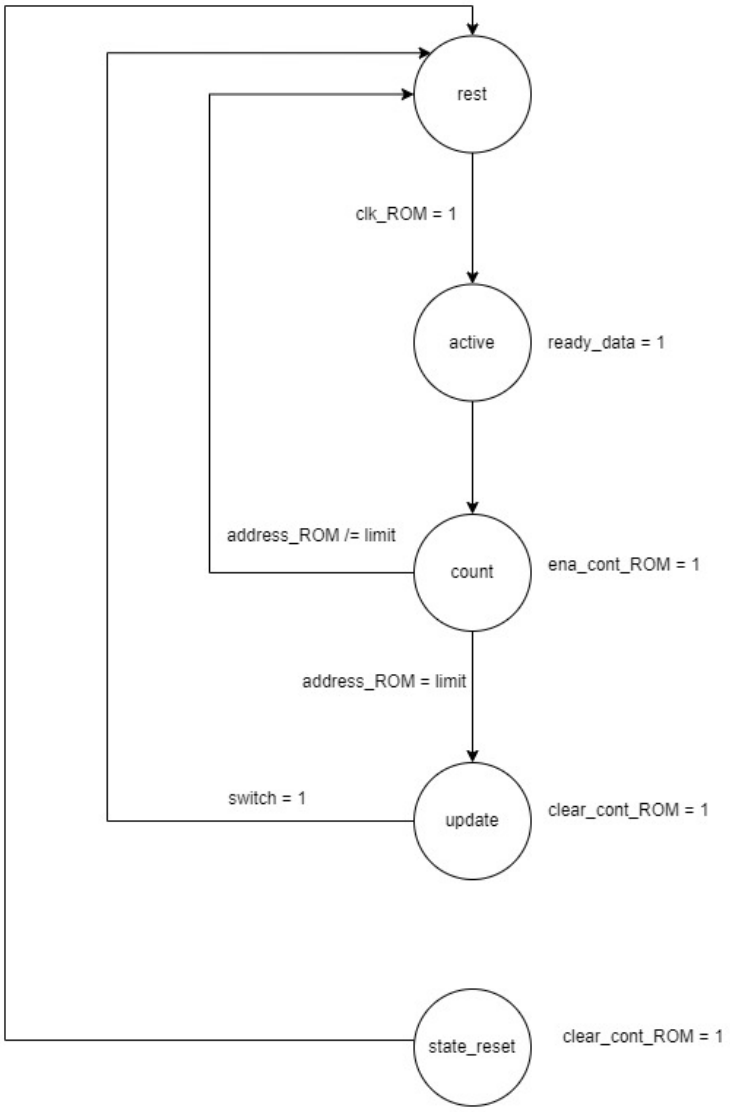


Figura 4.2: Máquina de estados del bloque ME_ROM.

4.2. Algoritmos

En esta sección se describen las consideraciones clave relacionadas con el diseño y funcionamiento de los algoritmos descritos en el Capítulo 2.

4.2.1. Tamaño de ventana

De acuerdo a lo determinado en la Sección 3.4, para los algoritmos Energy, Coastline, Hjorth, RMS Amplitude y VRS se utilizará un tamaño de ventana de 256 muestras.

Por otro lado, para el algoritmo NLA la ventana se compondrá de 30 grupos, conformados por 10 muestras cada uno.

4.2.2. Bloques aritméticos

Para realizar las operaciones demandadas por los algoritmos, tales como la multiplicación, el promedio, el cual implica una división por el tamaño de la ventana y la raíz cuadrada, inicialmente se utilizaron bloques de Quartus de manera de simplificar las operaciones. Sin embargo, estos bloques solamente funcionan en Altera, por ende si se desea utilizar este proyecto en otro FPGA que no sea de Altera, las operaciones serían obsoletas.

Para asegurar la portabilidad del sistema, se decidió crear entidades para las operaciones. Dentro de estas entidades, se encuentra instanciado el bloque correspondiente creado en Quartus. De esta manera, si se desea utilizar otro programa basta con modificar solamente el VHDL del bloque y no los algoritmos ni el programa principal.

Por otro lado, en el caso del bloque divisor, todos los algoritmos dependen de un único cociente que se representa como una potencia de dos, como se mencionó anteriormente. Por esta razón, se decidió eliminar el bloque divisor creado anteriormente, ya que dividir entre una potencia de dos equivale a realizar un desplazamiento ($shift$) del numerador.

4.2.3. Implementación en VHDL

En el Capítulo 2 se puede observar que los algoritmos comparten una operación común: la acumulación. De manera de emplear un sistema que optimice la cantidad de bloques, se decidió emplear acumuladores compartidos entre los algoritmos. Así, cada bloque de algoritmo se encarga únicamente de su operación específica, habilitando los acumuladores solo cuando sea necesario. Dado que dos de los seis algoritmos requieren calcular la media de la señal, se utilizan dos acumuladores: uno para acumular el resultado de la operación y otro para calcular la media. Se muestra un diagrama de bloques de esta implementación en la Figura 4.3.

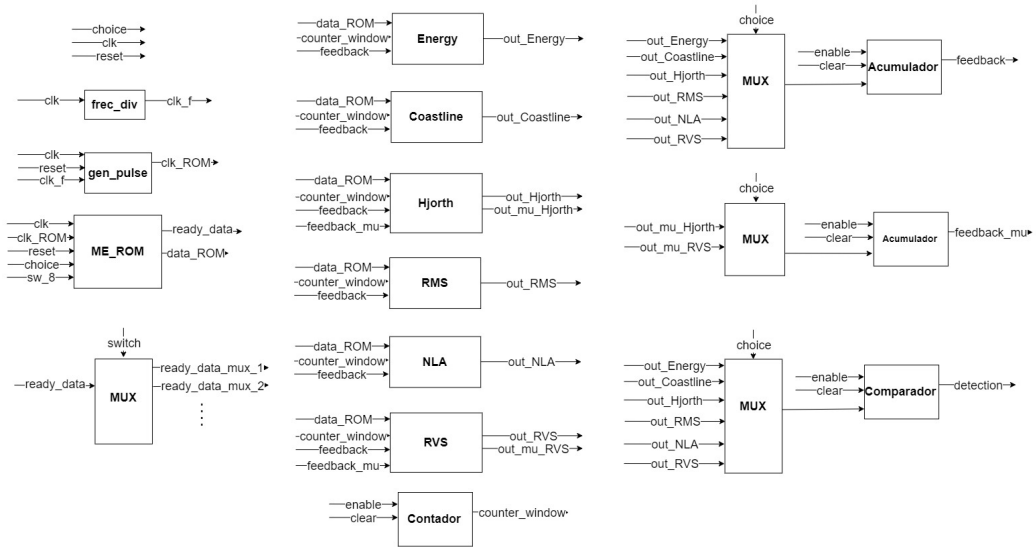


Figura 4.3: Diagrama de bloques con acumuladores y comparador compartidos. Los mismos se habilitan mediante multiplexores cuando corresponda.

Durante la implementación, se diseñaron máquinas de estados para gestionar las señales de control, lo que garantizó la sincronización adecuada y mejoró la eficiencia en el manejo de estas señales. En el caso de Energy, Coastline, Hjorth, y RMS Amplitude, se utilizó la misma máquina de estados, cuyo funcionamiento se detalla en el diagrama de la Figura 4.4.

En este caso, se incluye un estado llamado **rest**, donde espera la habilitación por parte del sistema. Una vez recibida esta habilitación, se transita al estado **operation**, en el cual se ejecuta la operación correspondiente y se habilita el acumulador. Si la señal *counter*, que lleva el conteo de muestras en la ventana, alcanza su límite, se avanza al estado **window_end**, donde se habilita el comparador y se reinician los acumuladores para realizar la comparación con el umbral correspondiente. Finalmente, existe un estado de **state_reset** en el que se restablecen tanto el contador como el acumulador.

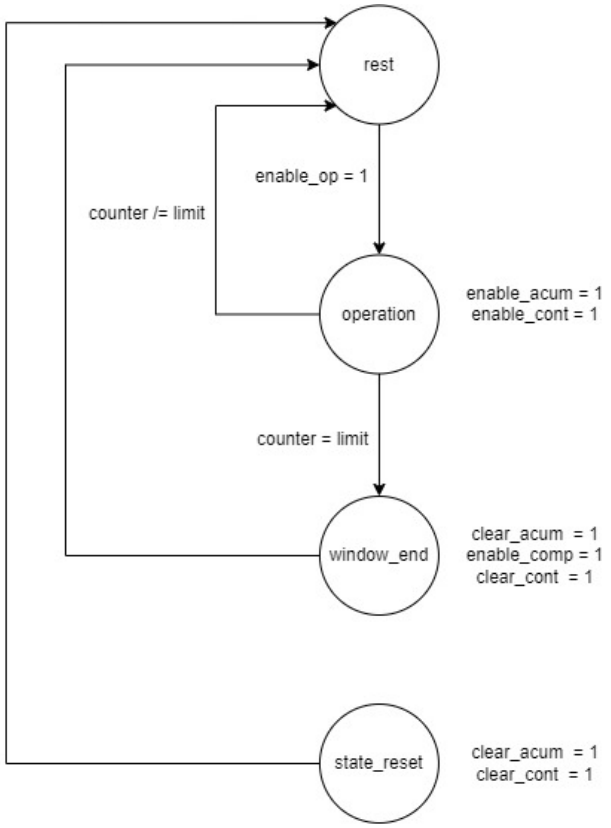


Figura 4.4: Máquina de estados de algoritmos Energy, Coastline, RMS Amplitude y Hjorth.

En el caso del algoritmo NLA, se implementó una máquina de estados diferente, ya que este algoritmo opera con grupos de muestras, lo que altera su funcionamiento respecto a los algoritmos mencionados anteriormente. Se añadieron dos estados adicionales: **state_group**, donde se cuentan las muestras que conforman cada grupo; cuando el contador de muestras alcanza su valor límite, el proceso avanza al estado **operation**. Si no se alcanza el límite, se regresa al estado **rest** para esperar el siguiente dato. El segundo estado, **compare**, se encarga de realizar las comparaciones entre los máximos y mínimos correspondientes (véase Figura 4.5). En este estado, se habilita el comparador y se reinician tanto los contadores como el acumulador.

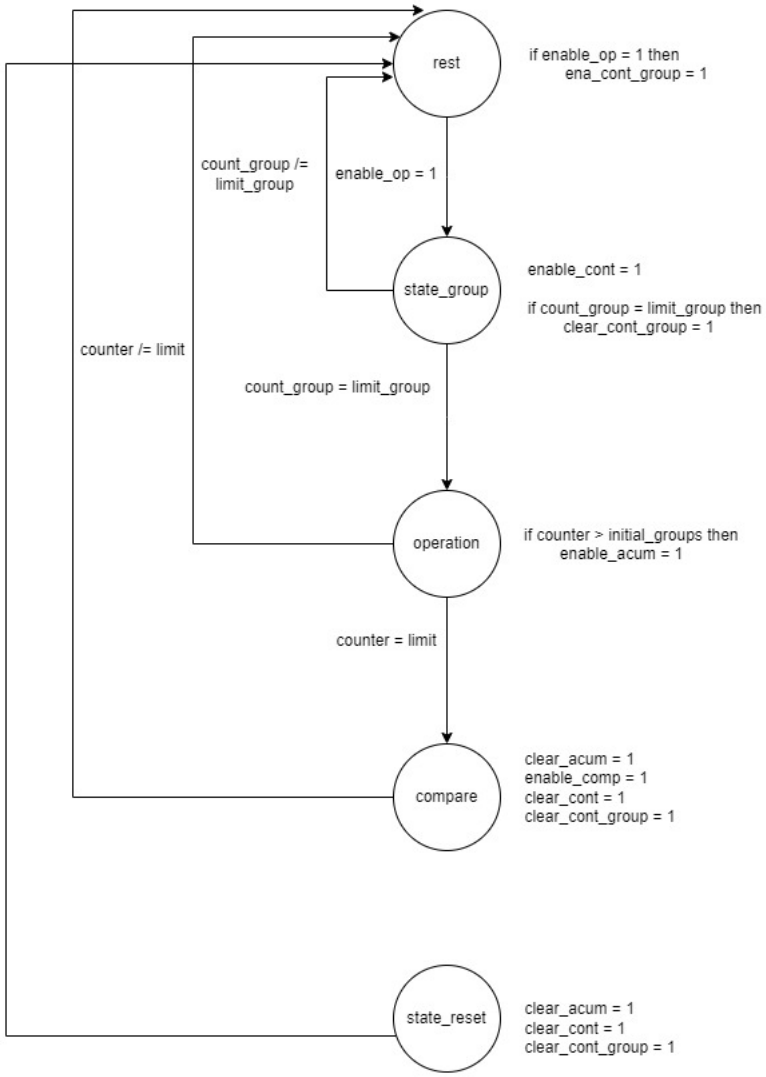


Figura 4.5: Máquina de estados de algoritmo NLA.

Finalmente, el algoritmo VRS se caracteriza por una lógica única, en la que fue necesario calcular primero la media de la señal para luego comparar cada muestra individualmente. Para lograrlo, se implementó una memoria RAM que almacena los datos recibidos, los cuales se emplean posteriormente en la operación. Esto requirió la adición de un estado específico para la escritura en la memoria RAM, **memory** (véase la Figura 4.6). En este estado, se habilita la escritura de la memoria, se habilita el acumulador que permite calcular la media y los contadores, volviendo a **rest**. Una vez culminada la escritura, se avanza al estado **operation** y se procede de la misma manera que en las máquinas de estado mencionadas anteriormente.

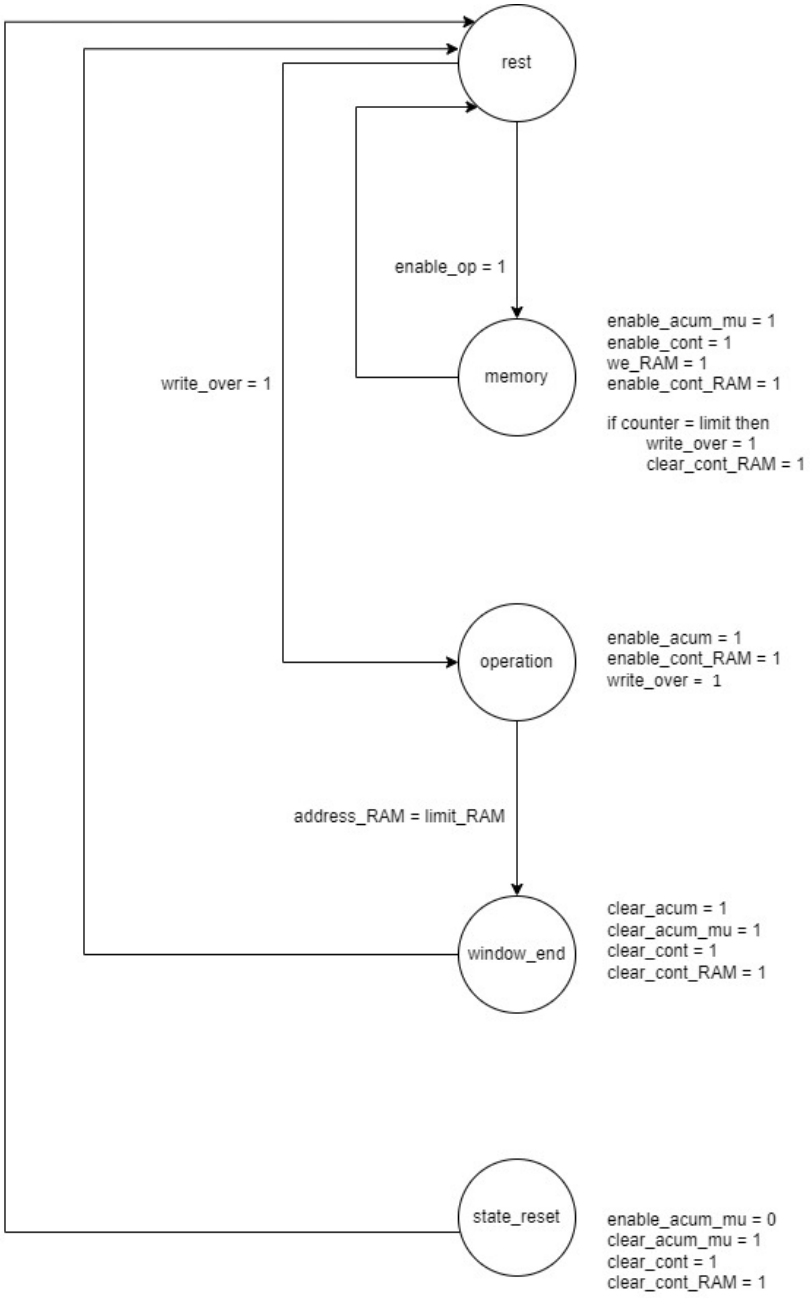


Figura 4.6: Máquina de estados de algoritmo VRS.

4.3. Sistema

Para la implementación del sistema que contiene todos los algoritmos, se consideraron dos enfoques principales. El primer enfoque consistió en aplicar un solo algoritmo a la vez, con el objetivo de optimizar el espacio del sistema mediante el uso de la menor cantidad de bloques posible. El segundo enfoque implicó el uso simultáneo de varios algoritmos, permitiendo la comparación de su eficiencia en la detección. Sin embargo, este enfoque requirió un mayor número de bloques, lo que limitó la optimización del espacio disponible.

4.3.1. Sistema 1: Aplicación de único algoritmo

El funcionamiento de los algoritmos para este sistema respeta el diagrama de bloques mostrado en la Figura 4.3, agregándole además el bloque de simulación de datos mostrado en la Figura 4.1 (Véase la Figura 4.10). El esquemático se encuentra disponible en el Apéndice H: Figura H.1.

El contador implementado se encarga de contar la cantidad de muestras que se van procesando a medida que llegan. La salida del mismo entra a los algoritmos de manera de compararla con el límite de ventana y lograr así realizar la comparación cuando corresponda. El esquemático del componente se muestra en la Figura 4.7.

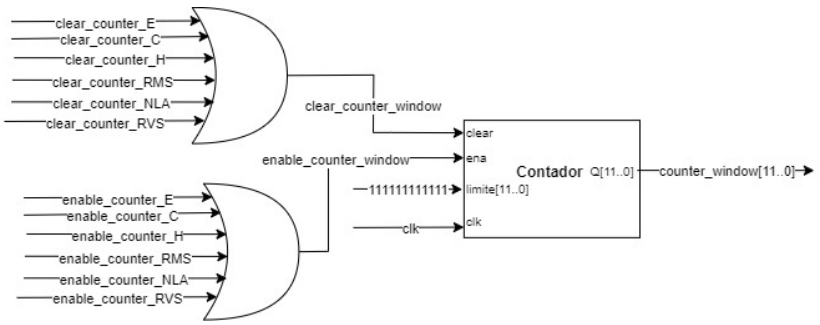


Figura 4.7: Esquemático del contador. El enable y el clear del contador se implementan con un OR entre los enables y clears que salen de los algoritmos. De esta manera, el contador se incrementa en uno cuando corresponde.

Por otro lado, se cuenta con dos acumuladores, uno que calcula la acumulación de los resultados de las operaciones y otro que calcula la media en caso que corresponda. Éstos se manejan mediante señales de habilitación de manera de optimizar el espacio ocupado por el sistema en la placa.

Previo a la compilación y sintetización del sistema, el usuario debe elegir el algoritmo que desea emplear, de manera de definir el umbral de detección (**threshold**) y las constantes **group_size_NLA** y **groups_NLA** para el caso en el cual se elija el algoritmo NLA. Esto se debe a que estas variables deben estar previamente

definidas en el código.

La selección del algoritmo se realiza externamente mediante interruptores en la placa, que generan la entrada **choice[2..0]** (Tabla 4.1). Esta entrada se utiliza para definir las salidas de ciertos bloques del sistema, principalmente los multiplexores que controlan las habilitaciones de los algoritmos, así como las señales a acumular y comparar.

choice[2..0]	**Algoritmo**
001	Energy
010	Coastline
011	Hjorth
100	RMS Amplitude
101	NLA
110	VRS

Tabla 4.1: Combinatoria definida para cada algoritmo.

La memoria ROM se carga con 16 ventanas de datos para su procesamiento. Una vez completado este procesamiento, el usuario puede actualizar el contenido de la ROM siguiendo el procedimiento detallado en la Subsección 4.1.3. Para notificar al sistema que está listo para un nuevo procesamiento, el usuario activa el Switch[8] en la placa, elevando su estado de 0 a 1 y luego regresándolo a 0. Este procedimiento garantiza que el sistema esté preparado adecuadamente para la nueva actualización de la memoria.

El sistema utiliza la salida **detection** para indicar si se ha detectado una crisis en una ventana de datos, lo cual se refleja mediante un LED en la placa. Para verificar el correcto funcionamiento, además de observar el encendido del LED cuando corresponde, se utilizó la herramienta SignalTap II Logic Analyzer del software Quartus. Esta herramienta permitió monitorear en tiempo real el valor de la salida del algoritmo, el acumulador y la entrada al comparador. Posteriormente, se compararon estos resultados con los obtenidos mediante simulaciones en Python, asegurándose de que fueran consistentes para validar el correcto desempeño del sistema.

En las Figuras 4.8 y 4.9 se muestran ejemplos de las pruebas realizadas para verificar el correcto funcionamiento del sistema.

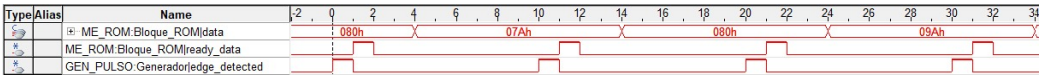


Figura 4.8: Imagen obtenida de la herramienta SignalTap Logic Analyzer de Quartus. Visualización del comportamiento del bloque ME-ROM (4.1) que simula la llegada de un dato.

En la Figura 4.8 se visualiza la prueba de la simulación de llegada de datos. La misma se realiza con el reloj del sistema trabajando a 50 MHz y el envío de datos a 5 MHz. Inmediatamente se verifica esto observando que la salida del generador de pulsos se activa cada 10 flancos de reloj, coherente con la relación entre las frecuencias mencionadas. Luego, la máquina de estados del bloque ROM detecta el flanco de subida y en el flanco siguiente habilita el procesamiento de los algoritmos a través de la señal **ready_data**. Luego, se espera un tiempo prudente, en este caso dos flancos de reloj, que permita a los algoritmos leer el dato correctamente, para cambiar la dirección de la ROM y preparar el siguiente dato para cuando se active nuevamente el generador de pulsos.

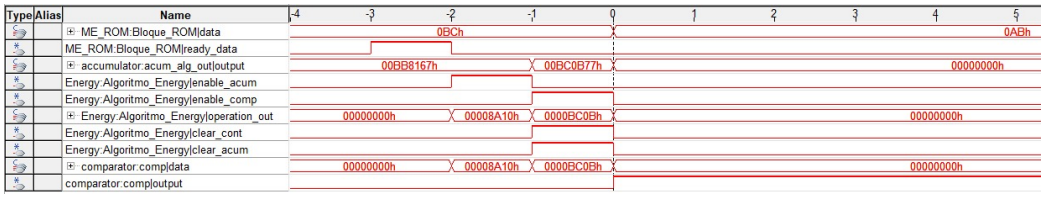


Figura 4.9: Imagen obtenida de la herramienta SignalTap Logic Analyzer de Quartus. Visualización del comportamiento del bloque del algoritmo Energy, trabajando en conjunto con los bloques acumulador, comparador y contador (4.3) cuando se llega al final de una ventana.

En el caso de la Figura 4.9, se observa el instante en el que finaliza el procesamiento de una ventana y se determina si hay crisis. La señal de **ready_data** proveniente del bloque ROM le indica al algoritmo, en este caso el Energy, que debe procesar un dato nuevo. Este realiza las operaciones pertinentes y le indica al acumulador (por medio de la señal **enable_acum**) que debe sumarlo. Dado que es el dato que completa la ventana, el algoritmo toma el valor acumulado hasta el momento, lo divide entre el tamaño de ventana (para obtener el promedio) y le indica al comparador que lo evalúe (a través de la señal **enable_comp**). Mientras tanto, se reinician los valores del acumulador y el contador de la ventana, en anticipación al próximo dato que llegue. Se verifica que el funcionamiento del algoritmo es correcto, dado que el valor que entra al comparador es el mismo que se obtiene al realizar los cálculos en Python. Esta prueba se llevó a cabo de forma análoga para todos los algoritmos, resultando de forma positiva en cada caso.

Una desventaja de esta aplicación es que, cada vez que se cambia la selección del algoritmo, es necesario actualizar el archivo de proyecto para ajustar el umbral correspondiente, lo que implica volver a programar la placa. Asimismo, la necesidad de modificar el archivo de proyecto y reprogramar la placa puede llevar a

tiempos de inactividad y aumentar la posibilidad de errores durante el proceso de configuración. Esto también puede resultar en un proceso menos flexible y más tedioso para el usuario final, especialmente si se requieren ajustes frecuentes. Asimismo, el hecho de que solo se pueda aplicar un algoritmo a la vez puede llevar a detecciones o no detecciones incorrectas, dependiendo de la eficacia del algoritmo elegido para la señal cargada, ya que no es posible comparar los resultados con los de otro algoritmo.

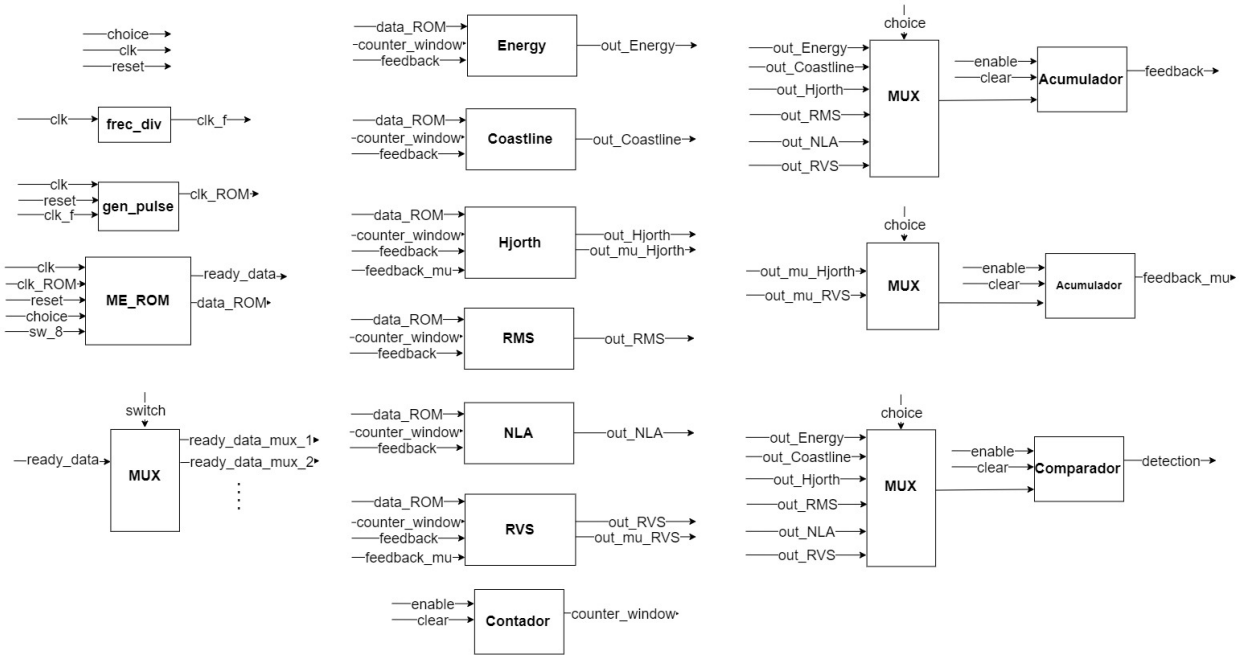


Figura 4.10: Diagrama de bloques simplificado del Sistema 1. Las señales de enable y clear se simplifican para una mejor comprensión del sistema. Las señales *ready_data_mux* corresponden a las señales de habilitación de cada algoritmo. Una vez les llega esa señal, el algoritmo procede a realizar la operación correspondiente y habilitar los acumuladores. Una vez terminada la ventana, el algoritmo habilita el comparador y a la salida se muestra si se detectó o no una crisis epiléptica.

4.3.2. Sistema 2: Aplicación simultánea de varios algoritmos

Para aplicar simultáneamente varios algoritmos, se debe modificar el sistema mencionado anteriormente. En este caso, se necesitan acumuladores independientes para cada algoritmo, así como comparadores; sin embargo, ya no es necesario el uso de multiplexores para las entradas de los acumuladores (el diagrama de bloques se puede observar en la Figura 4.11).

La elección del algoritmo es similar a la implementada anteriormente, con la excepción de que, en este caso, se incrementa la cantidad de bits a utilizar. De esta

manera, es posible seleccionar varios algoritmos simultáneamente (Ver Tabla 4.2)

switch[6..0]	**Algoritmo**
0000001	Energy
0000010	Coastline
0000100	Hjorth
0001000	RMS Amplitude
0010000	NLA
0100000	VRS
1000000	Other

Tabla 4.2: Combinatoria definida para cada algoritmo.

En la implementación anterior del sistema, se utilizó un bloque contador para calcular la cantidad de muestras en la ventana. Sin embargo, al utilizar la misma técnica para este sistema, resultó ineficaz ya que las señales de habilitación de cada algoritmo no se activaban simultáneamente. Como consecuencia, el contador se activaba en momentos incorrectos, lo que generaba errores en el conteo de muestras y afectaba el rendimiento de los algoritmos. Para corregir este problema, se implementaron contadores independientes para cada algoritmo.

Esta solución permite combinar algoritmos y aplicarlos simultáneamente a la misma señal. Consecuentemente, y en base a lo detallado en la Sección 3.9, este sistema posibilita el uso de operaciones lógicas como OR y AND, facilitando un análisis más robusto de los resultados obtenidos.

Una cuestión relevante fue considerar si es necesario mantener contadores independientes para cada algoritmo, o si sería más eficiente compartir un contador entre aquellos algoritmos cuyo tamaño de ventana óptimo es el mismo, reduciendo así el uso de bloques lógicos. Sin embargo, esta optimización podría dar lugar a errores similares a los mencionados, como activaciones desincronizadas de los contadores, lo que resultaría en un conteo incorrecto de las muestras.

La salida del sistema se ajustó para representar la detección de cada algoritmo mediante un aumento en la cantidad de bits. Cada bit se conectó a un LED en la placa, de modo que, ante una detección positiva, el LED correspondiente se encendía.

Esta implementación también facilita una configuración flexible y personalizada por parte del usuario. Con la idea de tratar cada componente (acumuladores, contadores, núcleos de algoritmos) como bloques, el usuario tiene la posibilidad de construir su propio sistema de detección. Esto significa que puede elegir aplicar uno, dos o varios algoritmos simultáneamente, decidiendo si comparten un mismo acumulador o utilizan acumuladores independientes. Esta flexibilidad permite

adaptar el sistema de detección a las necesidades específicas del usuario, haciendo posible una configuración rápida y sencilla según la señal que desee analizar y los algoritmos que considere más apropiados.

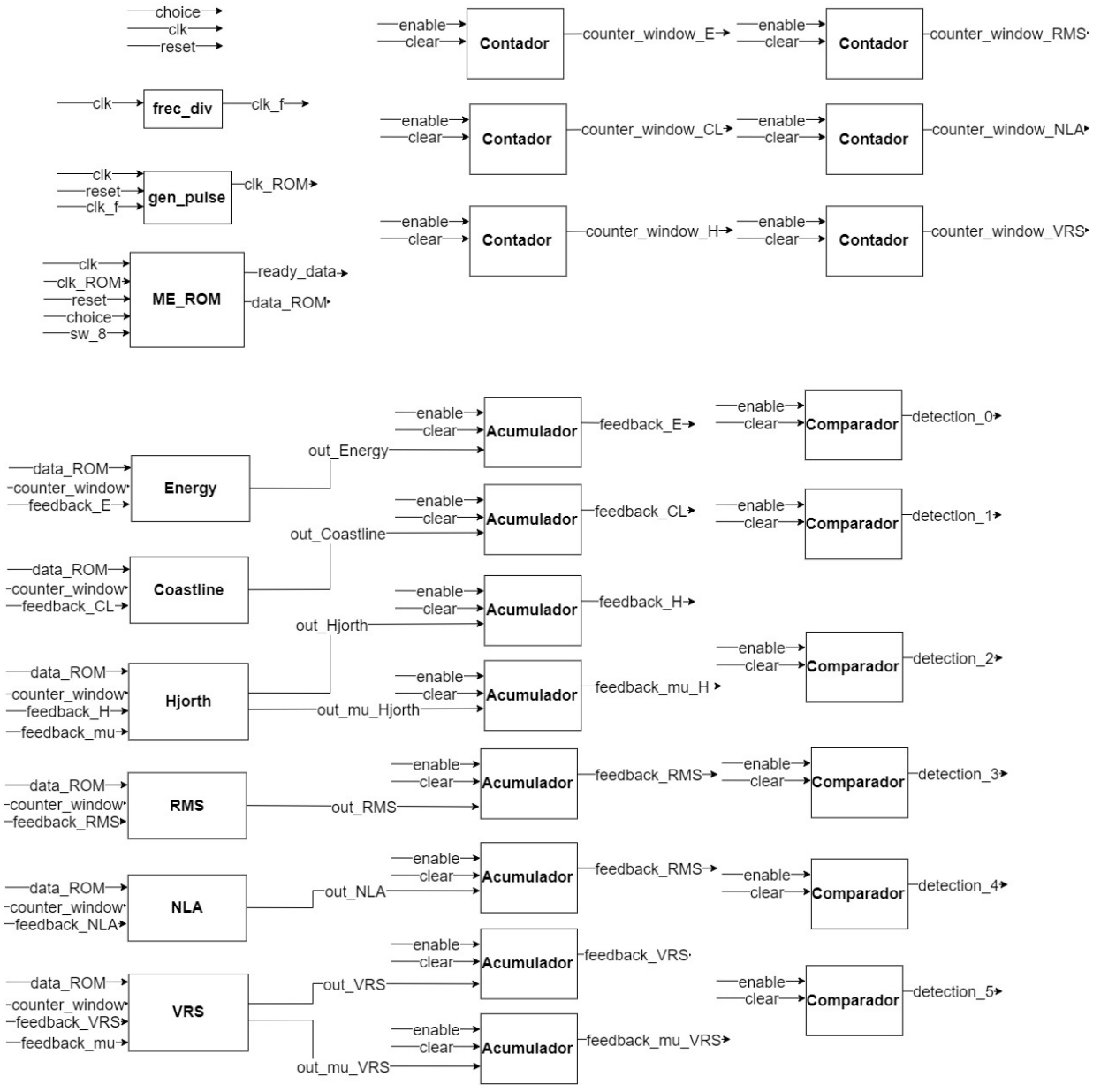


Figura 4.11: Diagrama de bloques simplificado del Sistema 2. Cada algoritmo cuenta con sus propios acumuladores, contador y comparador.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Medida de consumo

El consumo de potencia es un aspecto crucial a tener en cuenta durante el diseño de sistemas digitales. Los FPGAs tienen un consumo de potencia compuesto por dos componentes principales: consumo estático y consumo dinámico.

El consumo estático ocurre incluso cuando el dispositivo no está ejecutando ninguna operación activa. Este tipo de consumo está relacionado con fugas de corriente en los transistores del chip y con el mantenimiento del estado de los elementos lógicos programados.

Por otro lado, el consumo dinámico está asociado con las operaciones que realiza el FPGA. Este consumo aumenta a medida que las señales en los registros, puertas lógicas y otros elementos del FPGA cambian de estado, lo que genera un consumo de energía adicional relacionado con la conmutación de estos componentes.

Comprender y controlar el consumo de potencia en un FPGA es crucial para optimizar el rendimiento y la eficiencia de cualquier sistema implementado. A lo largo de este trabajo, se realizaron diversas mediciones y técnicas para caracterizar el consumo del diseño. Estos experimentos permitieron obtener un panorama detallado del comportamiento de consumo, separando el componente dinámico del estático. El consumo dinámico es el que tiene mayor relevancia al comparar los algoritmos, ya que está directamente vinculado a su funcionamiento, mientras que el consumo estático está asociado a las características propias de la FPGA. Esto proporciona un enfoque más preciso para analizar el impacto de los distintos algoritmos sobre el consumo total del sistema.

5.1. Metodología

Para medir el consumo de potencia del FPGA, programado con los sistemas mencionados anteriormente, fue necesario modificar la placa DE0, ya que esta no está adaptada para realizar mediciones de consumo directamente. La adaptación

utilizada se basó en una metodología tomada de la Tesis de Doctorado mencionada en [18], que consistió en retirar el regulador que alimenta el núcleo del chip y reemplazarlo por un regulador externo. Además, se integró una resistencia *shunt* para permitir la medición utilizando un multímetro Fluke 45. Un esquema demostrando la modificación se muestra en la Figura 5.1.

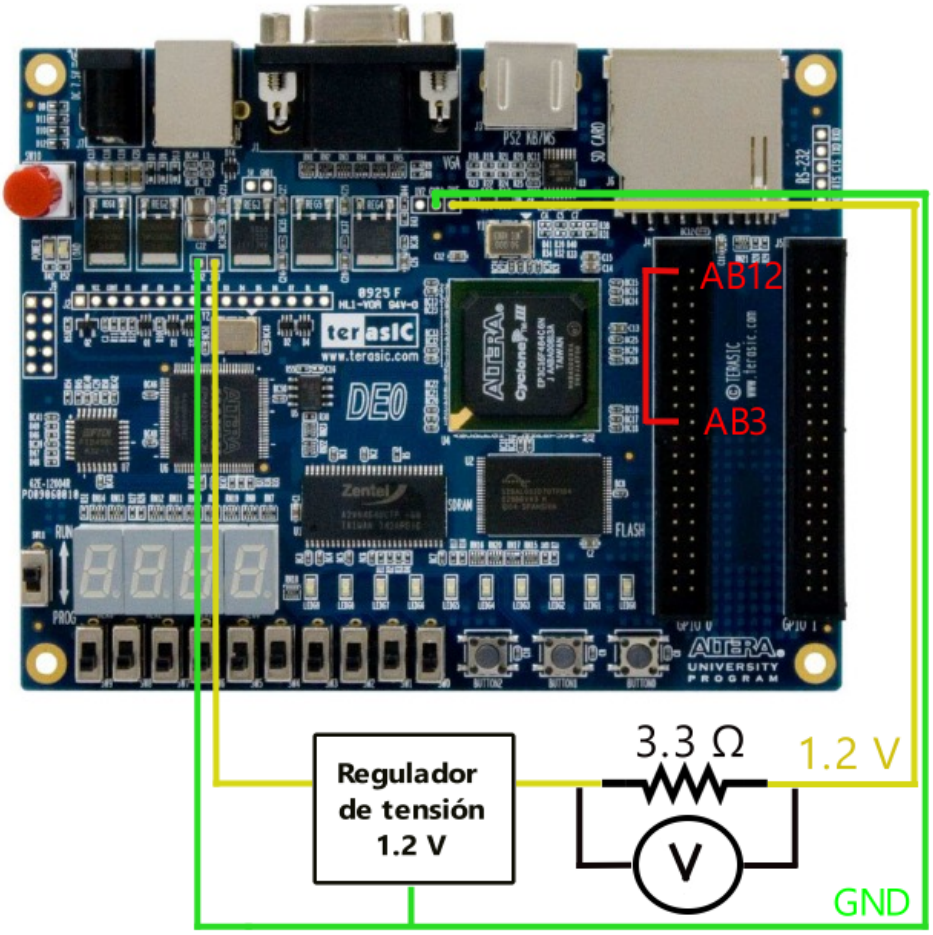


Figura 5.1: Diagrama detallando la modificación realizada a la placa DE0 para realizar medidas de consumo. Imagen de placa DE0 obtenida de [19]. La conexión externa de los pines AB12 y AB3 corresponde a una entrada y una salida de reloj, respectivamente, empleada para llevar a cabo mediciones del consumo estático del diseño.

Para visualizar en el multímetro la caída de voltaje a través de la resistencia *shunt*, es fundamental tener en cuenta varios factores. En primer lugar, la velocidad de transferencia de datos desde la memoria ROM al sistema debe ser adecuada. Si la frecuencia de envío de datos es elevada, los algoritmos no tendrán tiempo suficiente para procesarlos. Por otro lado, cuando la frecuencia es baja, el intervalo

de inactividad entre el procesamiento de muestras consecutivas puede ser prolongado. Esto provoca que el consumo observado se asemeje más al de un estado de reposo que al de un estado activo de procesamiento. En segundo lugar, el valor de la resistencia *shunt* debe ajustarse al rango de consumo de corriente esperado. Para corrientes muy bajas, es necesario aumentar su valor, mientras que para corrientes elevadas, se debe reducir para mantener la caída de tensión dentro de los límites tolerables del FPGA, los cuales varían en el rango de 1.15 V a 1.25 V [20]. Teniendo en cuenta estos factores, se decidió utilizar una resistencia de 3,3 Ω y una frecuencia de envío de datos de 5 MHz.

Es importante destacar que el consumo medido no incluye el de los pines de entrada y salida, y que siempre habrá un consumo estático del chip. De hecho, el consumo dinámico del sistema es significativamente menor en comparación con el estático. Para medir el consumo estático, se agregó al sistema una entrada y una salida de reloj, que se conectaba externamente mediante un cable a través de los pines AB12 y AB3 de la placa (Véase la Figura 5.1). De esta manera, para medir el consumo estático simplemente se desconectaba el cable, dejando al circuito sin fuente de reloj. Para medir el consumo total, se conectaba nuevamente el cable para restaurar el funcionamiento completo del sistema.

Una técnica eficaz para mejorar la precisión de la medición del consumo y aumentar su visibilidad en el multímetro consiste en instanciar el algoritmo múltiples veces, incrementando así la potencia consumida. Para este fin, se generaron archivos que contenían un generador de pulsos, un divisor de frecuencia, un bloque ROM, un contador, el bloque del algoritmo correspondiente, uno o dos sumadores dependiendo del algoritmo y un comparador (Véase la Figura 5.2). El bloque del algoritmo, sus sumadores y el comparador fueron instanciados 100 veces, en pasos de 10 y se realizaron mediciones tanto con el reloj desactivado como en funcionamiento (Véase Figura 5.3). Además, se deshabilitó la herramienta de Quartus *Remove duplicated registers*, para evitar que el sintetizador eliminara registros redundantes y asegurar que cada instancia del algoritmo se mantuviera. Esto permitió realizar las mediciones de forma precisa.

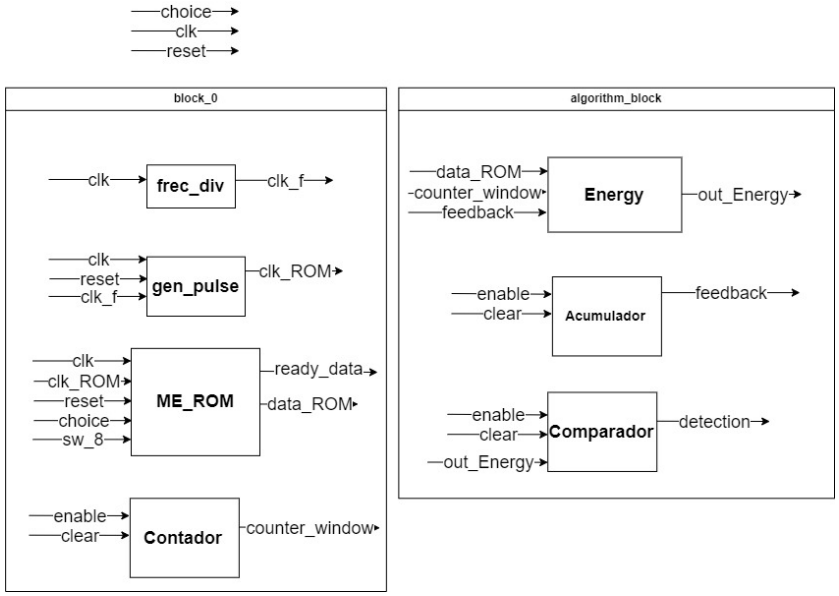


Figura 5.2: Diagrama de bloques simplificado del diseño utilizado para medir el consumo. En el caso de utilizar otro algoritmo, el bloque Energy se sustituye por el algoritmo deseado.

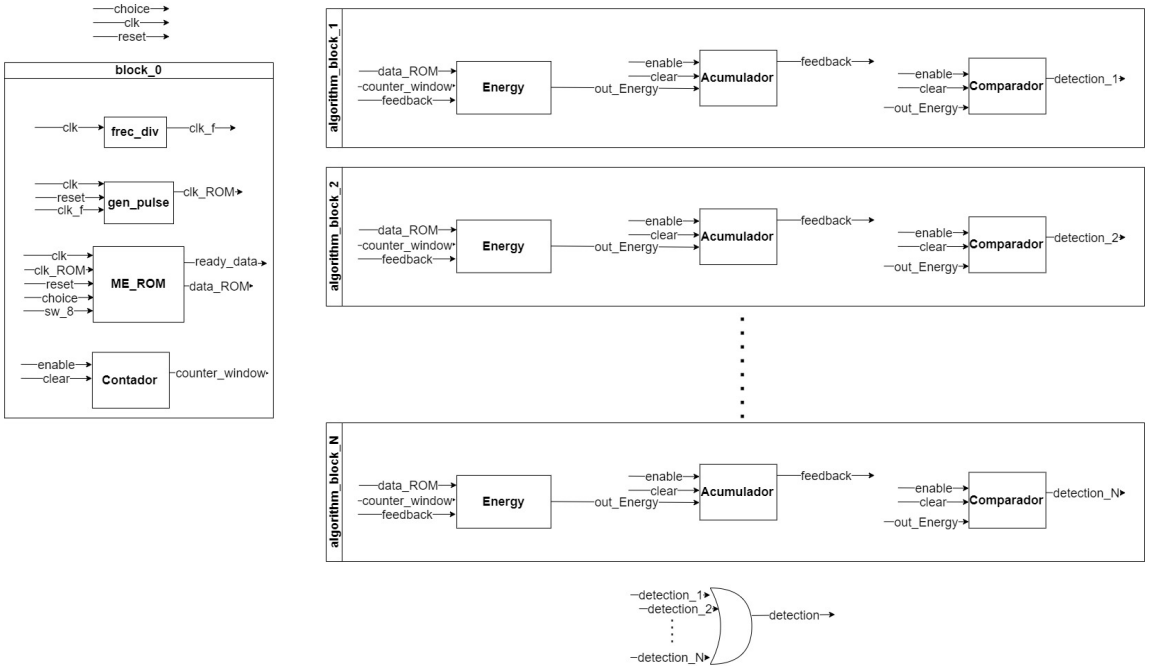


Figura 5.3: Diagrama de bloques simplificado del diseño utilizado para medir el consumo de N instancias del Energy. En el caso de utilizar otro algoritmo, el bloque Energy se sustituye por el algoritmo deseado. La salida de este diseño se prende si las instancias tienen una detección positiva.

5.2. Resultados

5.2.1. Consumo de algoritmos individualmente

En una primera instancia, se midió el consumo de cada algoritmo individualmente. Para ello, se realizaron varias instancias del algoritmo, como se mencionó en la Sección 5.1, utilizando una ventana de 256 datos que contenía eventos de crisis, lo cual es representativo de los datos procesados por los algoritmos en situaciones de detección. Se definieron umbrales bajos para asegurar que cada algoritmo realizara detecciones, garantizando así un comportamiento constante y comparable, y se midió tanto el consumo estático como el total de cada instancia.

Las medidas se encuentran detalladas en el Apéndice I. Con base en estas, se elaboró un gráfico que ilustra la dependencia del consumo total en función de la cantidad de instancias. Este gráfico evidencia un crecimiento en el consumo a medida que se incrementa el número de instancias del algoritmo correspondiente (véase la Figura 5.4).

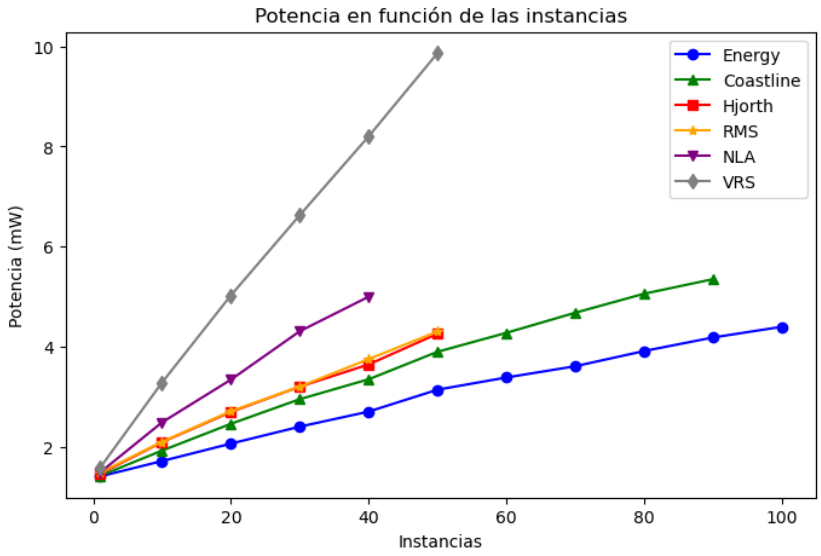


Figura 5.4: Consumo total de algoritmos en función de la cantidad de instancias.Algunos algoritmos presentan un menor número de instancias, ya que al aumentar la cantidad de elementos lógicos utilizados, la demanda de recursos del FPGA incrementa, alcanzando así el límite de capacidad disponible.

El consumo total del diseño mostrado en la Figura 5.2 se puede representar de la siguiente manera:

$$P_{design}^{total}(1) = P^{static}(1) + P_{block\_0}^{dynam} + P_{algorithm\_block}^{dynam} \tag{5.1}$$

Al realizar N instancias del bloque *algorithm_block*, la ecuación anterior se transforma en:

$$P^{total}_{design}(N) = P^{static}(N) + P^{dynam}_{block\_0} + N * P^{dynam}_{algorithm\_block} \quad (5.2)$$

Una observación a destacar es que el consumo estático puede presentar una ligera variación al medirlo en las diferentes instancias, por ello se considera diferente para cada una. Restando ambas ecuaciones se obtiene:

$$(N-1) * P^{dynam}_{algoritm\_block} = P^{total}_{design}(N) - P^{total}_{design}(1) - (P^{static}(N) - P^{static}(1)) \quad (5.3)$$

Despejando el consumo dinámico del bloque se obtiene:

$$P^{dynam}_{algoritm\_block} = \frac{P^{total}_{design}(N) - P^{total}_{design}(1) - (P^{static}(N) - P^{static}(1))}{N-1} \quad (5.4)$$

Aplicando esta ecuación se obtiene el consumo dinámico del algoritmo para una instancia en función de todas las medidas previas (Véase la Figura 5.5).

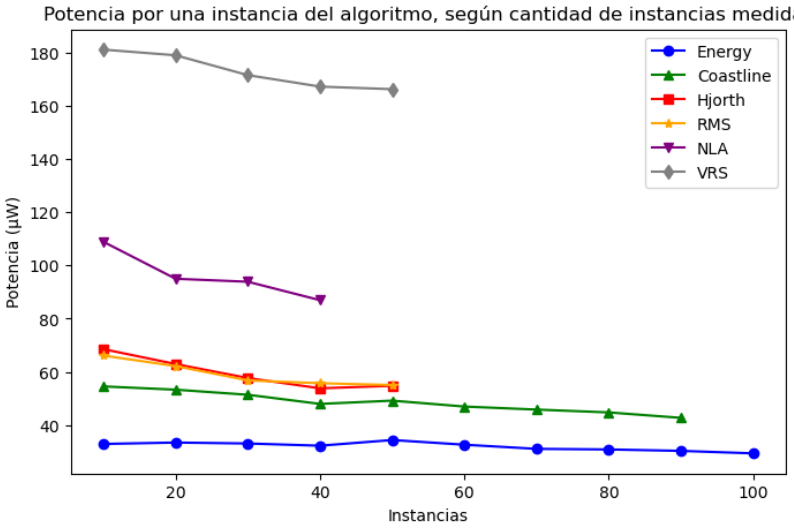


Figura 5.5: Consumo dinámico de algoritmos en función de la cantidad de instancias.

Se puede observar que los valores obtenidos por instancia mantienen una relación bastante constante uno de los otros. Esto se puede verificar en el Apéndice I. Finalmente, se calcula un promedio de estos valores para determinar el consumo dinámico promedio del algoritmo correspondiente, como se muestra en la Tabla 5.1. Además, se realizaron estimaciones de consumo con la herramienta *PowerPlay Power Analyzer* de Quartus, con el objetivo de evaluar si estas pueden ser útiles para un estudio preliminar del diseño.

La tasa de conmutación, o *toggle rate*, de un diseño digital representa la frecuencia con la que las señales cambian de estado (de 0 a 1 o de 1 a 0) durante

su funcionamiento. En Quartus, la estimación precisa del consumo de energía depende en gran medida de esta tasa de conmutación, ya que el consumo dinámico está directamente relacionado con la actividad de las señales. Al realizar una simulación del diseño y generar un archivo VCD, se capturan los cambios de estado de las diferentes señales a lo largo del tiempo. Este archivo se puede utilizar en la herramienta *PowerPlay Power Analyzer* para proporcionar una estimación más precisa del consumo de potencia, ya que refleja el comportamiento real del sistema bajo las condiciones de simulación.

Para la estimación, se utilizó el archivo VCD obtenido de la simulación. Dado que *PowerPlay Power Analyzer* requiere que se defina un *toggle rate* para las señales de entrada/salida (I/O) y para las señales restantes, se asignó un 10 % de *toggle* para las I/O. Para las señales restantes, se optó por un enfoque *vectorless*, lo que permitió obtener estimaciones más precisas en función de la actividad de estas señales.

Algoritmo	Consumo dinámico promedio (μW)	Estimación de Quartus (μW)
Energy	32.00	190
Coastline	48.50	220
Hjorth	59.53	250
RMS	59.16	230
NLA	96.17	540
VRS	173.64	810

Tabla 5.1: Medidas vs estimaciones de consumo dinámico de bloque *algorithm_block*.

De manera de ilustrar los valores de la Tabla 5.1 se presenta la Figura 5.6.

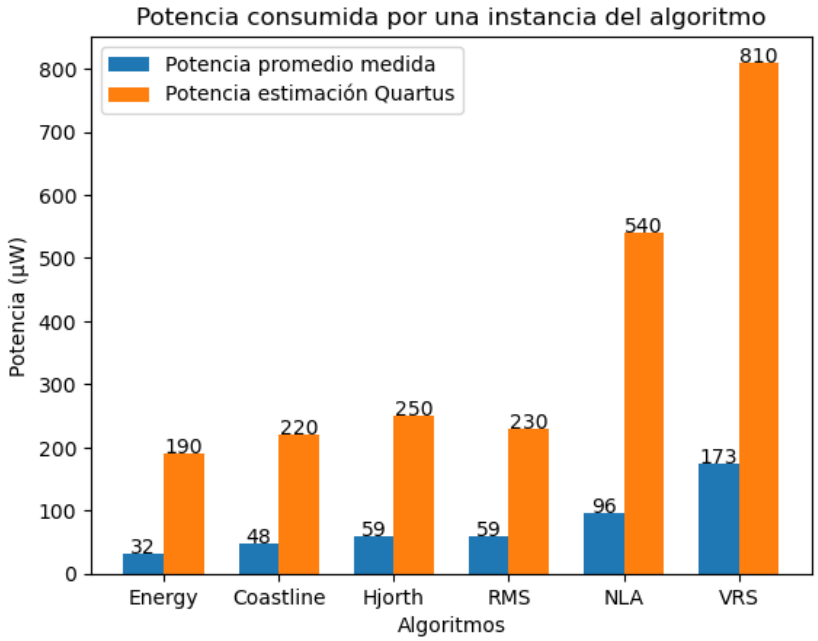


Figura 5.6: Consumo dinámico medido vs estimado bloque *algorithm_block*.

Se observa que las estimaciones de consumo dinámico obtenidas a través de la herramienta *PowerPlay Power Analyzer* de Quartus presentan discrepancias significativas en comparación con los valores promedio calculados. Esto sugiere que la herramienta no es del todo confiable para ofrecer una visión precisa del consumo del diseño. Las diferencias notables en las estimaciones, como se evidencia en la Figura 5.6, indican que estas estimaciones no deben ser consideradas como la única fuente de información para entender el consumo real del sistema.

5.2.2. Consumo de sistemas

El siguiente paso consistió en medir el consumo de los sistemas descritos en la Sección 4.3. Para el caso del Sistema 1 se realizaron medidas variando los bits de la representación de la señal de entrada de manera de observar si la variación afecta el consumo. Todas las medidas se realizaron bajo las mismas condiciones mencionadas previamente.

Algoritmo aplicado	Consumo total (mW)	Consumo dinámico (mW)
Energy	1.432	0.448
Coastline	1.428	0.445
Hjorth	1.435	0.452
RMS	1.432	0.448
NLA	1.424	0.441
VRS	1.468	0.485

Tabla 5.2: Medidas de consumo Sistema 1 para bits_in = 8. La cantidad de elementos lógicos ocupado por el diseño es de 942.

Algoritmo aplicado	Consumo total (mW)	Consumo dinámico (mW)
Energy	1.518	0.524
Coastline	1.507	0.513
Hjorth	1.529	0.535
RMS	1.522	0.528
NLA	1.504	0.510
VRS	1.583	0.589

Tabla 5.3: Medidas de consumo Sistema 1 para bits_in = 10. La cantidad de elementos lógicos ocupado por el diseño es de 1201.

Algoritmo aplicado	Consumo total (mW)	Consumo dinámico (mW)
Energy	1.655	0.661
Coastline	1.637	0.643
Hjorth	1.681	0.687
RMS	1.659	0.665
NLA	1.634	0.640
VRS	1.760	0.766

Tabla 5.4: Medidas de consumo Sistema 1 para bits_in = 12. La cantidad de elementos lógicos ocupado por el diseño es de 1487.

Algoritmo aplicado	Consumo total (mW)	Consumo dinámico (mW)
Energy	1.731	0.737
Coastline	1.710	0.716
Hjorth	1.764	0.770
RMS	1.735	0.741
NLA	1.710	0.716
VRS	1.879	0.885

Tabla 5.5: Medidas de consumo Sistema 1 para bits_in = 14. La cantidad de elementos lógicos ocupado por el diseño es de 1811.

Se puede observar fácilmente que el consumo aumentó a medida que se aumenta la cantidad de bits de los datos a procesar. Esto puede deberse a que a medida que se incrementa el número de bits, se necesitan más elementos lógicos para manejar las operaciones relacionadas con esos datos, como sumas, multiplicaciones, y comparaciones. Esto implica el uso de más *lookup tables* (LUTs), registros, y otros componentes del FPGA, lo cual incrementa el consumo de energía. Esto implica un aumento en el consumo tanto por el almacenamiento en registros más grandes como por la transmisión de datos a través de buses internos del FPGA. Por otro lado, como el sistema cuenta con una memoria ROM interna, procesar datos de mayor tamaño puede requerir acceder a más celdas de memoria simultáneamente y por consiguiente aumentar el consumo.

Por otro lado, se puede observar que el consumo del Sistema 1 al aplicar un algoritmo a la vez es notablemente mayor que el consumo del bloque *algorithm_block* (Tabla 5.1). Esto puede deberse a que el sistema, además de implementar los algoritmos, incluye otros componentes como la memoria ROM, multiplexores y contadores, que son necesarios para gestionar el flujo de datos y controlar la ejecución. Estos elementos adicionales introducen un consumo extra, ya que las operaciones de lectura, escritura y conmutación de datos, así como las transiciones de los contadores, contribuyen al consumo total del sistema. Por lo tanto, es importante tener en cuenta no solo las mediciones de consumo de los algoritmos de forma aislada, sino también el impacto del consumo de la estructura de control y manejo de datos en el sistema completo. Además, aunque cada algoritmo cuenta con su propia señal de habilitación, los bloques de multiplicadores, raíz cuadrada y comparadores siguen operando de forma continua, independientemente de si el algoritmo está habilitado o no. Esto puede contribuir a un aumento en el consumo total del sistema.

Posteriormente, se midió el consumo del Sistema 2 al aplicar algoritmos tanto individualmente como simultáneamente, obteniendose los resultados que se muestran en la Tabla 5.6 a continuación.

Algoritmos aplicados	Consumo total (mW)	Consumo dinámico (mW)
Energy	1.634	0.643
Coastline	1.641	0.651
Hjorth	1.648	0.658
RMS	1.655	0.665
NLA	1.645	0.654
VRS	1.717	0.726
E + C	1.663	0.672
E + C + H	1.695	0.705
E + C + H + RMS	1.728	0.737
E + C + H + RMS + NLA	1.742	0.752
Todos	1.839	0.849

Tabla 5.6: Medidas de consumo Sistema 2 para bits_in = 12. La cantidad de elementos lógicos ocupado por el diseño es de 1598.

En este caso, se observa que el consumo aumenta a medida que se incrementa la cantidad de algoritmos aplicados simultáneamente. Estos resultados son coherentes con el comportamiento esperado. Sin embargo, un aspecto interesante a destacar es que, dado que este sistema cuenta con una mayor cantidad de sumadores, contadores y comparadores, cabría esperar que el consumo medido por cada algoritmo fuera superior al registrado en el Sistema 1. Sorprendentemente, esto no es así. Esto puede deberse a que los FPGAs están diseñados para ser eficientes en cuanto a su uso de recursos. Es posible que la lógica adicional introducida por los nuevos bloques de sumadores y comparadores se haya optimizado internamente por el propio FPGA, minimizando el impacto en el consumo.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Comparación de algoritmos

En este capítulo, se llevará a cabo una comparación exhaustiva de los algoritmos analizados, enfocándose en tres aspectos fundamentales: la eficiencia en la detección de crisis epilépticas, la cantidad de elementos lógicos utilizados y el consumo de potencia. Esta evaluación permitirá determinar no solo cuál de los algoritmos ofrece el mejor rendimiento en términos de detección, sino también su viabilidad en entornos de hardware restringido, considerando los recursos que requieren y su impacto energético.

6.1. Eficiencia en detección

Para comparar la eficiencia en la detección de los diferentes algoritmos, se utilizaron los datos y resultados obtenidos de la implementación en Python.

Con el objetivo de simular condiciones reales, el análisis se llevó a cabo replicando las mismas condiciones de la implementación en VHDL. Esto incluye el uso de señales truncadas, ventanas independientes y la elección de las señales para garantizar compatibilidad con todos los algoritmos.

Después de aplicar los algoritmos a las señales de las listas B.1 y B.2, y calcular la AUROC correspondiente para cada caso, se obtuvo el promedio de estos valores, representado en la Figura 6.1.

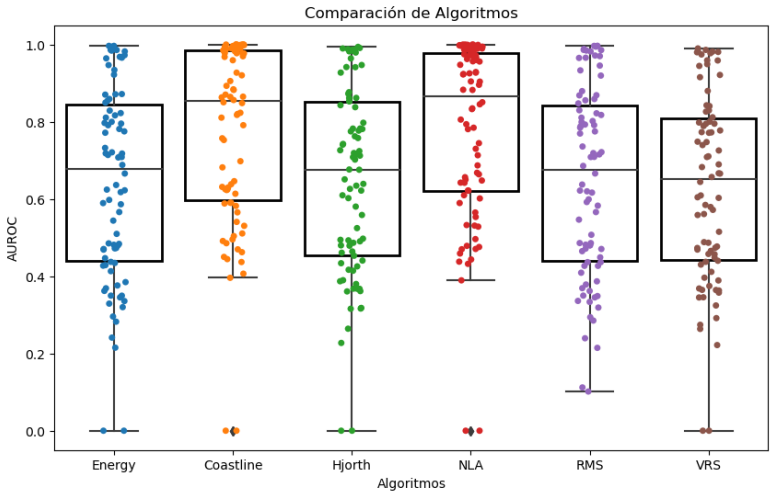


Figura 6.1: Comparativa de algoritmos para una ventana de 256 muestras y 12 bits de representación de los datos.

Se observa que los algoritmos Energy, Hjorth y RMS presentan una AUROC similar, alrededor del 60 %. Por otro lado, el algoritmo VRS muestra una AUROC ligeramente inferior a la de los algoritmos mencionados, lo que lo coloca en una posición desfavorable en términos de eficiencia para la detección de crisis epilépticas.

En contraste, los algoritmos Coastline y NLA destacan al alcanzar valores cercanos al 90 %. Esto sugiere que, en términos de eficiencia para la detección de crisis epilépticas, ambos algoritmos superan a Energy, Hjorth, RMS y VRS. Asimismo, se observa que, en general, al aumentar el tamaño de la ventana, la capacidad de detección de todos los algoritmos mejora.

En el caso específico del algoritmo Hjorth, se nota una ligera mejora en la detección al reducir el tamaño de la ventana a 64 muestras (3.26). Esto permite disminuir el tiempo de reacción y aumentar la eficacia simultáneamente en la implementación de este algoritmo. No obstante, dicho incremento en la AUROC no es suficiente para igualar la capacidad de detección del Coastline (3.26) o el NLA (3.27) para tiempos de reacción similares, por lo que estos permanecen como las mejores alternativas.

Con la intención de determinar si se lograba aumentar la eficacia de los algoritmos al mismo tiempo que se disminuye el tiempo de reacción, se implementó una configuración de ventana móvil. Con la misma, se determinó que no se lograba un incremento en la AUROC independientemente de la cantidad de muestras que se superpusieran entre ventanas consecutivas. Por el contrario, generalmente no se presentaron cambios. Resaltar esto es suficiente para llevar a cabo la comparación de los algoritmos individualmente, pero puede haber otras implicancias relaciona-

das a la ventana móvil que se exploran mejor en el Capítulo 7.

Por otro lado, el análisis de la variación en la representación de bits de entrada revela que, en general, para los algoritmos Energy, Hjorth, RMS y VRS, a partir de 5 o menos bits, la capacidad de detección disminuye considerablemente. Sin embargo, para los algoritmos Coastline y NLA, la detección se mantiene robusta incluso con esta representación de menor precisión. Esta característica es favorable, ya que permite la posibilidad de implementar menos bits en el diseño, lo que podría contribuir a mejorar otros aspectos, como el consumo.

6.2. Elementos lógicos utilizados

Una vez implementados los algoritmos, se registró la cantidad de elementos lógicos utilizados por cada diseño. Los elementos lógicos presentados en la Figura 6.2 corresponden al bloque *algorithm_block* variando el algoritmo utilizado.

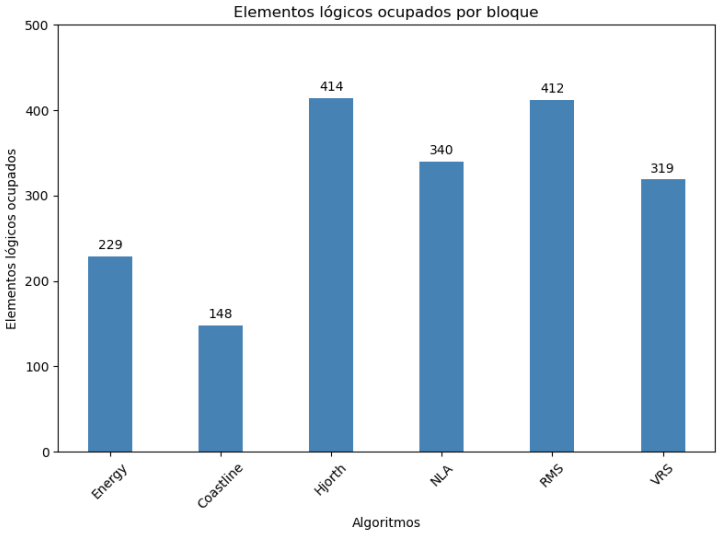


Figura 6.2: Elementos lógicos utilizados por cada algoritmo.

Entre los algoritmos evaluados, el Coastline es el que utiliza la menor cantidad de elementos lógicos, con solo 148, lo que lo hace el más eficiente en términos de uso de recursos. Le sigue el Energy, con 229 elementos lógicos, que también muestra un uso moderado de recursos y destaca por su simplicidad.

Por otro lado, los algoritmos Hjorth y RMS presentan un mayor uso de elementos lógicos, alcanzando los 414 y 412, respectivamente. Esta diferencia se debe a la mayor complejidad matemática de estos algoritmos, lo cual impacta directamente en la cantidad de operaciones requeridas para su implementación. El NLA y el VRS

se encuentran en una posición intermedia, utilizando 340 y 319 elementos lógicos, respectivamente, lo cual refleja su equilibrio entre complejidad de procesamiento y uso de recursos del FPGA.

Dado que al implementar los diseños, no solo es importante la cantidad de elementos lógicos ocupados por el bloque del algoritmo, sino también los ocupados por el diseño completo descrito en la Sección 5.1, en la Figura 6.3 se presentan los elementos lógicos ocupados por el mismo.

Se puede observar que si bien la cantidad de elementos lógicos aumenta en todos los casos, el Coastline persiste como el algoritmo que utiliza la menor cantidad de elementos lógicos, con un total de 179. En una línea similar, el algoritmo Energy le sigue, utilizando 258. Por otro lado, NLA y VRS se encuentran en un rango medio de ocupación, utlizando 369 y 348 respectivamente. Finalmente, Hjorth y RMS son los algoritmos que requieren más recursos, con 444 y 443 elementos lógicos cada uno.

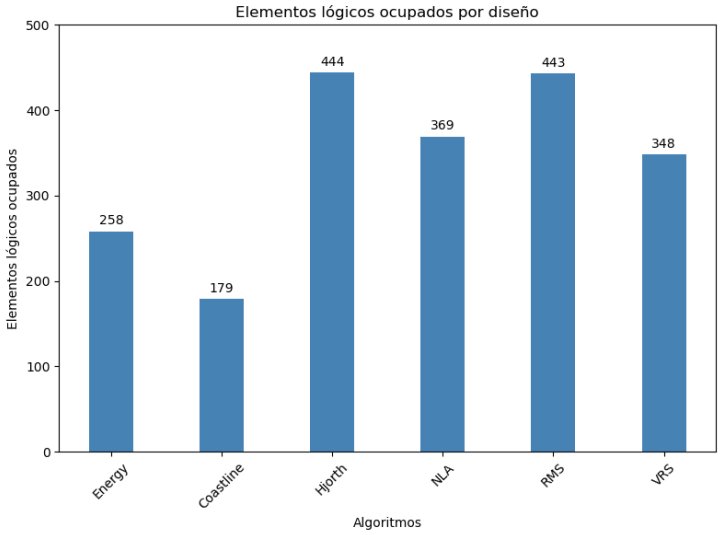


Figura 6.3: Elementos lógicos utilizados por el diseño implementado para las medidas de consumo mostrado en Figura 5.2.

En conclusión, la comparación muestra que los algoritmos más simples como Coastline y Energy son más eficientes en términos de ocupación de recursos, lo que los hace ideales para implementaciones en sistemas con limitaciones de espacio y potencia. En contraste, los algoritmos Hjorth y RMS consumen más recursos, lo que debe considerarse al momento de elegir el algoritmo adecuado para aplicaciones específicas.

6.3. Consumo

Dados los resultados presentados en la Figura 5.6, a continuación se realiza una comparación de los algoritmos.

El Energy realiza una operación de promedio de los cuadrados de los valores, lo cual implica multiplicaciones y sumas. Sin embargo, no necesita manejar grandes cantidades de memoria intermedia ni realizar cálculos iterativos complejos, lo que resulta en un consumo relativamente bajo.

Por otro lado, aunque el cálculo básico del Coastline implica una simple resta entre valores consecutivos, el uso de un registro para almacenar el valor previo añade un elemento de almacenamiento que debe actualizarse en cada ciclo de reloj, lo que aumenta el consumo en comparación con el Energy. Adicionalmente, las operaciones de resta y la necesidad de leer y escribir en el registro generan un incremento en el consumo.

El cálculo del parámetro de Hjorth implica primero determinar la media de la ventana de datos y luego realizar una resta con el cuadrado de cada muestra. Esto involucra múltiples sumas y multiplicaciones, además de cálculos de promedio. La complejidad matemática es mayor que la del Coastline y del Energy, lo que se traduce en un consumo superior debido al procesamiento adicional requerido.

El algoritmo de RMS calcula la raíz cuadrada del valor obtenido por el algoritmo de Energy. Aunque conceptualmente puede parecer sencillo, la operación de raíz cuadrada es costosa en términos de recursos de hardware, especialmente en un FPGA, donde puede requerir varias etapas de cálculo. Esto explica por qué su consumo es ligeramente inferior al Hjorth y superior al Energy.

El NLA involucra la búsqueda de máximos y mínimos en varias ventanas de datos, lo que requiere comparadores adicionales y una mayor complejidad lógica para determinar los valores de HV y LV. Luego, estos valores se restan y acumulan. La cantidad de comparaciones y operaciones de búsqueda de mínimos y máximos requiere una mayor cantidad de elementos de hardware, lo que resulta en un consumo significativamente mayor que los algoritmos anteriores.

En el caso del VRS, el alto consumo puede explicarse por la necesidad de utilizar una memoria RAM para almacenar temporalmente los datos mientras se calcula μ, que es la media de los valores absolutos de las muestras. Dado que el cálculo de la varianza rectificada requiere conocer previamente μ, primero es necesario guardar los datos en la RAM, calcular la media, y luego realizar la operación. Este proceso implica un uso intensivo de recursos de memoria, ya que la RAM debe escribir y leer datos de forma continua, lo que incrementa el consumo. Además, la lógica asociada a la gestión de la memoria, incluyendo los controladores que manejan las transferencias de datos, también contribuye al mayor uso de potencia. Por

lo tanto, esta combinación de almacenamiento temporal en RAM y procesamiento de los datos es un factor clave que explica el consumo significativamente más alto del algoritmo VRS en comparación con otros métodos.

Al analizar los resultados de las estimaciones de consumo proporcionadas por la herramienta *PowerPlay Power Analyzer*, se observa que estas son significativamente mayores que las mediciones experimentales obtenidas. Incluso utilizando el archivo VCD, que a partir de una simulación determina la tasa de cambio en los nodos del sistema, las estimaciones presentadas son al menos 5 veces mayores que las medidas. Esto da lugar a concluir que esta herramienta no es adecuada para realizar una estimación de consumo en ninguna instancia del diseño.

6.4. Conclusiones del capítulo

Con el propósito de comparar los algoritmos en eficiencia de detección, consumo y área ocupada, se presenta la Figura 6.4. En ella se utilizan los datos promedio de eficiencia en detección presentados en la Figura 6.1, los resultados de consumo medido mostrados en la Tabla 5.1 y los elementos lógicos utilizados expuestos en la Figura 6.2. Estos datos constituyen un punto de comparación representativo del rendimiento de los algoritmos.

Adicionalmente, se determinó una figura de mérito con el propósito de ponderar el consumo y el área simultáneamente, facilitando así una evaluación equilibrada de los resultados:

$$\text{Figura de mérito} = \text{Consumo} \times \text{Área} \tag{6.1}$$

Siendo Área la cantidad de elementos lógicos utilizados.

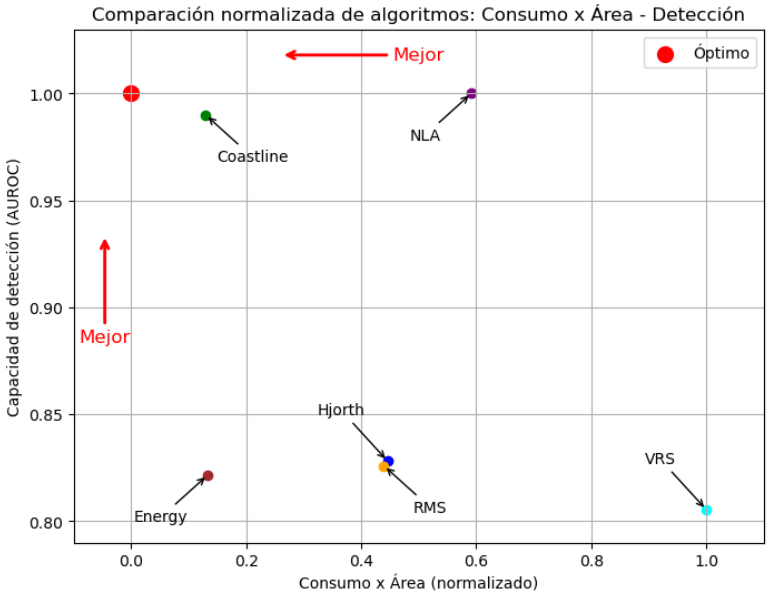


Figura 6.4: Ilustración comparativa de algoritmos: Consumo x Área en función de Capacidad de detección. Área refiere a cantidad de elementos lógicos. Capacidad de detección refiere a valor de AUROC. Se indica en el punto rojo el punto en el plano con mayor capacidad de detección y menor consumo x área. Las flechas rojas representan la dirección de mejora en los ejes.

En primer lugar, se observa que el algoritmo VRS presenta una baja capacidad de detección en comparación con los demás y una alta figura de mérito. Sin embargo, utiliza menos elementos lógicos que los algoritmos RMS y Hjorth. Esto evidencia que el consumo del algoritmo VRS afecta significativamente su rendimiento. Asimismo, tanto el Hjorth como el RMS no presentan diferencias al compararlos.

Entre los algoritmos que presentan una menor eficiencia en detección, el Energy es el que manifiesta una menor figura de mérito. Esto se debe a que no sólo presenta un menor consumo, sino que también utiliza una menor cantidad de elementos lógicos que el Hjorth, debido a la simplicidad de sus operaciones.

Finalmente, los algoritmos que presentan una mayor eficiencia en detección son el Coastline y el NLA. La diferencia entre ellos reside en que el NLA presenta una mayor figura de mérito. Esto se debe a que es significativamente superior tanto en el uso de elementos lógicos como en el consumo, duplicando sus valores respecto al Coastline.

Se puede concluir entonces que el Coastline presenta el mejor rendimiento respecto de los demás algoritmos.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Conclusiones

7.1. Conclusiones

Reducir el impacto de trastornos neurológicos en la salud de la población es medular en la misión de distintos entes y profesionales en todo el mundo.

Esto se vuelve evidente al encontrarse con la gran cantidad de bases de datos a disposición en foros públicos, que incluyen señales neurológicas de pacientes con diferentes trastornos. En el presente trabajo se limita la búsqueda a señales que pertenecieran a pacientes con epilepsia focal refractaria y que estuvieran etiquetadas con anotaciones temporales donde indicara el inicio y fin de crisis electrográficas. Finalmente, la base de datos seleccionada se destacó por la posibilidad de observar las señales etiquetadas previo a su descarga, permitiendo seleccionar los canales de interés para su análisis. Además, cada estudio considerado cuenta con documentación en la que se detalla información del paciente y registro de actividad neuronal. Sumado a esto, se encontraron herramientas para interactuar fácilmente con el portal, permitiendo descargar y acondicionar las señales para su uso. En particular, se estableció la frecuencia de muestreo de todas las señales a 250 Hz para simplificar su posterior procesamiento.

La implementación inicial de los algoritmos en Python permitió realizar análisis comparativos de detección y establecer los parámetros clave para su posterior implementación en hardware. La flexibilidad en el diseño de los algoritmos, incluyendo la posibilidad de ajustar el tamaño de la ventana y su configuración como móvil, permitió una variedad de análisis escenciales para evaluar tanto la capacidad de detección como los tiempos de reacción del diseño.

El uso de las curvas ROC como método de evaluación cuantitativa aseguró una comparación precisa entre los algoritmos, y la elección de 80 puntos para construir estas curvas garantizó la representatividad de los valores AUROC obtenidos.

Además, el estudio sobre la sustitución de números flotantes por enteros mostró

que esta simplificación no comprometió la capacidad de detección, facilitando así una implementación más eficiente en hardware sin pérdida de precisión en los resultados.

El análisis de la eficiencia en la detección de crisis epilépticas reveló que los algoritmos Coastline y NLA se destacan significativamente, alcanzando una AUROC cercana al 90 %, lo que indica una mayor capacidad de detección en comparación con los algoritmos Energy, Hjorth, RMS y VRS, que presentan una AUROC promedio de alrededor del 60 %. Este hallazgo resalta la efectividad de los algoritmos Coastline y NLA en condiciones prácticas de detección. Esta comparación se realizó con tamaños de ventanas fijos que suponen tiempos de reacción razonables para la detección de crisis.

Por otra parte, variando el tamaño de las ventanas, se observó que la capacidad de detección mejora al aumentar el tamaño de la ventana en todos los casos. En particular, el algoritmo Hjorth muestra además una leve mejora al reducir el tamaño de la ventana a 64 muestras. Esto resulta positivo dado que permite disminuir el tiempo de reacción. A pesar de esto, el algoritmo Coastline presenta una AUROC superior para el mismo tamaño de ventana.

Por su cuenta, el algoritmo NLA también presenta un aumento en su capacidad de detección para mayores tamaños de ventanas. Sin embargo, para que el incremento sea significativo, es necesario aumentar considerablemente el tiempo de reacción, por fuera de rangos razonables. Por esta razón, mantenerse en el rango de ventanas mas pequeñas optimiza tanto la AUROC como el tiempo de reacción.

Haciendo uso de la funcionalidad de ventanas móviles, se estudió el efecto de su implementación en la capacidad de detección de los algoritmos. Los resultados sugieren que cualquier superposición de muestras entre ventanas consecutivas no genera cambios significativos en la AUROC para ningún algoritmo. Si bien no hay un incremento en la capacidad de detección, este resultado implica que se puede disminuir el tiempo de reacción considerablemente sin sufrir una consecuencia negativa en la eficacia. Sin embargo, la implementación de ventanas móviles en hardware requiere el almacenamiento de datos en memoria. Como ya se comprobó, esto aumenta considerablemente el consumo, en especial si la cantidad de datos a almacenar es elevado. En consecuencia, se descarta la posibilidad de implementar los algoritmos con ventanas móviles en hardware a causa de que no se observa una mejora en detección y su mejora en tiempo de reacción se ve eclipsada por la dificultad en su implementación y su incremento de consumo.

El análisis de la variación en la representación de bits de entrada destaca que, mientras que la detección de los algoritmos Energy, Hjorth, RMS y VRS se ve afectada negativamente a partir de 5 bits, los algoritmos Coastline y NLA mantienen una robustez notable, permitiendo la posibilidad de utilizar menos bits sin comprometer la detección. Esta característica no solo contribuye a una mayor fle-

xibilidad en el diseño, sino que también puede potencialmente mejorar el consumo del sistema.

Este análisis ha puesto de manifiesto la relación directa entre la resolución de la señal y la eficiencia en el uso de recursos del FPGA. A pesar de que se implementó un sistema utilizando 12 bits para la representación de la señal, los resultados obtenidos indican que una cantidad significativamente menor de bits, específicamente 6 bits, es suficiente para mantener un rendimiento óptimo en la detección de eventos epilépticos a través de todos los algoritmos evaluados. La reducción en el número de bits no solo simplifica el diseño, sino que también minimiza el área ocupada en el FPGA y, por ende, el consumo asociado. Este hallazgo sugiere que se pueden lograr resultados satisfactorios sin necesidad de una alta resolución en la representación de la señal.

Con base en los resultados obtenidos sobre la utilización de recursos en el FPGA, se registró la cantidad de elementos lógicos empleados por cada algoritmo tras su implementación.

Entre los algoritmos evaluados, el Coastline se destaca por ser el que menos recursos requiere, lo que lo convierte en la opción más eficiente en términos de uso de elementos lógicos. A este le sigue el Energy, que también muestra un uso moderado de recursos.

Por el contrario, los algoritmos Hjorth y RMS requieren una mayor cantidad de recursos, lo que se atribuye a su complejidad matemática y a las operaciones más extensas que requieren para su implementación. En una posición intermedia se encuentran el NLA y VRS, que reflejan un equilibrio entre la complejidad del procesamiento y la eficiencia en el uso de recursos.

Es importante señalar que, al implementar estos diseños, no solo se considera la ocupación de elementos lógicos del algoritmo, sino también la del diseño de los Sistemas 1 y 2. La comparación revela que, a pesar del aumento general en el uso de recursos al implementarlos, el Coastline sigue siendo el algoritmo más eficiente en términos de ocupación. El Energy también mantiene un uso razonable de recursos. En contraste, los algoritmos Hjorth y RMS, debido a su mayor complejidad, requieren más recursos, lo que debe considerarse al seleccionar el algoritmo adecuado para aplicaciones específicas.

El análisis del consumo de potencia de los algoritmos implementados ha permitido obtener valiosas conclusiones sobre su eficiencia en términos de recursos. El algoritmo Energy se destaca por su operación sencilla de promedio de cuadrados, lo que resulta en un bajo consumo energético al no requerir almacenamiento intermedio ni cálculos complejos. En comparación, el Coastline, aunque simple en su cálculo, incorpora un registro para el almacenamiento de datos, lo que incrementa su consumo en relación al Energy.

El algoritmo Hjorth, debido a su complejidad matemática, requiere más sumas y multiplicaciones, resultando en un mayor consumo. Por otro lado, el RMS, muestra un consumo similar al Hjorth, aunque ligeramente inferior, debido a la complejidad que implica la operación de raíz cuadrada en un FPGA.

El NLA presenta un consumo significativamente mayor debido a la necesidad de buscar máximos y mínimos en las ventanas de datos, como también su registro, lo que incrementa la cantidad de elementos de hardware necesarios. Finalmente, el VRS es el algoritmo con el mayor consumo energético, justificado por su necesidad de utilizar memoria RAM para el almacenamiento temporal de datos y el proceso asociado al cálculo de varianza, que implica un uso intensivo de recursos.

La jerarquía de consumo entre los algoritmos es clara: el VRS presenta el mayor consumo, seguido del NLA, Hjorth, y Coastline mientras que el Energy se mantiene como el más eficiente. Esta información es crucial para la selección del algoritmo más adecuado, especialmente en aplicaciones donde el consumo energético es un factor determinante.

La elección de la placa DE0 no resultó ser la más apropiada, debido a que no se encuentra adaptada para mediciones de consumo. Igualmente, se logró realizar las medidas necesarias gracias a un equipo modificado externamente.

En resumen, los resultados obtenidos de la implementación de los algoritmos revelan que, aunque el Coastline y el NLA son los más eficientes en la detección de crisis epilépticas, el algoritmo Energy se destaca por su bajo consumo energético, seguido por el Coastline. Por otro lado, el VRS, a pesar de su alto consumo, presenta una capacidad de detección limitada, lo que lo posiciona como la opción menos favorable. Además, se observó que el tamaño de la ventana y la representación de bits influyen significativamente en la eficiencia de detección, siendo fundamental seleccionar parámetros que optimicen el rendimiento.

A través de la figura de mérito, se ha podido establecer una referencia clara para comparar la relación entre el consumo y el área ocupada por cada algoritmo. Esta métrica resulta crucial para la optimización en el diseño, donde el espacio y la eficiencia son factores determinantes. El Coastline y el NLA demostraron ser los más eficientes en esta evaluación, mientras que el Energy a pesar de sus limitaciones en detección ofrece ventajas en términos de simplicidad y bajo consumo.

7.2. Trabajo a futuro

Una opción prometedora para mejorar la eficiencia de los algoritmos implementados es el uso de una representación de 6 bits en lugar de 12. Esta reducción no solo podría disminuir significativamente el consumo al minimizar la cantidad de

operaciones requeridas para manejar los datos, sino que también podría resultar en un uso más eficiente de los elementos lógicos en el FPGA. Al simplificar la complejidad de los cálculos y la cantidad de memoria necesaria para las operaciones intermedias, se espera que esta modificación optimice el rendimiento general de los algoritmos, facilitando su implementación en sistemas con limitaciones de recursos. Asimismo, el algoritmo Coastline mostró un rendimiento prometedor incluso con una representación de solo 2 bits. Esto implica una gran optimización de los recursos, que podría minimizar el consumo. Por lo tanto, se considera que esta implementación es otra posible línea de investigación futura.

Con el fin de realizar las medidas de consumo en la placa DE0, fue necesario utilizar un equipo modificado previamente. Esto pudo afectar la precisión de las medidas registradas de consumo. El uso de una placa con una funcionalidad que permita obtener medidas de consumo sin necesidad de realizar modificaciones, permitiría obtener datos más exactos, además de facilitar el proceso de comparación entre los algoritmos.

Para realizar operaciones complejas en Hardware, múltiples algoritmos utilizan bloques generados por Quartus, los cuales deben modificarse si se desea trabajar con otro dispositivo. Por esta razón, aplicar estimaciones matemáticas que sustituyan las operaciones complejas, y por lo tanto, los bloques, puede mejorar la escalabilidad del sistema. Adicionalmente, se puede analizar si esto tiene un impacto en el consumo y en la capacidad de detección de los algoritmos.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

API para descarga de señales

El código que se presenta a continuación se obtuvo de un repositorio que incluye herramientas para interactuar con el portal IEEG.org. [21]

```
1
2  import sys
3  from ieeg.auth import Session
4  from ieeg.processing import ProcessSlidingWindowPerChannel,
       ProcessSlidingWindowAcrossChannels
5  from pennprov.connection.mprov_connection import
       MProvConnection
6
7  import matplotlib.pyplot as plt
8  import numpy as np
9
10 from scipy import signal
11 from statistics import mean
12 import pickle as pkl
13 import os
14
15 default_user = 'sample'
16 default_password = 'default'
17 data_base = 'Study 006' #'Study 005'
18
19
20 if len(sys.argv) < 3:
21     print('To run this sample program, you must supply your
          user ID and password on the command-line')
22     print('Syntax: read_sample [user id (in double-quotes if
          it has a space)] [password] [Prov userID] [Prov
          Password]')
23     sys.exit(1)
24
25 print ('Logging into IEEG:', sys.argv[1], '/ ****')
26 with Session(sys.argv[1], sys.argv[2]) as s:
27
28     conn = None
29     if len(sys.argv) > 3:
```

```
30          default_user = sys.argv[3]
31          default_password = sys.argv[4]
32          print ('Logging into local MProv: ', default_user, '/
                ****')
33          conn = MProvConnection(default_user, default_password
                , None)
34          print("Successfully connected to the MProv server")
35
36      # We pick one dataset...
37      ds = s.open_dataset(data_base)#s.open_dataset('
            I004_A0003_D001')
38
39      # Iterate through all of the channels and print their
            metadata
40      for name in ds.get_channel_labels():
41          print (ds.get_time_series_details(name))
42
43      print("annotation..")
44      print(ds.get_annotation_layers())
45      print(ds.get_annotation_layers().keys())
46      #print (ds.get_annotations(list(ds.get_annotation_layers
            ().keys())))
47      print(ds.get_annotation_layers().items())
48      #print(list(ds.get_annotation_layers().keys()))
49
50      for layer_name in ds.get_annotation_layers().keys():
51          print("Layer name: ", layer_name)
52          #print (ds.get_annotations(layer_name))
53          start = [];
54          end = [];
55          for annotation in ds.get_annotations(layer_name):
56              if annotation.type == 'Seizure':
57                  start.append(annotation.
                        start_time_offset_usec);
58                  end.append(annotation.end_time_offset_usec);
59                  print("annotation val: ", annotation.
                        start_time_offset_usec, annotation.type)
60
61      start = np.array(start);
62      end = np.array(end);
63
64
65      labels = [['LS1'], ['LS2'], ['LS3']]; #'Study 006'
66      target_frequency = 250; # Hz
67      window_width = 10*60e6; #us
68      preictal_time = 100e6;
69      preictal_window = int(preictal_time//window_width);
70      shift = 100e3; # 100 ms
71      shift_steps = int(window_width//shift); # 100 steps
72      accum = 0;
73      for start_time in start:
```

```
74          for label in labels:
75              print(start_time-window_width+60e6)
76              if (start_time-window_width+60e6) > 1:
77                  sequence_from_get_data = np.squeeze(ds.
                        get_data(start_time-window_width+60e6,
                        window_width, ds.get_channel_indices(label
                        )));
78                  ts = ds.get_time_series_details(label[0]);
79                  current_frequency = ts.sample_rate;
80                  downsampling_factor = round(current_frequency
                        /target_frequency); # works if current
                        freq approx N*250 Hz
81
82                  sequence_data_at_250Hz = signal.decimate(np.
                        asarray(sequence_from_get_data, dtype='
                        float64'), downsampling_factor, ftype='fir
                        '); # downsampling at 250 Hz
83
84                  filename = '/Users/Usuario/Desktop/DatosEpi/'
                         + data_base + label[0] + str(start_time)
85                  filename = filename.replace(" ", "");
86                  print(filename)
87                  fileObject = open(filename, "wb+")
88                  pkl.dump(sequence_data_at_250Hz, fileObject)
89                  fileObject.close()
90                  accum += 1;
91
92          ##### TO READ DATA ############
93          #for filename in os.listdir('/Users/smartinez/
                Downloads/dataseries/preictal/'):
94          #    f = os.path.join('/Users/smartinez/Downloads/
                dataseries/preictal/', filename);
95          #    if os.path.isfile(f):
96          #        with open(f, 'rb') as fileObject:
97          #            data = pkl.load(fileObject)
98          #            plt.plot(data);
99          #            plt.show();
100         #            print(f)
101         #exit()
102
103     s.close_dataset(ds)
104     print("total snapshots: ", accum)
105     accum = 0;
106     exit()
```

Listing A.1: API para descarga de señales de portal IEEG.org

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Lista de señales utilizadas

Estudio	Duración señal (min)	Cantidad crisis
Study005LTD1168257000000	30	4
Study019LT21147366000000	50	3
Study030LG39436961000000	10	1
Study030LG39456090000000	10	1

Tabla B.1: Señales iniciales.

Estudio	Duración señal (min)	Cantidad crisis
Study005LTD1100147350543	10	1
Study005LTD1105713292243	3	1
Study005LTD1106116334259	3	1
Study005LTD1107251522510	3	1
Study005LTD1109590749772	3	1
Study005LTD1111656361139	3	1
Study005LTD1112956116000	3	1
Study005LTD1116354921448	3	1
Study005LTD1116999605414	3	1
Study005LTD1118474077821	3	1
Study005LTD1120030098351	3	1
Study005LTD1121412160524	3	1
Study005LTD1125670828008	3	1
Study005LTD1132500871418	3	1
Study005LTD1133327355200	3	1
Study005LTD1137450643456	3	1
Study005LTD1138784070581	3	1
Study005LTD1139592027964	3	1
Study005LTD1141293688584	3	1
Study005LTD1144067442720	3	1
Study005LTD1146942583761	3	1

Estudio	Duración señal (min)	Cantidad crisis
Study005LTD1149839253761	3	1
Study005LTD1151241547696	3	1
Study005LTD1152891167683	3	1
Study005LTD1155072797631	3	1
Study005LTD1156907167983	3	1
Study005LTD1160264069637	3	1
Study005LTD1163150176672	3	1
Study005LTD1169037396178	3	1
Study005LTD1169456163120	3	1
Study005LTD1169965625932	3	1
Study005LTD1170892393368	3	1
Study005LTD1171302161636	3	1
Study005LTD1171885036057	3	1
Study005LTD1173018116916	3	1
Study005LTD1173319034937	3	1
Study005LTD1173886337509	3	1
Study005LTD1174879298256	3	1
Study005LTD1176052779630	3	1
Study005LTD1176647885374	3	1
Study005LTD1177850808223	3	1
Study005LTD1178509984012	3	1
Study005LTD1179631194488	3	1
Study005LTD1186092858895	3	1
Study005LTD1186573656375	3	1
Study005LTD1187799831550	3	1
Study005LTD1188296387916	3	1
Study005LTD1189656623029	3	1
Study005LTD1190601905953	3	1
Study005LTD1191624980304	3	1
Study005LTD1192486559660	3	1
Study005LTD1193196111783	3	1
Study005LTD1193866868551	3	1
Study005LTD1195209287302	3	1
Study005LTD1195814118855	3	1
Study005LTD1196205593674	3	1
Study005LTD1197272233173	3	1
Study005LTD1198989107625	3	1
Study005LTD1200184561085	3	1
Study005LTD1201480999331	3	1
Study005LTD1202375008719	3	1
Study005LTD1203799112711	3	1
Study005LTD1205923849076	3	1
Study005LTD1206495225404	3	1
Study005LTD1207099899927	3	1

Estudio	Duración señal (min)	Cantidad crisis
Study005LTD1210562569255	3	1
Study019LT154264593542	3	1
Study019LT1510625404189	3	1
Study019LT1514936213327	3	1
Study019LT1523667038963	3	1
Study019LT1526064315068	3	1
Study019LT1529440721350	3	1
Study019LT1543240238191	3	1
Study019LT1623667038963	3	1
Study019LT1626064315068	3	1
Study019LT1629440721350	3	1
Study019LT1643240238191	3	1
Study019LT15428150254943	3	1
Study019LT16109700699869	3	1

Tabla B.2: Otras señales procesadas.

Estudio	Duración señal (min)	Cantidad crisis
Study006LG159736528754	3	1
Study006LG169736528754	3	1
Study006LG239736528754	3	1
Study006LG249736528754	3	1
Study006LG379736528754	3	1
Study006LG389736528754	3	1
Study006LG1528251523053	3	1
Study006LG1628251523053	3	1
Study006LG2328251523053	3	1
Study006LG2428251523053	3	1
Study006LG3728251523053	3	1
Study006LG3828251523053	3	1
Study006LG15115233354116	3	1
Study006LG16115233354116	3	1
Study006LG23115233354116	3	1
Study006LG24115233354116	3	1
Study006LG37115233354116	3	1
Study006LG38115233354116	3	1
Study006LS79736528754	3	1
Study006LS728251523053	3	1
Study006LS7115233354116	3	1

Tabla B.3: Señales del Study006.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Gráficas para análisis de AUROC

Las gráficas a continuación corresponden al análisis realizado en la Sección 3.5, donde las curvas ROC se construyen con diferente cantidad de puntos. Esto permite evaluar cuantos puntos son necesarios para obtener valores de AUROC que reflejen correctamente el rendimiento de los algoritmos.

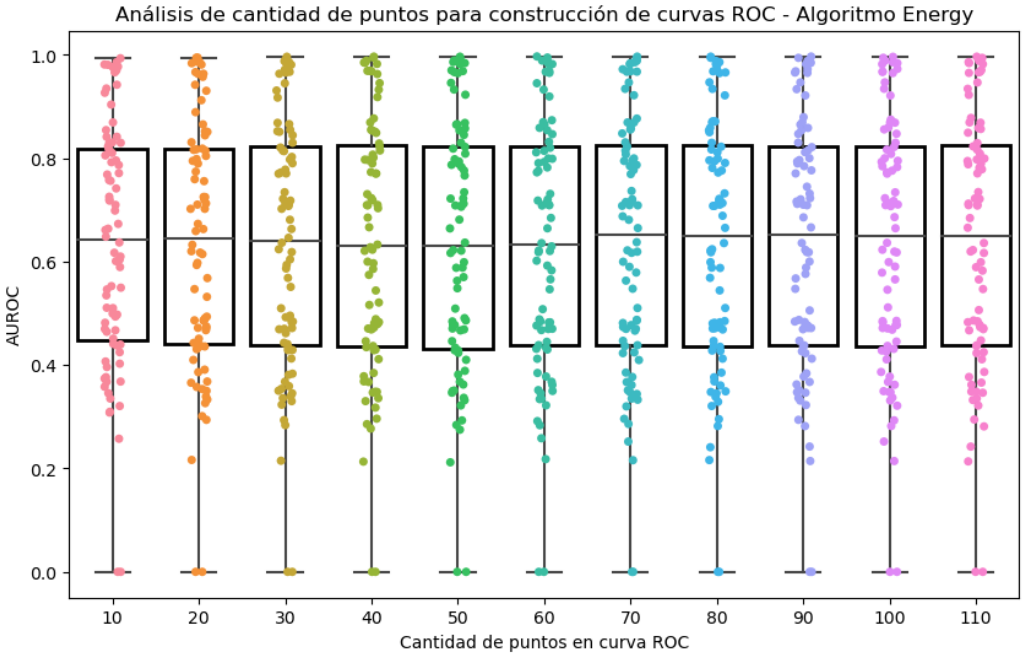


Figura C.1: Análisis de puntos de construcción de curvas ROC - Algoritmo Energy - Ventana de 256 muestras.

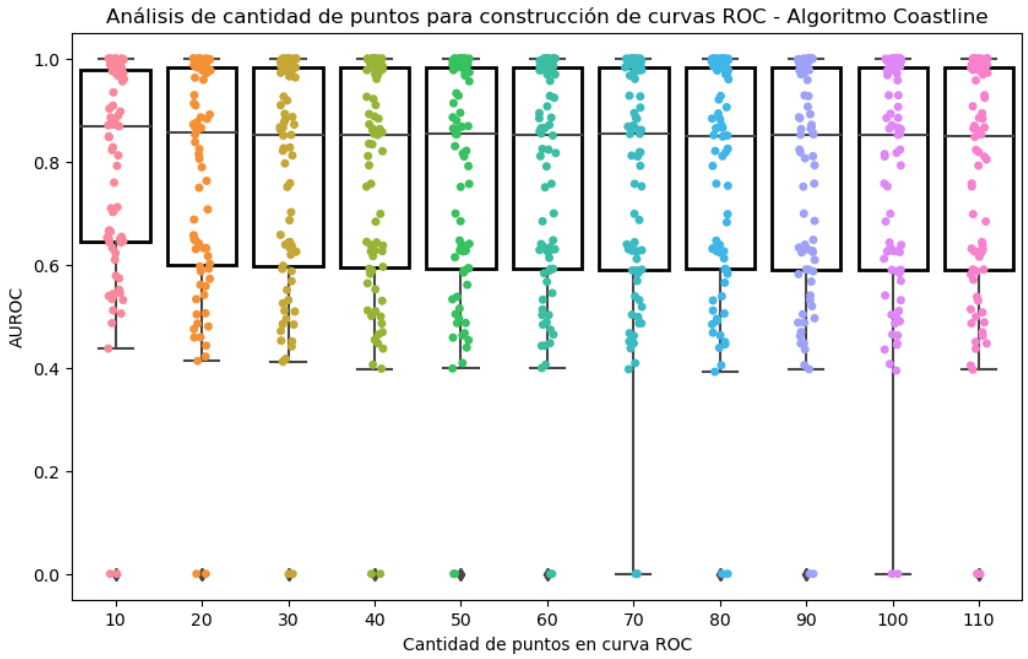


Figura C.2: Análisis de puntos de construcción de curvas ROC - Algoritmo Coastline - Ventana de 256 muestras.

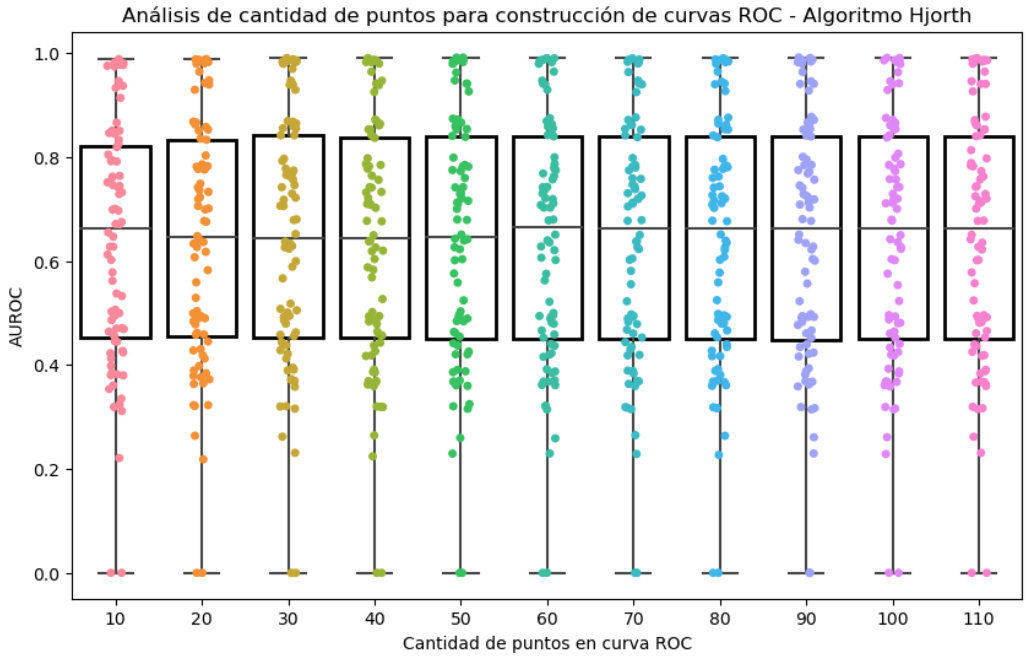


Figura C.3: Análisis de puntos de construcción de curvas ROC - Algoritmo Hjorth - Ventana de 256 muestras.

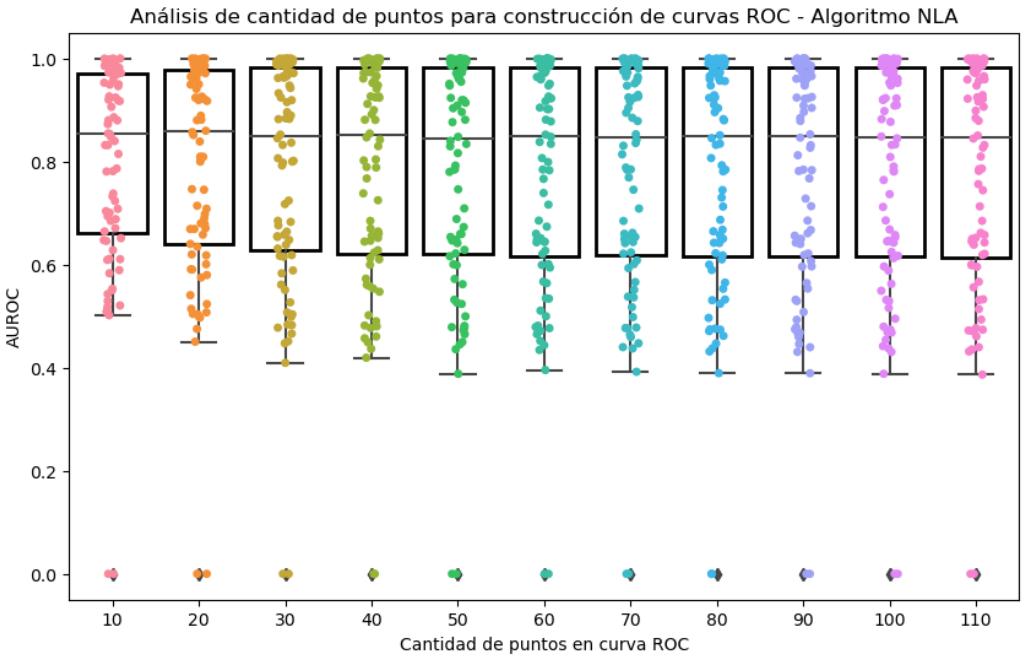


Figura C.4: Análisis de puntos de construcción de curvas ROC - Algoritmo NLA - Ventana de 40 grupos - 10 muestras por grupo.

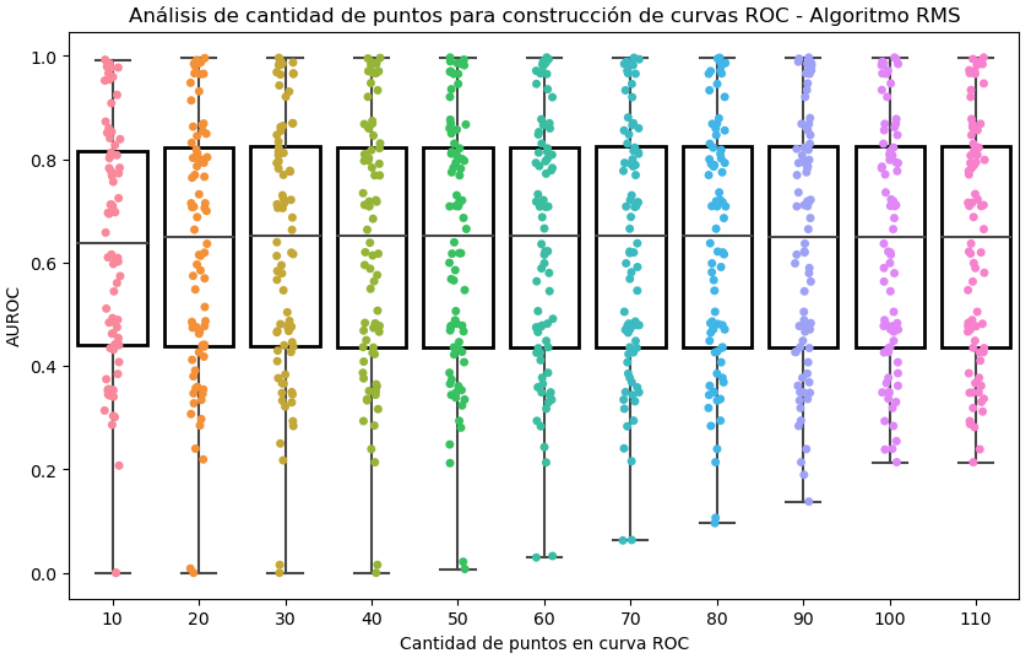


Figura C.5: Análisis de puntos de construcción de curvas ROC - Algoritmo RMS Amplitude - Ventana de 256 muestras.

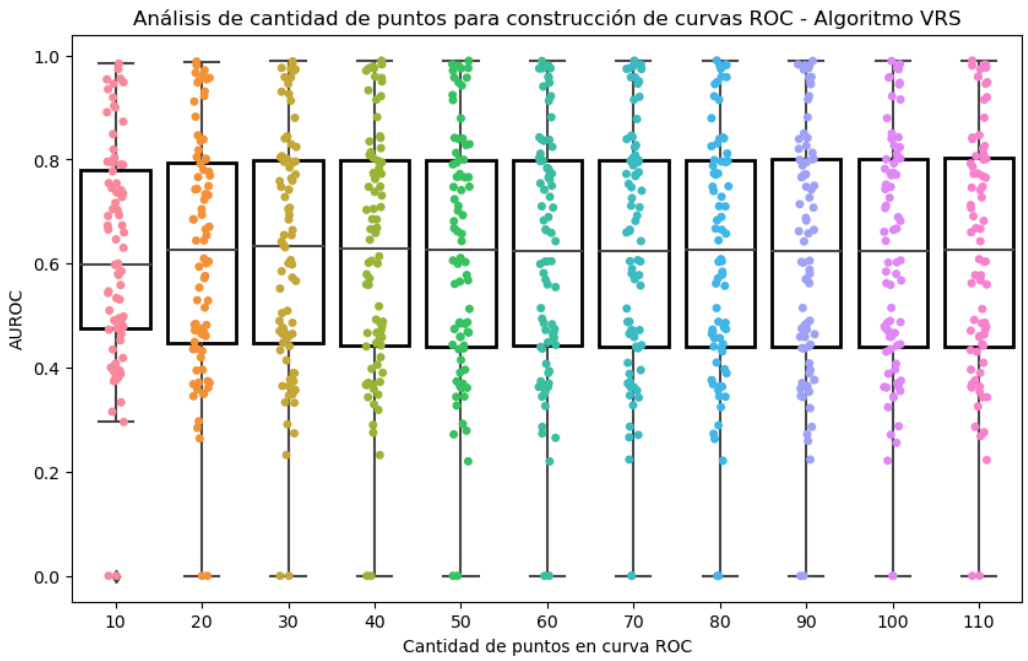


Figura C.6: Análisis de puntos de construcción de curvas ROC - Algoritmo VRS - Ventana de 256 muestras.

Apéndice D

Gráficas para análisis de ventanas

Las gráficas a continuación corresponden al análisis realizado en la Sección 3.10, donde se evalúan los algoritmos utilizando diferentes tamaños de ventanas.

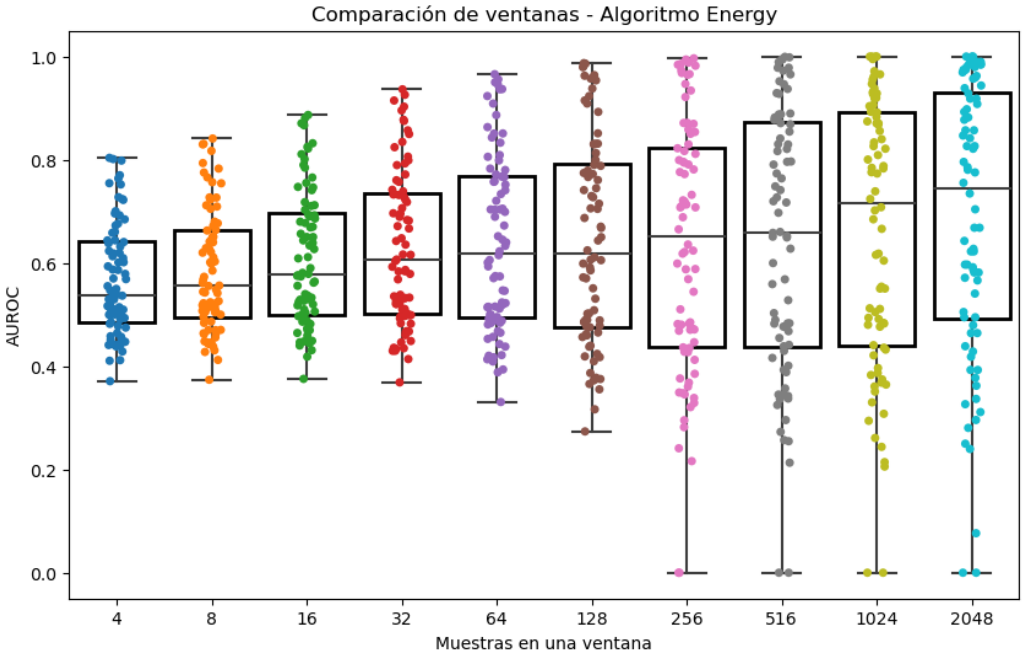


Figura D.1: Análisis comparativo de tamaño de ventanas para algoritmo Energy.

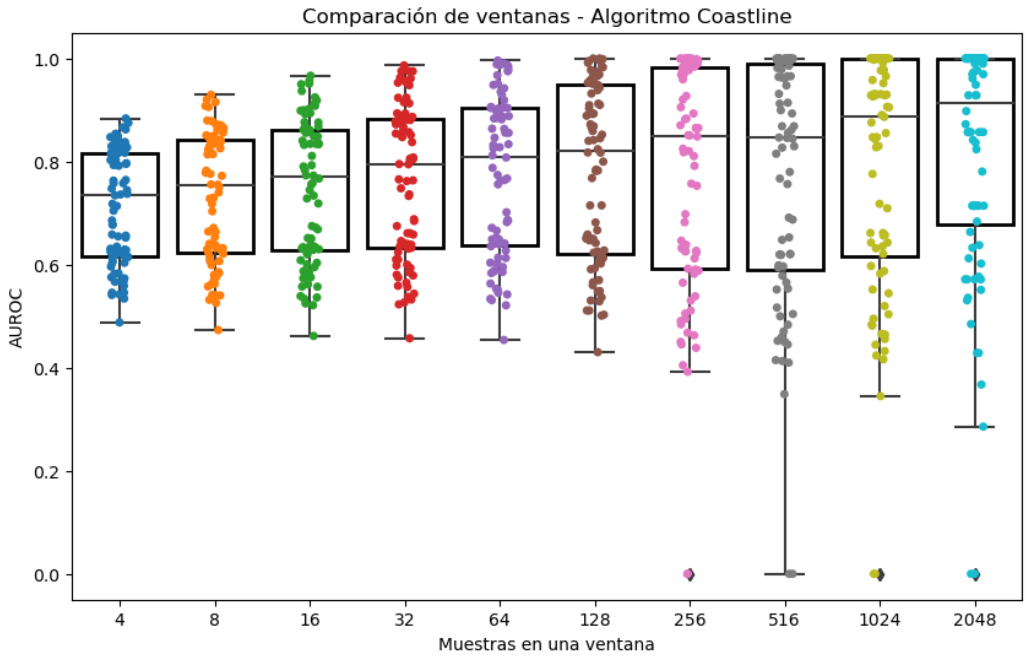


Figura D.2: Análisis comparativo de tamaño de ventanas para algoritmo Coastline.

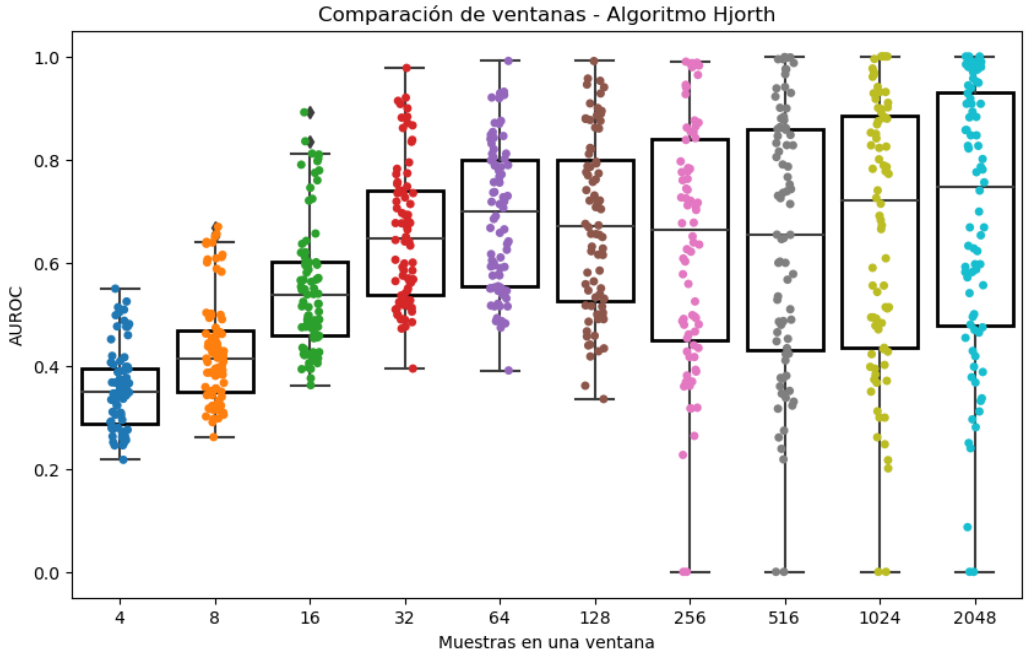


Figura D.3: Análisis comparativo de tamaño de ventanas para algoritmo Hjorth.

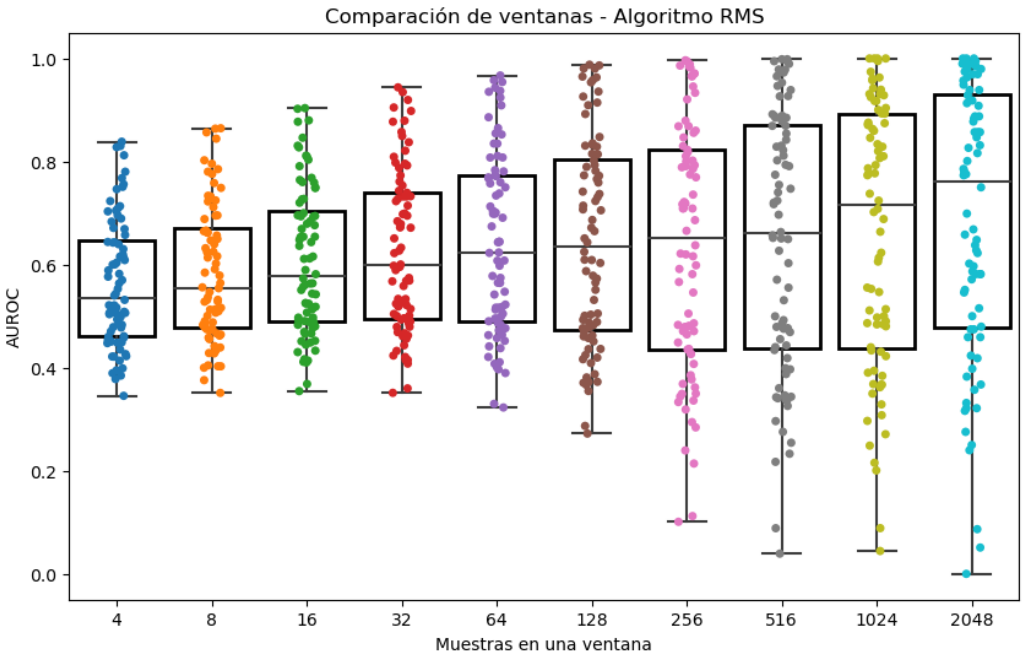


Figura D.4: Análisis comparativo de tamaño de ventanas para algoritmo RMS Amplitude.

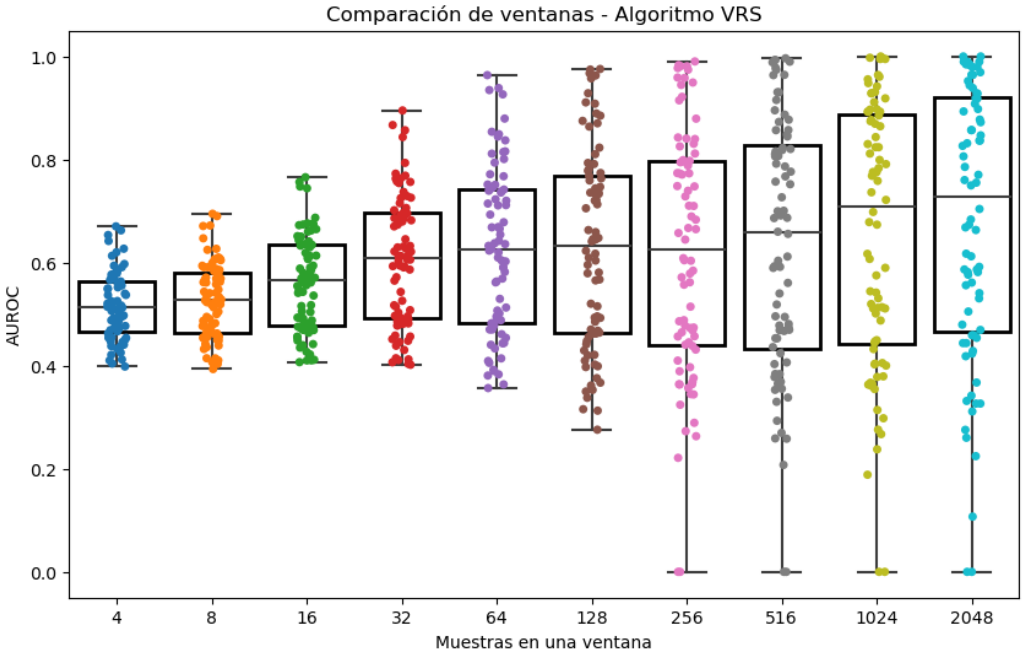


Figura D.5: Análisis comparativo de tamaño de ventanas para algoritmo VRS.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice E

Gráficas para análisis de superposición en ventanas

Las gráficas a continuación corresponden al análisis realizado en la Sección 3.11, donde se evalúan los algoritmos al cambiar la cantidad de muestras que se superponen entre ventanas.

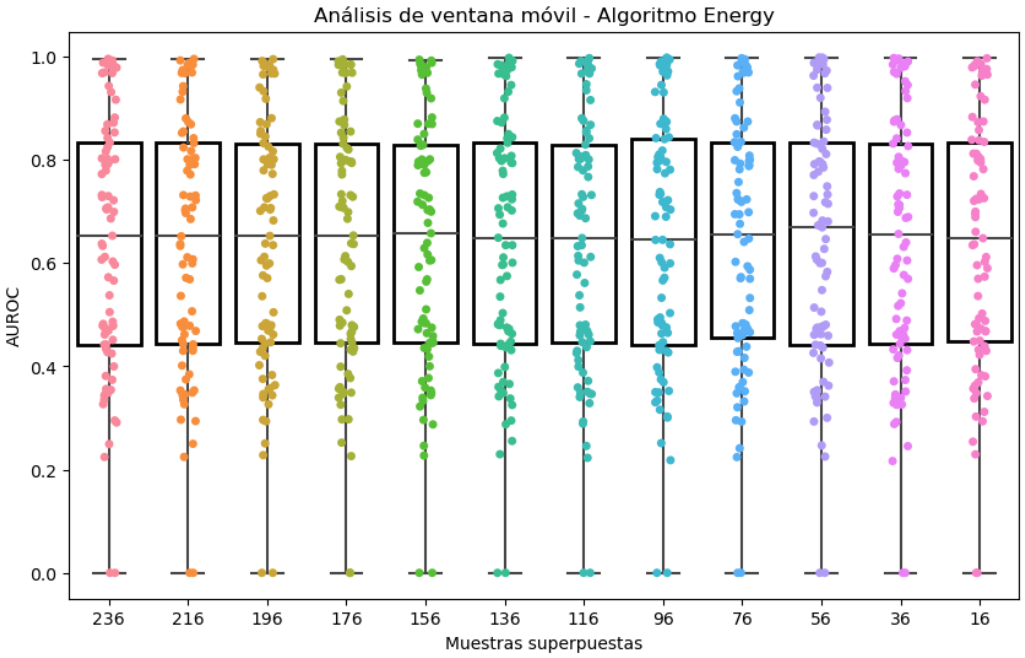


Figura E.1: Análisis comparativo de superposición de muestras para algoritmo Energy.

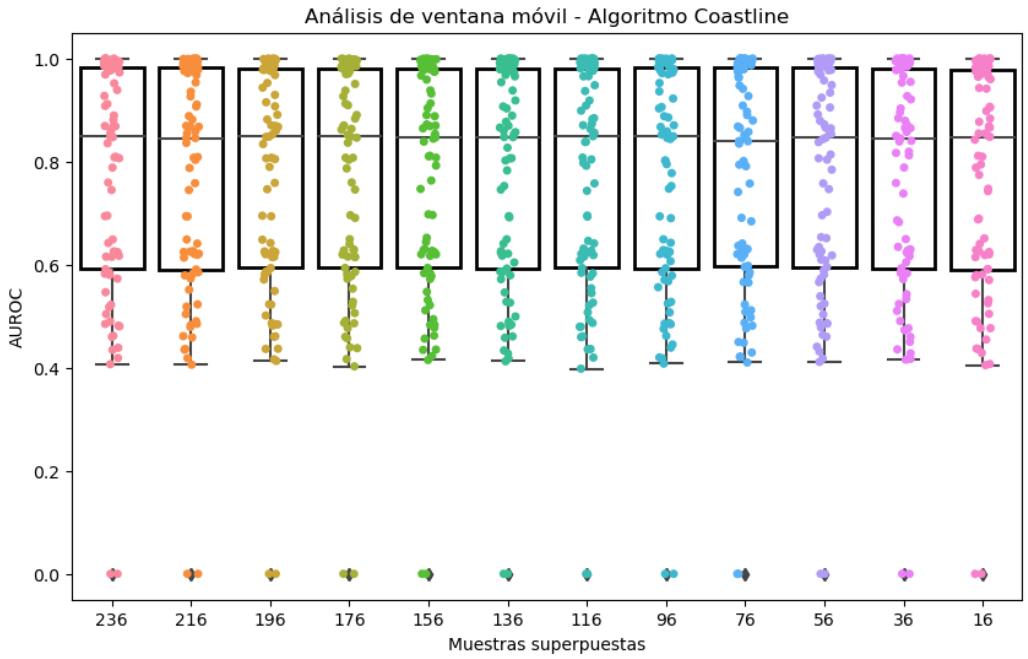


Figura E.2: Análisis comparativo de superposición de muestras para algoritmo Coastline.

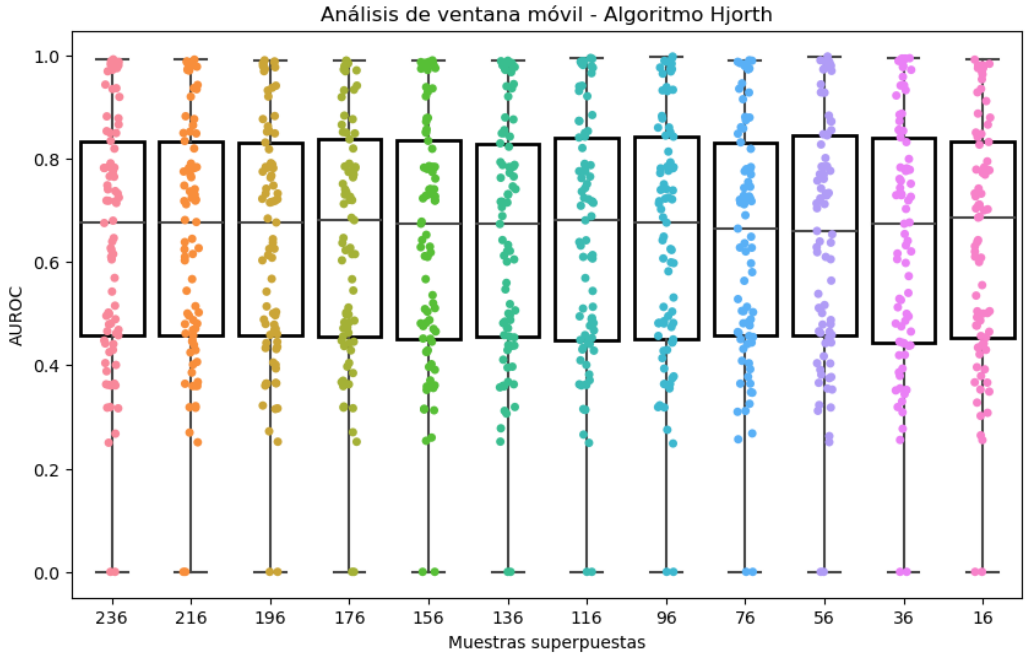


Figura E.3: Análisis comparativo de superposición de muestras para algoritmo Hjorth.

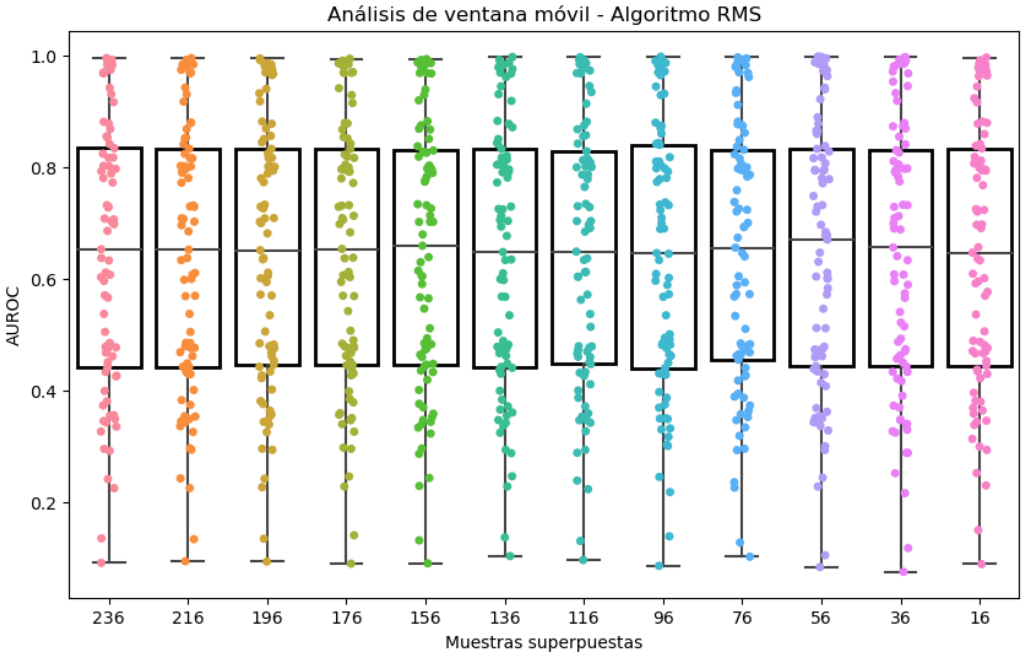


Figura E.4: Análisis comparativo de superposición de muestras para algoritmo RMS Amplitude.

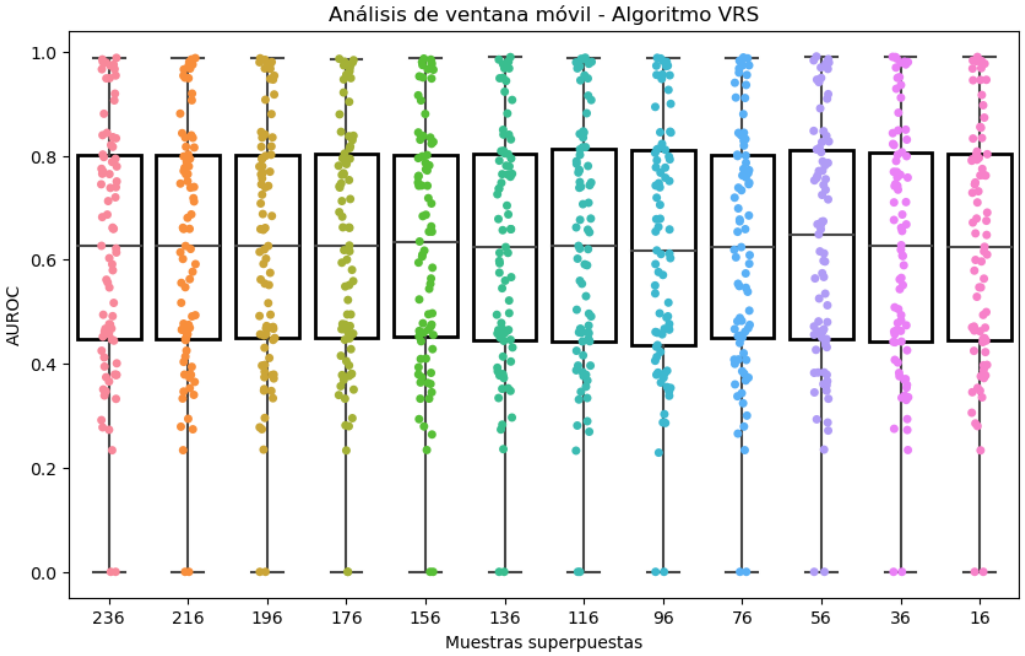


Figura E.5: Análisis comparativo de superposición de muestras para algoritmo VRS.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice F

Gráficas para análisis de cantidad de bits para representación de datos

Las gráficas a continuación corresponden al análisis realizado en la Sección 3.12, donde se evalúan los algoritmos al cambiar la cantidad de bits con la que se representan los datos.

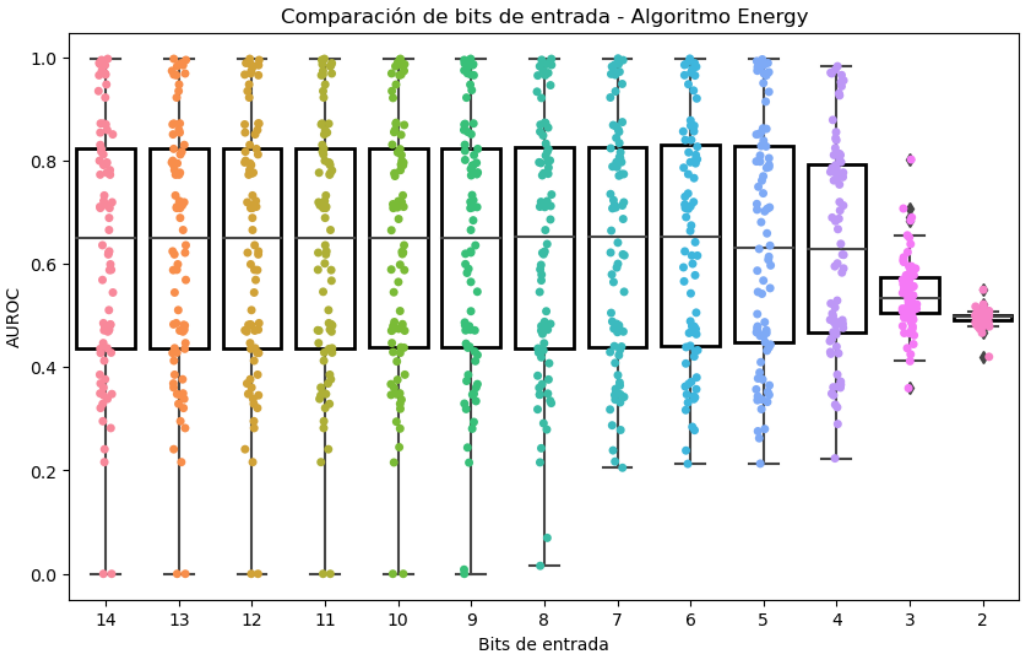


Figura F.1: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo Energy - Ventana de 256 muestras.

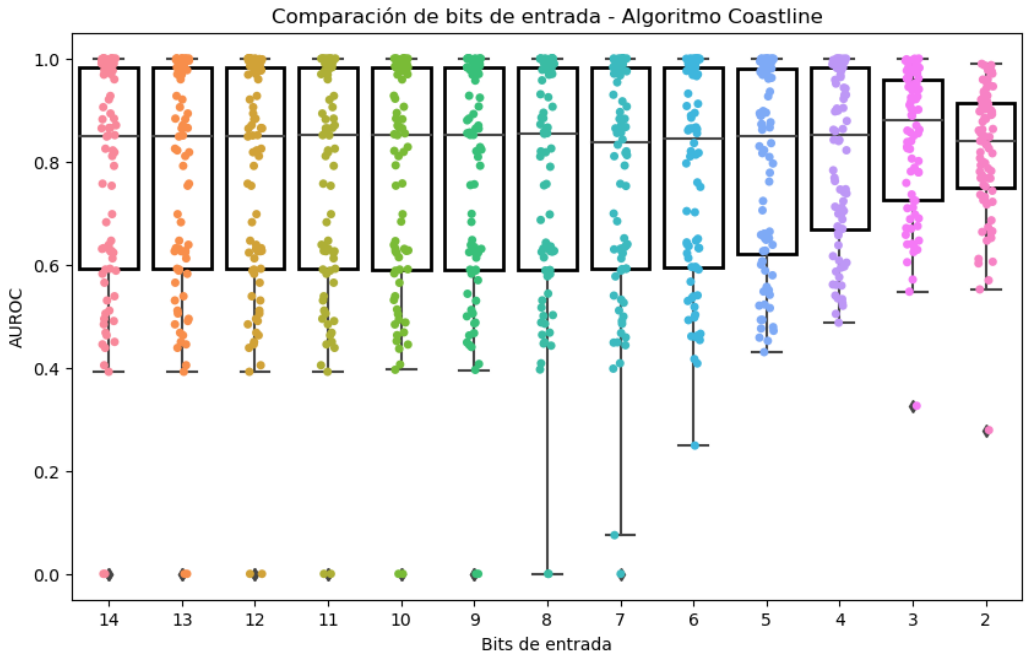


Figura F.2: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo Coastline - Ventana de 256 muestras.

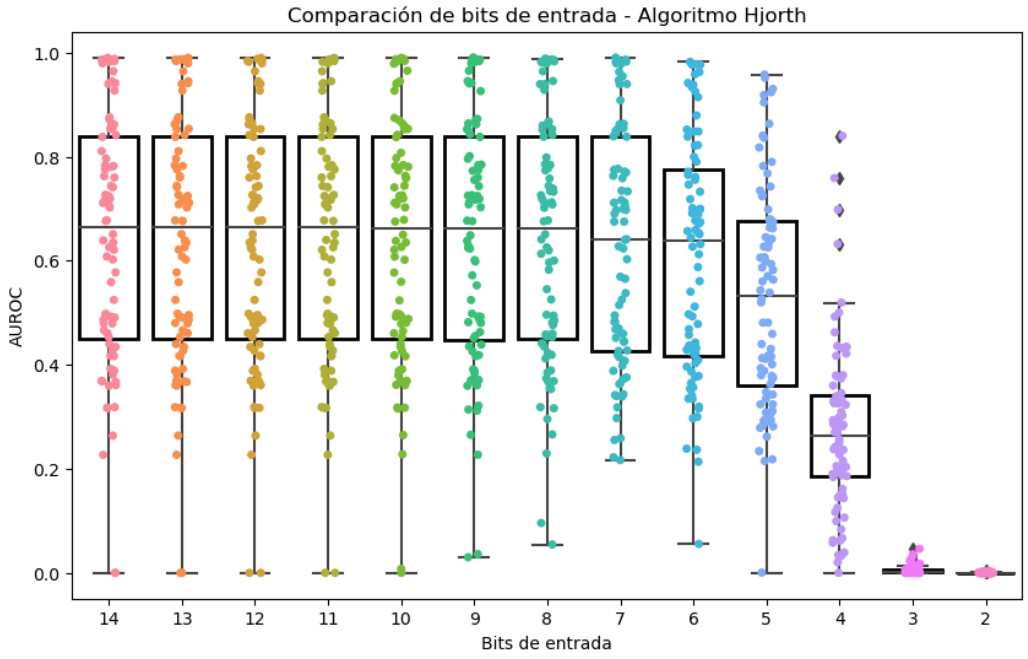


Figura F.3: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo Hjorth - Ventana de 256 muestras.

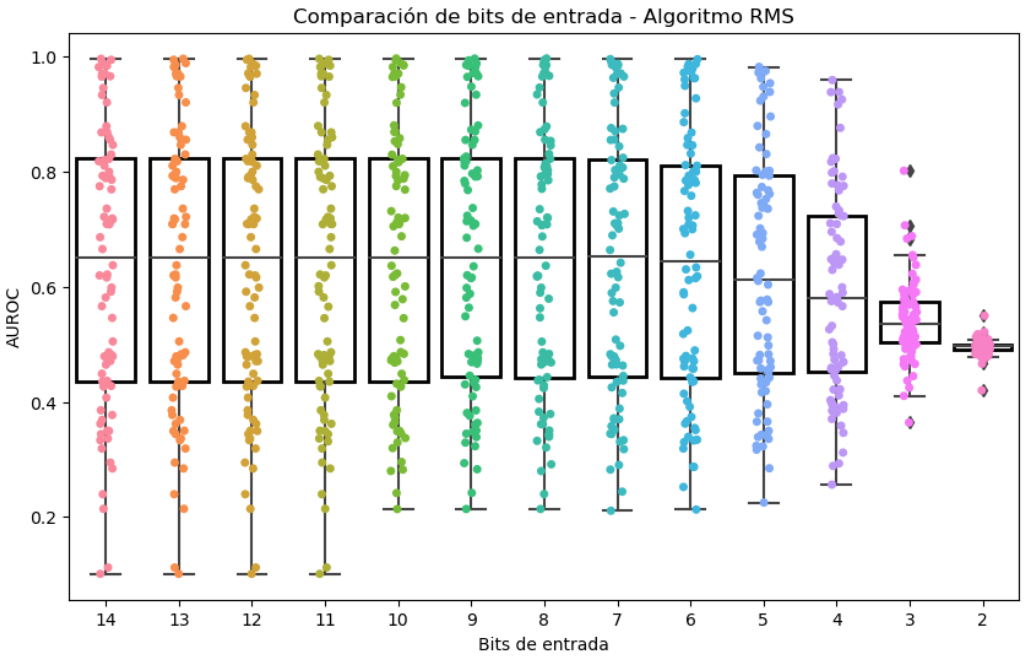


Figura F.4: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo RMS Amplitude - Ventana de 256 muestras.

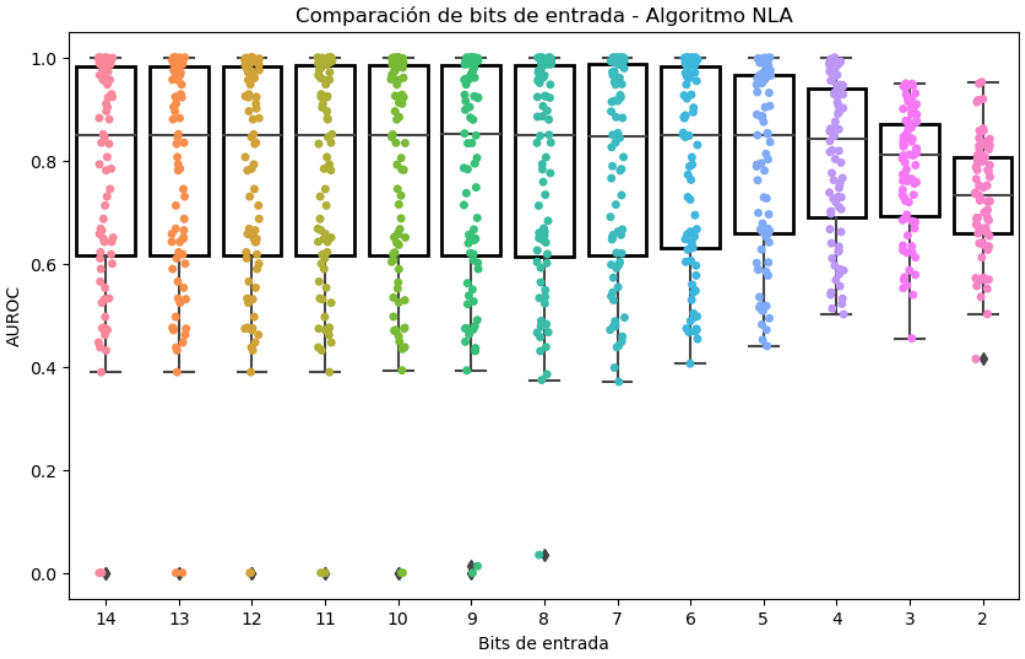


Figura F.5: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo NLA - Ventana de 256 muestras.

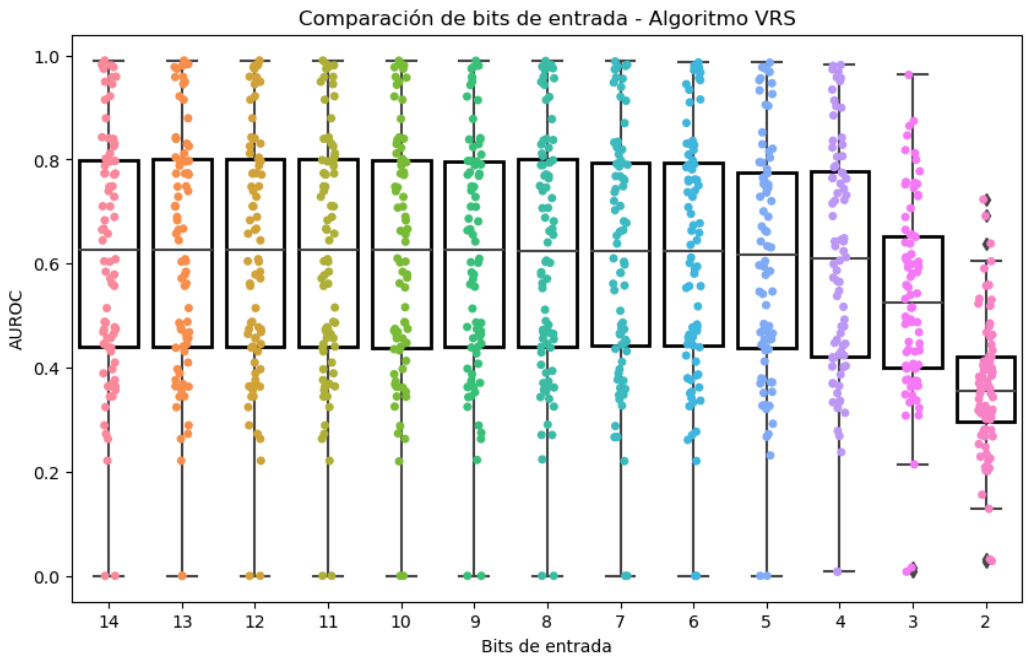


Figura F.6: Análisis comparativo de cantidad de bits para representación de datos - Algoritmo VRS - Ventana de 256 muestras.

Apéndice G

Código TCL

```
# Obtener el nombre del hardware y del dispositivo dinámicamente
set hwname [lindex [get_hardware_names] 0]
set devname [lindex [get_device_names -hardware_name ${hwname}] 0]

# Verificar si se encontraron nombres válidos
if {$hwname ne "" && $devname ne ""} {
    # Iniciar la secuencia de edición de memoria
    begin_memory_edit -hardware_name ${hwname} -device_name ${devname}

    # Especificar la ruta completa al archivo MIF
    set mif_file_path "C:/Users/Usuario/Desktop/dislog2/gr1/Proyecto/quartus/Prueba_datos_reales.mif"  ;# Reemplaza con la ruta completa

    # Verificar si el archivo MIF existe antes de intentar abrirlo
    if {[file exists $mif_file_path]} {
        # Actualizar el contenido de la memoria desde el archivo MIF
        update_content_to_memory_from_file -instance_index 0 -mem_file_path $mif_file_path -mem_file_type mif
    } else {
        puts "Error: El archivo MIF especificado ($mif_file_path) no existe."
    }

    # Finalizar la secuencia de edición de memoria
    end_memory_edit
} else {
    puts "Error: No se encontraron nombres válidos de hardware o dispositivo."
}
```

Figura G.1: Código TCL para escritura de ROM.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice H

Diagramas de bloques de los sistemas

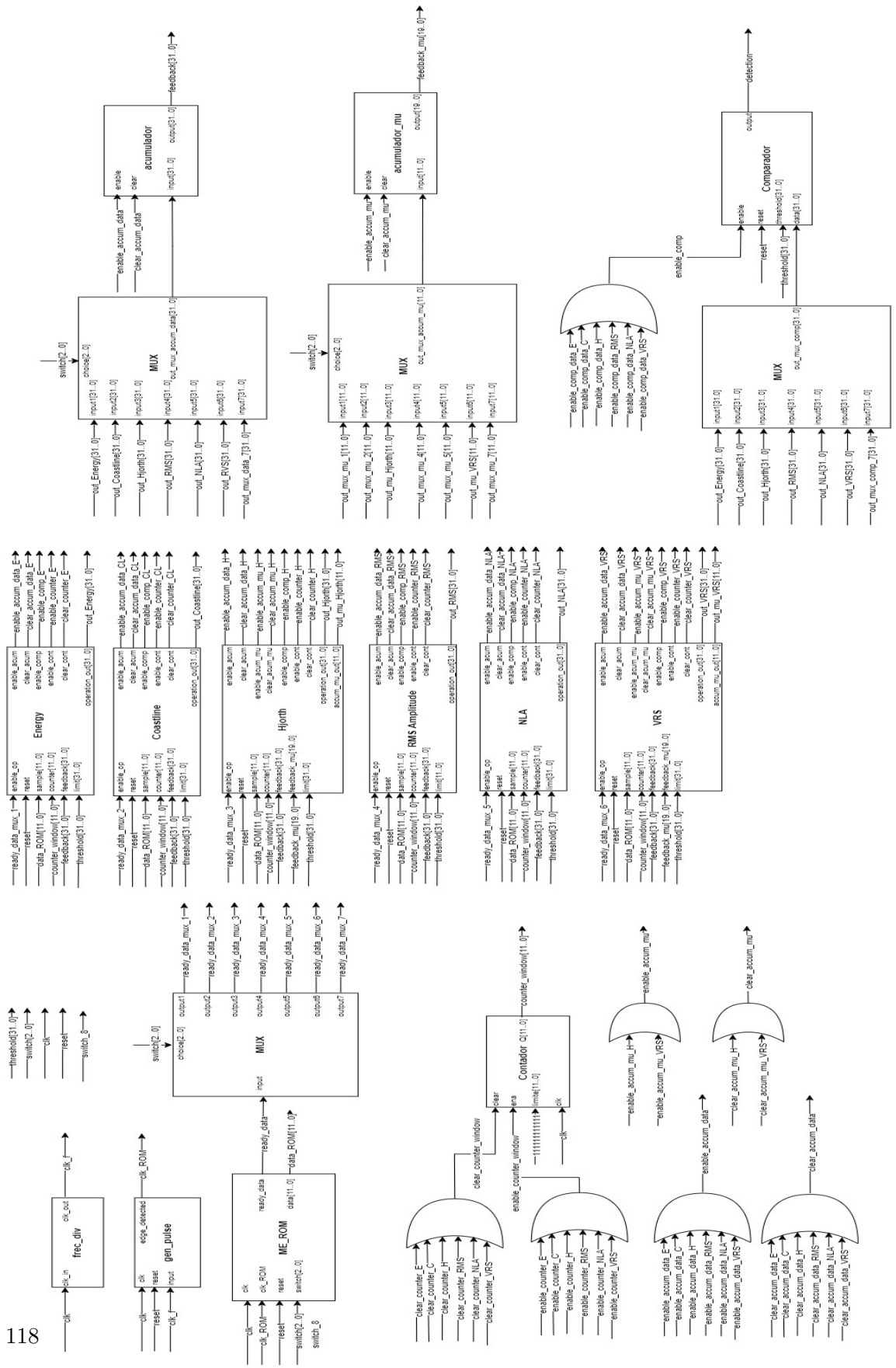


Figura H.1: Diagrama de bloques del sistema de aplicación de un único algoritmo.

Figura H.2: Diagrama de bloques del sistema de aplicación simultánea de algoritmos.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice I

Medidas de consumo obtenidas

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	1.67	0.972	1.403	0.00063	
10	6.68	0.983	1.710	0.00083	32.89
20	12.27	0.994	2.059	0.0011	33.41
30	17.86	1.009	2.398	0.0014	33.06
40	23.46	1.012	2.699	0.0017	32.23
50	29.02	1.023	3.137	0.0023	34.36
60	34.61	1.027	3.381	0.0026	32.60
70	40.19	1.034	3.606	0.0029	31.04
80	45.79	1.045	3.910	0.0034	30.82
90	51.40	1.056	4.181	0.0038	30.28
100	57.02	1.059	4.395	0.0042	29.35

Tabla I.1: Medidas de consumo del sistema de aplicación individual del Energy.

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	1.16	0.976	1.414	0.0006	
10	9.81	0.990	1.919	0.0010	54.52
20	19.39	1.005	2.455	0.0015	53.29
30	29.02	1.019	2.947	0.0020	51.38
40	38.57	1.038	3.345	0.0025	47.94
50	48.18	1.048	3.895	0.0034	49.17
60	57.75	1.066	4.274	0.0040	46.94
70	66.37	1.077	4.676	0.0047	45.81
80	76.01	1.085	5.056	0.0055	44.74
90	85.53	1.110	5.347	0.0061	42.70

Tabla I.2: Medidas de consumo del sistema de aplicación individual del Coastline.

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	2.88	0.987	1.457	0.0007	
10	18.71	1.001	2.088	0.0011	68.52
20	36.32	1.027	2.692	0.0017	62.92
30	53.91	1.056	3.198	0.0023	57.67
40	69.79	1.074	3.642	0.0030	53.80
50	89.09	1.103	4.256	0.0040	54.76

Tabla I.3: Medidas de consumo del sistema de aplicación individual del Hjorth.

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	2.88	0.983	1.479	0.0007	
10	18.65	1.005	2.095	0.0011	66.11
20	36.17	1.030	2.707	0.0017	62.16
30	53.73	1.056	3.198	0.0023	56.79
40	71.21	1.081	3.749	0.0031	55.71
50	88.68	1.106	4.299	0.0040	55.04

Tabla I.4: Medidas de consumo del sistema de aplicación individual del RMS.

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	2.39	0.987	1.482	0.0007	
10	22.16	1.009	2.484	0.0015	108.88
20	44.19	1.038	3.338	0.0025	94.99
30	66.12	1.088	4.306	0.0040	93.87
40	88.04	1.110	4.996	0.0054	86.94

Tabla I.5: Medidas de consumo del sistema de aplicación individual del NLA.

Instancias	Elem. Log. (%)	P^{static} (mW)	P^{total}_{design} (mW)	$\Delta P^{total}_{design}$ (mW)	$P^{dynam}_{alg\_block}$ (μW)
1	2.26	0.987	1.576	0.0007	
10	20.21	1.034	3.280	0.0025	184.14
20	40.10	1.034	5.024	0.0054	179.01
30	60.01	1.063	6.627	0.0093	171.55
40	79.87	1.092	8.203	0.0142	167.24
50	99.55	1.132	9.868	0.0206	166.27

Tabla I.6: Medidas de consumo del sistema de aplicación individual del VRS.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Shriram Raghunathana, Sumeet K. Gupta b, Kaushik Roya Himanshu S. Markandeya, and Pedro P. Irazoqui. A hardware-algorithm co-design approach to optimize seizure detection algorithms for implantable applications. *Journal of Neuroscience Methods*, 2010.

[2] Ivan Osorio, Hitten P. Zaveri, Mark Frei, and Susan Arthurs. *Epilepsy. The Intersection of neurosciences, biology, mathematics, engineering, and physics.* Taylor and Francis Group, LLC, Boca Raton, Florida, 2011.

[3] Felice T Sun and Martha J Morrell. The RNS system: responsive cortical stimulation for the treatment of refractory partial epilepsy. *Expert Review of Medical Devices*, 2014.

[4] William O Tatum IV, Aatif M. Husain, Selim R. Benbadis, and Peter W. Kaplan. *Handbook of EEG Interpretation.* Demos.

[5] N. J. Hill, D. Gupta, P. Brunner, A. Gunduz, M. A. Adamo, A. Ritaccio, and G. Schalk. Recording human electrocorticographic (ECoG) signals for neuroscientific research and real-time functional cortical mapping. *Jove Journal Neuroscience*, 2012.

[6] Kai J. Miller, Dora Hermes, and Nathan P. Staff. The current state of electrocorticography-based brain–computer interfaces. *Journal of Neurosurgery*, 2020.

[7] Robert S. Fisher. Deep brain stimulation of thalamus for epilepsy. *Neurobiology of Disease*, 2023.

[8] Panebianco M, Rigby A, and Marson AG. Vagus nerve stimulation for focal seizures. *Cochrane Database of Systematic Reviews*, 2022.

[9] Yongpei Ma, Chunyu Liu, Maria Sabrina Ma, Yikai Yang, Nhan Duy Truong, Kavitha Kothur, Armin Nikpour, and Omid Kavehei. TSD: Transformers for seizure detection. *bioRxiv*, 2023.

[10] Sun Yulin, Jin Weipeng, Si Xiaopeng, Zhang Xingjian, Cao Jiale, Wang Le, Yin Shaoya, and Ming Dong. Continuous seizure detection based on transformer and long-term iEEG. *IEEE Journal of Biomedical Health Informatics*, 2022.

[11] Brian Litt, Rosana Esteller, Javier Echauz, Maryann D'Alessandro, Rachel Shor, Thomas Henry, Page Pennell, Charles Epstein, Roy Bakay, Marc Dichter, and George Vachtsevanos. Epileptic seizures may begin clinical study hours in advance of clinical onset: A report of five patients. *Neuron*, 2001.

[12] Uisub Shin, Cong Ding, Bingzhao Zhu, Yashwanth Vyza, Alix Trouillet, Emilie C. M. Revol, Stéphanie P. Lacour, and Mahsa Shoaran. NeuralTree: A 256-Channel 0.227 μJ/Class versatile neural activity classification and closed-loop neuromodulation SoC. 2022.

[13] Andrew White, Philip Williams, Damien Ferraro, Suzanne Clark, Shilpa Kadamb, Edward Dudek, and Kevin Staley. Efficient unsupervised algorithms for the detection of seizures in continuous EEG recordings from rats after brain injury. *Journal of Neuroscience Methods*, 2006.

[14] White AM, Williams PA, Ferraro DJ, Clark S, Kadam SD, and Dudek FE. Efficient unsupervised algorithms for the detection of seizures in continuous EEG recordings from rats after brain injury. *Journal of Neuroscience Methods*, 2006.

[15] `https://www.ieeg.org/main.html`. Online; accessed 21 February 2024.

[16] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27, 2006.

[17] `https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=es-419`. Online; accessed 21 February 2024.

[18] J Oliver. *Técnicas de bajo consumo en FPGAs*. PhD thesis, Universidad de la República (Uruguay). Facultad de Ingeniería, 2014. Disponible en `https://hdl.handle.net/20.500.12008/20195`.

[19] Altera. *DE0 User Manual*, 2012.

[20] Altera. *Cyclone III Device Handbook*, 2012. Vol. 2.

[21] `https://github.com/ieeg-portal/ieegpy`. Online; accessed 21 February 2024.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.