

Evaluación del rendimiento de microcontroladores en procesamiento de señales para la implementación de Interfaces Cerebro-Computadora

Alexander Rafael López Zavaleta

Director:

Mg. Bioing. Eduardo Filomena

Evaluadores:

Bioing. Gonzalo Cuenca

Mg. Bioing. Luciano Schiaffino

Mg. Bioing. Yanina Atum

A. ÍNDICE

A. ÍNDICE.....	1
B. OBJETIVOS	4
B.1. Objetivo General.....	4
B.2. Objetivos particulares	4
C. RESUMEN	5
D. INTRODUCCIÓN	6
E. CAPÍTULOS	7
CAPÍTULO 1: MARCO TEÓRICO	7
1.1. Interfaces Cerebro-Computadora.	7
1.1.1. Concepto.....	7
1.1.2. Potencial Evocado Relacionado a un Evento: P300	8
1.1.3. Diagrama de Bloques	10
1.1.3.1. Adquisición de señales.....	10
1.1.3.2. Procesamiento	11
1.1.3.3. Aplicación	11
1.1.3.4. Realimentación	12
1.2. Microcontrolador LPC 4337: Basado en el procesador Cortex M4-F	12
1.2.1. Procesador Cortex-M4F	13
1.2.1.1. Arquitectura ARMv7-M.....	13
1.2.1.2. Tiempo de ejecución	14
1.2.1.3. Herramientas para DSP	15
1.2.1.4. Bytes, media palabra, palabra, palabra doble	18
CAPÍTULO 2: ESPECIFICACIONES Y CONSIDERACIONES.....	19
2.1. El Deletreador de Donchin.	19
2.2. Recursos a evaluar	21
2.2.1. Tiempo de ejecución.	21
2.2.2. Memoria RAM	21
2.2.3. Memoria ROM	23
2.3. Algoritmos a Implementar	23
2.3.1. Promediación coherente.	23
2.3.2. Extractor de Características.....	24
2.3.2.1. Submuestreo Equitemporal.....	24
2.3.2.2. Submuestreo basado en Eliminación Recursiva de Características	24
2.3.3. Clasificador LDA.....	25

2.4. Modificación de parámetros en la implementación	26
2.4.1. 10 Canales vs. 1 canal	26
2.4.2. Longitud de las Señales.....	26
2.4.3. Tipos de datos.....	27
2.3.4. Implementación.....	31
 CAPÍTULO 3: METODOLOGÍA.....	 34
3.1. Generación de herramientas	34
3.1.1. Comunicación uC-PC.....	34
3.1.2. Algoritmos de conversión de datos.....	34
3.2. Método de medición de parámetros.	34
3.2.1. Tiempo de Ejecución.....	34
3.2.1.1. Medición mediante Timer.....	36
3.2.1.2. Medición mediante DWT	37
3.2.1.3. Repetición de las mediciones.....	39
3.2.2. Memoria RAM	40
3.2.3. Memoria ROM	41
3.3. Verificación del procesamiento.....	41
 CAPÍTULO 4: IMPLEMENTACIÓN DE ALGORITMOS	 43
4.1. Herramientas utilizadas.....	43
4.1.1. CMSIS	43
4.1.2. SIMD.....	44
4.1.3. MAC y FPU.....	44
4.2. Algoritmos de verificación del procesamiento.....	45
4.3. Algoritmos de procesamiento	46
4.3.1. Promediación coherente	46
4.3.1.1. Promediación coherente – 2 Señales	46
4.3.1.2. Promediación coherente – 4 Señales	48
4.3.2. Extractor de características	49
4.3.2.1. Submuestreo Equitemporal.....	49
4.3.2.2. Submuestreo ERC	52
4.3.3. Clasificador LDA.....	53
 CAPÍTULO 5: RESULTADOS	 55
5.1. Verificaciones.....	55
5.1.1. Promediación coherente	55
5.1.2. Extractor de Características.....	56
5.1.3. Clasificador LDA.....	57
5.2. Mediciones de parámetros.....	58
5.2.1. Tiempo	58
5.2.1.1. Repetibilidad.....	59
5.2.1.2. Tiempo de ejecución Vs. Longitud de la señal	63
5.2.1.3. Tiempo de ejecución Vs. Tipo de dato	65

5.2.2.	Memoria ROM	70
5.2.2.1.	Promediación Coherente	70
5.2.2.2.	Extractores de características.....	71
5.2.2.3.	Clasificador LDA.....	72
5.2.2.4.	Procesamiento total: Condiciones límites	72
5.2.3.	Memoria RAM	74
5.2.3.1.	Promediación Coherente	74
5.2.3.2.	Submuestreo	74
5.2.3.3.	Clasificador LDA.....	75
5.2.3.4.	Procesamiento total: Condiciones límites	75
CAPÍTULO 6: ANÁLISIS ECONÓMICO		76
6.1. Introducción.....		76
6.2. Proyecto propuesto.....		76
6.2.1.	Objetivo	76
6.2.2.	Resumen.....	76
6.2.3.	Alcances y Limitaciones	77
6.2.4.	Duración.....	77
6.2.6.	Equipo de trabajo	77
6.2.7.	Infraestructura	77
6.2.8.	Impacto	77
6.3. Financiamiento		78
6.3.1. Presupuesto		78
6.4. Análisis de costos.....		79
F. CONCLUSIONES		81
G. BIBLIOGRAFÍA		83
H. ANEXOS		85
Anexo N°1: “Tiempos de ejecución de algoritmos en función de la longitud de señal y el tipo de dato”		85
Anexo N°2: “Presupuestos y precios de lista para el análisis económico”		91
Anexo N°3: “Estipendio de Becas doctorales del CONICET según zona de trabajo”		95

B. OBJETIVOS

B.1. Objetivo General

- Evaluar el rendimiento de microcontroladores en el procesamiento digital de señales para la implementación de Interfaces Cerebro-Computadora en tiempo real basadas en el paradigma de potenciales evocados visuales P300.

B.2. Objetivos particulares

- Integrar los conocimientos de la carrera de Bioingeniería.
- Definir e implementar una metodología para la evaluación del rendimiento de los microcontroladores.
- Determinar criterios para la elección de algoritmos de procesamiento y sus modificaciones a ejecutar.

C. RESUMEN

En el primer capítulo, la primera sección trata sobre Interfaces Cerebro-Computadora, comprende definiciones de estos sistemas, así como su funcionamiento por bloques. Esto nos provee un panorama de los requerimientos de la aplicación buscada. En la segunda sección se verá el microcontrolador LPC 4337, basado en un procesador Cortex-M4F, que es la arquitectura elegida para nuestra evaluación. Se caracterizan herramientas que posee dicho procesador y que lo hacen interesante para llevar a cabo la evaluación propuesta, como SIMD, FPU, MAC, pipeline, arquitectura Harvard, entre otros.

En el segundo capítulo se presenta la información necesaria para contextualizar el trabajo. Se explica el deletreador de Donchin como aplicación ejemplo de una Interfaz Cerebro-Computadora; los algoritmos que se implementan en la evaluación y su función en el procesamiento de señales.

Por otro lado, se especifican los recursos en función de los cuales se evalúa el rendimiento del procesador (tiempo de ejecución, memoria RAM y memoria ROM); así como también parámetros claves que impactan en el rendimiento del procesador, como el tipo de datos utilizado y la longitud de la señal procesada, y se establecen variaciones de éstos para su evaluación.

En el tercer capítulo se aborda la metodología empleada para llevar a cabo la evaluación propuesta. Se explicitan las herramientas generadas que fueron necesarias para ello; los métodos de evaluación de parámetros desarrollados o investigados para llevar a cabo la medición de recursos en uso y la necesidad de emplear una verificación del procesamiento implementado.

En el cuarto capítulo se describen las herramientas utilizadas para la implementación de los algoritmos sobre el microprocesador. Se corrobora el uso de algunas herramientas que ofrece el procesador Cortex M4-F para la ejecución de procesamiento digital de señales (MAC, SIMD y FPU). Por otro lado, se detalla cómo se realizó la implementación de los algoritmos de verificación de los algoritmos de procesamiento diseñados.

En el quinto capítulo se reportan los resultados de las mediciones de los recursos utilizados al ejecutar el procesamiento establecido. Se muestran tanto los resultados del tiempo de ejecución en función de la longitud de la señal, como en función del tipo de dato utilizado. También se exponen resultados respecto al empleo de la memoria RAM y ROM. En cada uno de estos parámetros se brinda información de los casos límites de consumo (mínimo y máximo) con el objetivo de tener un panorama general del uso de recursos ante una ejecución del procesamiento completo (promediación coherente, extracción de características y clasificación).

En el sexto capítulo se realiza un análisis económico de un proyecto que toma como punto de partida los resultados y conclusiones alcanzados en este trabajo y tiene como objetivo implementar y optimizar una Interfaz cerebro-computadora para la detección de potenciales P300 en un contexto académico. Para ello se caracteriza el proyecto mencionado, se elige una línea de financiamiento y se estiman los recursos necesarios para llevarlo a cabo.

Las conclusiones de este trabajo brindan un punto de vista optimista para la implementación de estos sistemas en dispositivos dedicados, descartando que la etapa de procesamiento digital de señales utilizadas en este trabajo sea una limitante de tal implementación sobre estas plataformas.

D. INTRODUCCIÓN

La investigación en el área de las Interfaces Cerebro-Computadora (ICC) ha aumentado enormemente en los últimos tiempos [1]. Esto se ve reflejado en la cantidad de artículos científicos publicados acerca de funciones cerebrales, optimización en la adquisición de señales biológicas y procesamiento de señales para la implementación de estos sistemas.

Las ICC son sistemas que permiten comunicación sin movimiento alguno, por ello pueden ser la única alternativa de comunicación para los usuarios con discapacidades neuromusculares graves, que no pueden hablar o usar teclados, cursores u otras interfaces tradicionales [3]. Es debido a esto, que en el campo de la Bioingeniería, los avances se encuentran dirigidos al desarrollo de tecnologías de comunicación aumentativa¹ [2], control de prótesis, entre otros.

Por otro lado, las ICC también tienen potencial aplicación con otros fines (bélicos, entretenimiento, entre otros), por lo que se ha generado interés clínico, científico y comercial para el desarrollo de estos sistemas en dispositivos dedicados² [3].

Para una eficaz implementación de una ICC, se debe conocer las condiciones que el propósito de la ICC impone, para poder satisfacer las necesidades de la misma, y los recursos que éstos indefectiblemente consumen. Las condiciones impuestas por la aplicación de la ICC generalmente son el tiempo de respuesta del sistema y la tasa de efectividad de la misma para la tarea planteada.

En la fase de investigación de los algoritmos de procesamiento de señales de las ICC, el objetivo principal es conseguir algoritmos eficaces. Generalmente en esta instancia no se evalúa la eficiencia de los algoritmos, ya que ésta es dependiente del hardware sobre el cual se ejecuta el procesamiento.

Por lo tanto, como paso previo a la implementación de ICC en dispositivos dedicados, se debe conocer el rendimiento de estas unidades ante esta tarea. Estos dispositivos ofrecen prestaciones, así como también marcan limitaciones tecnológicas, para llevar a cabo el procesamiento requerido por una ICC.

Es por ello que el presente trabajo propone realizar una evaluación del uso de recursos para la implementación de ICC en tiempo real soportados en dispositivos dedicados basados en la arquitectura Cortex-M4F de ARM. Con esto se brinda un panorama general del desempeño de estos microprocesadores en función de parámetros que se establecen como claves para implementar este sistema: Tiempo de ejecución de procesamiento, Memoria ROM y Memoria RAM.

¹ La comunicación aumentativa involucra formas de expresión distintas al lenguaje hablado, que tienen como objetivo aumentar y/o compensar las dificultades de comunicación de muchas personas con diversas discapacidades.

² Un dispositivo dedicado, o sistema dedicado, es un sistema de cómputo dedicado exclusivamente a una tarea.

E. CAPÍTULOS

Capítulo 1: Marco Teórico

Este capítulo comprende dos tópicos importantes para el desarrollo de este trabajo: Las Interfaces Cerebro-Computadora y el microcontrolador LPC 4337 basado en el procesador Cortex M4-F.

1.1. Interfaces Cerebro-Computadora.

1.1.1. Concepto.

Una Interfaz Cerebro-Computadora (ICC o BCI de sus siglas en inglés *Brain-Computer Interface*) es un sistema que adquiere, procesa e interpreta señales cerebrales para establecer una comunicación usuario-máquina. Para llevar a cabo esta comunicación, estos sistemas no dependen de salidas normales, tales como las vías nerviosas periféricas y/o músculos [4], sino que se vale de otro tipo de salidas para crear una nueva modalidad de interacción hombre-máquina [5].

Según la definición habitualmente adoptada, cualquier ICC debe [1]:

- Basarse en mediciones directas de la actividad cerebral.
- Proporcionar retroalimentación para el usuario.
- Operar en línea (*on-line*).
- Basarse en el control intencional (es decir, los usuarios deben optar por realizar una tarea mental para enviar un mensaje o comando cada vez que quieren utilizar la ICC).

Una definición más reciente describe una ICC como un sistema que mide la actividad del Sistema Nervioso Central (SNC) y la convierte en una salida artificial que reemplaza, realza, restaura, acentúa, suple, o mejora la salida natural del SNC. Esta definición incluye ICC que no requieren el control intencional, que a veces son referidos como ICC pasivos [1]. La Figura 1.1 ilustra el principio básico de una ICC de acuerdo con la definición expuesta previamente.

Hoy en día, las ICC determinan la intención del usuario basándose en una variedad de señales electrofisiológicas diferentes como: potenciales corticales lentos, potenciales evocados P300, ritmos μ y β registradas desde el cuero cabelludo, y actividad neuronal cortical registrada por electrodos sobre la superficie del cuero cabelludo o implantados en una determinada área nerviosa. Tales señales se traducen en tiempo real en comandos que operan un determinado dispositivo [3].

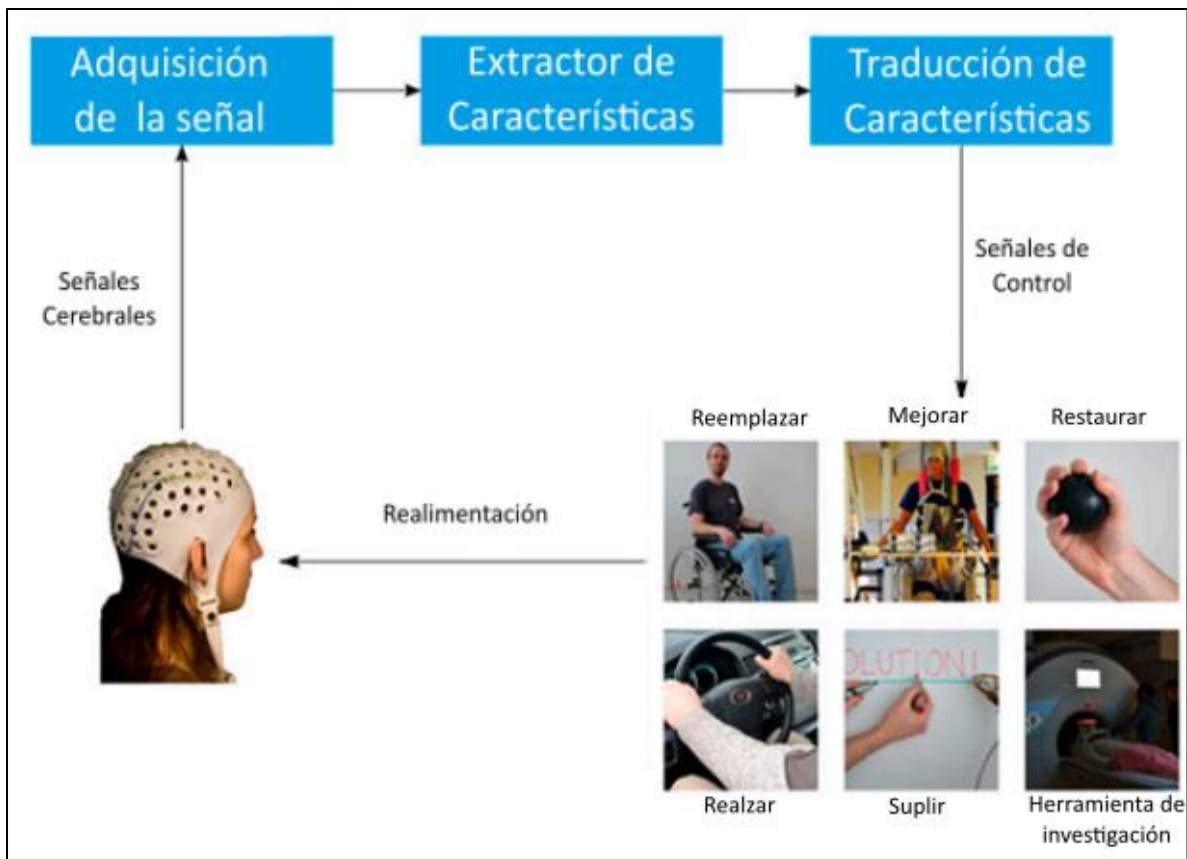


Figura 1.1. Principio básico de funcionamiento de una ICC. (Imagen adaptada de [1]).

1.1.2. Potencial Evocado Relacionado a un Evento: P300

Este trabajo implementa algoritmos para la detección del potencial P300 en la etapa de procesamiento de la ICC, lo cual es la base para establecer la comunicación entre usuario-máquina.

Muchos estudios han demostrado que, cuando a un sujeto se le asigna una tarea que requiere clasificar un ítem de una serie de éstos en dos categorías, donde uno de ellos ocurre de manera poco frecuente (denominados “ítems raros”), la visualización por parte del usuario de estos estímulos evocan un Potencial cerebral Relacionado a este Evento (PRE o ERP de sus siglas en inglés Event-Related brain Potential), con un componente positivo marcado con una latencia de 300 mseg. Este potencial es denominado P300 [6].

Ha sido ampliamente documentado que la amplitud del P300 varía directamente con la relevancia del evento evocado y directamente con la probabilidad del estímulo. Este paradigma experimental se ha denominado “paradigma oddball”.

Se destaca que, no es necesario que el sujeto reporte la ocurrencia del evento objetivo de alguna manera (por ejemplo: presionando un botón). En otras palabras, la señal P300 es evocado por el reconocimiento del sujeto del evento raro sin ningún tipo de comunicación verbal o motora. Esta característica del potencial P300 es la que brinda la posibilidad de desarrollar un nuevo tipo de comunicación usuario-máquina por medio del paradigma oddball [6].

Existen diferentes variantes de la estrategia que emplea este principio. La Figura 1.2 ilustra esquemáticamente estas variantes: En la tarea de solo un estímulo, el estímulo

objetivo se presenta con poca frecuencia sin otros estímulos presentes en el experimento (Figura 1.2 - Arriba). En el tradicional paradigma oddball de dos tipos de estímulos, un estímulo objetivo infrecuente se produce en un contexto de estímulos frecuentes estándar (Figura 1.2 - Centro). En el oddball de tres tipos de estímulos, el estímulo objetivo se presenta con poca frecuencia, estos estímulos se encuentran entre estímulos normales con frecuencia estándar y los estímulos distractores ocurren también con poca frecuencia (Figura 1.2 - Abajo).

En cada caso, el estímulo objetivo provoca como respuesta un gran potencial positivo que aumenta en amplitud desde el electrodo frontal al parietal y tiene una latencia de pico de alrededor de 300 [ms] para estímulos auditivos y 400 [ms] para los estímulos visuales en adultos jóvenes [7].

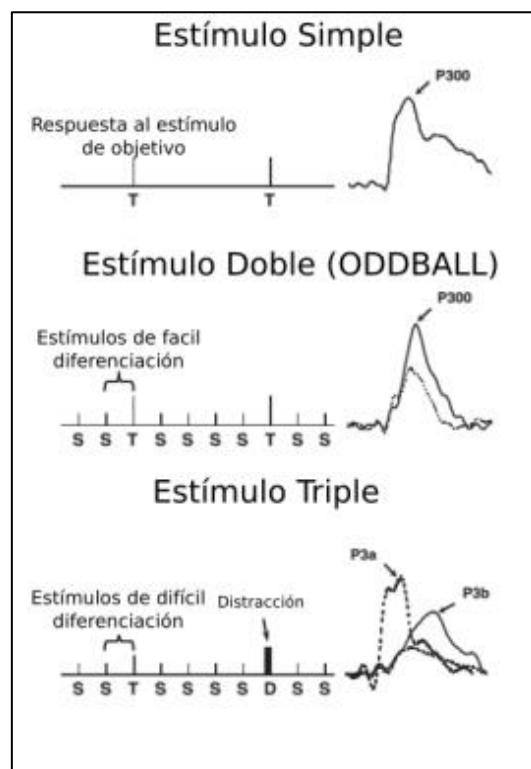


Figura. 1.2. Ilustración esquemática de las variantes del paradigma oddball de un solo estímulo (arriba), de 2 estímulos (medio) y de 3 estímulos (abajo) con el Potencial Relacionado a un Evento (PRE) de cada estímulo de cada tarea presentada a la derecha.

En la Figura 1.2, la tarea de un solo estímulo presenta un objetivo infrecuente (T: *Target*) en ausencia de cualquier otro estímulo. En la tarea de 2 estímulos diferentes, éstos se presentan en una secuencia aleatoria, ocurriendo de manera mucho menos frecuente el estímulo objetivo que la otra (T: *Target*; S: *Standart*). La tarea de 3 estímulos es similar a la anterior con un estímulo adicional, el distractor (D: *Distracter*), que también ocurre infrecuentemente.

En cada tarea, el sujeto es instruido para que responda solo al objetivo y que se abstengan de responder ante ningún otro estímulo. El estímulo distractor de la tarea con 3 estímulos provoca un P3a y el estímulo objetivo provoca un P3b (P300) [7].

1.1.3. Diagrama de Bloques

En general una ICC, se integra por los siguientes bloques: Adquisición de señales, Procesamiento digital de señales, Aplicación y retroalimentación del sistema. En la Figura 1.3 se presenta un esquema del diagrama de bloques de una ICC.

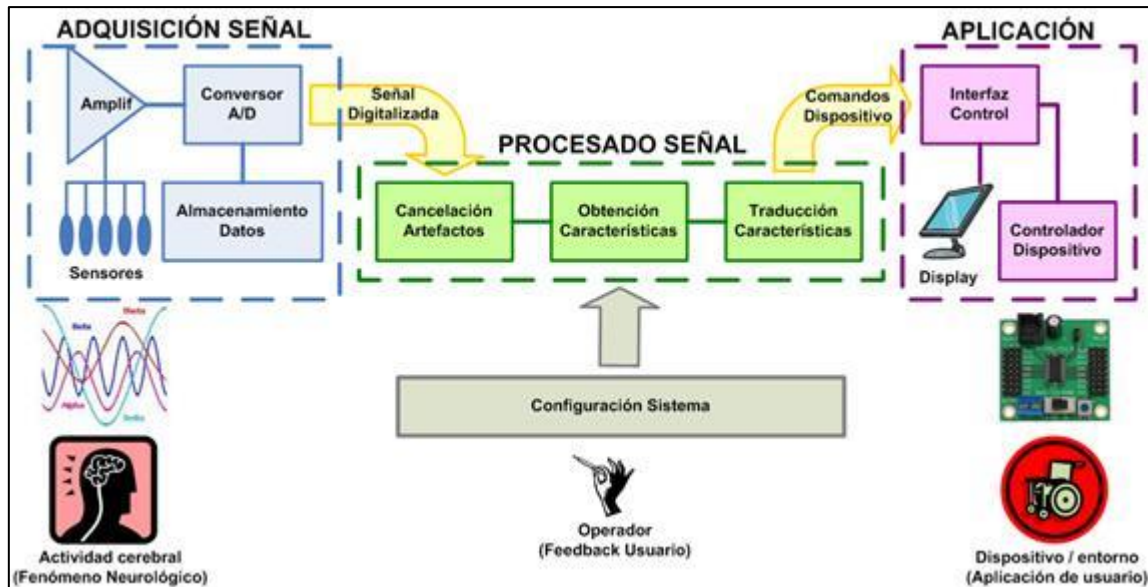


Fig. 1.3. Diagrama en bloques de un ICC. Imagen extraída de [8]

1.1.3.1. Adquisición de señales

Las señales cerebrales empleadas para establecer una comunicación mediante la detección del P300 se obtienen registrando señales electroencefalográficas (EEG), dado que varias de sus características y formas de onda se relacionan con procesos cerebrales bien conocidos que pueden ser inducidos mediante estímulos visuales, sonoros o somatosensitivos [5] [9].

El EEG se clasifica según su método de adquisición en: invasivo, si los electrodos atraviesan o se encuentran dentro del cuero cabelludo; o no invasivo, si se utilizan electrodos superficiales montados sobre el cuero cabelludo [5] [9].

Usualmente se utilizan electrodos superficiales para el registro de EEG cuando se busca implementar una ICC. El ancho banda típico de esta señal se encuentra en el rango de 0 a 100 Hz, por lo que la frecuencia de muestreo generalmente utilizada para su registro es de 256 Hz [9]. La amplitud típica del EEG registrado superficialmente se encuentra en el rango de 100 μ V, por lo que la resolución mínima recomendada para su correcto registro es de 1 μ V, lo cual no es fácil de lograr en señales que usualmente se encuentran inmersas en mucho ruido, fundamentalmente proveniente de las redes del tendido eléctrico [10].

A su vez, la interfaz entre los electrodos de registro y la piel del paciente agrega dos inconvenientes más: primero, la impedancia de contacto es de valor elevado y muy variable; segundo, se genera en la interfaz, un potencial de origen electroquímico, cuyo valor supera en varios ordenes de magnitud a la señal a registrar, pudiendo llegar a ± 300 mV [10].

Es por ello que se necesita una etapa de adquisición robusta que contemple estos inconvenientes propios del registro de la señal de EEG.

Por otro lado, los amplificadores empleados en la adquisición de señales eléctricas biológicas, denominados de amplificadores de biopotenciales, acondicionan la señal a un nivel adecuado para las etapas posteriores de digitalización y almacenamiento. También rechazan interferencias electromagnéticas, principalmente la producida por la red de distribución eléctrica. Para esto, es esencial una alta Relación de Rechazo de Modo Común (RRMC), bajo nivel de ruido y un nivel de ganancia adecuado [11]. Estas funciones son indispensables para llevar a cabo esta etapa del sistema, es por esto que las características de este tipo de amplificadores son esenciales.

1.1.3.2. Procesamiento

La etapa de procesamiento consiste en una serie de algoritmos que transforman las señales de entrada en una salida acorde para la aplicación deseada.

En este trabajo, la etapa de procesamiento comprende algoritmos que toman como entrada señales de EEG, que son vistas un conjunto de características en el dominio temporal que reflejan la actividad cerebral, y las convierten en una salida discreta de tipo Si/No al detectar (o no) la presencia del potencial P300.

El procesamiento implementado consiste en tres etapas: Promediación Coherente, Extractores de Características y Clasificadores lineales.

En la promediación coherente se reduce el ruido aleatorio inmerso en la señal; los extractores de características disminuyen la cantidad de datos que entran al clasificador, y éste último se encarga de decidir la presencia o no del potencial P300 en la señal. Se explica más detalladamente cada procesamiento en sección 2.3.

En la actualidad, existe una amplia diversidad de tipos procesamiento de señales, debido en parte a la diversidad objetivos de sus aplicaciones en el mundo real. Sin embargo, en todos los casos la meta es maximizar la eficacia y la practicidad para las aplicaciones elegidas [4].

1.1.3.3. Aplicación

Las salidas de las ICC pueden ser de distinta índole, y dependen del paradigma empleado para implementación del sistema. Como por ejemplo estas aplicaciones pueden ser: el movimiento de un cursor, selección de íconos o letras, o el control de una neuroprótesis. [4]

Las ICC pueden ser usadas para los siguientes objetivos (entre otros) [1]:

1. Reemplazar la salida del SNC el cual ha sido perdido debido a una lesión o enfermedad. Por ejemplo: Sistemas de delecteo, sintetizador de voz o control de silla de ruedas motorizada.
2. Restaurar la pérdida natural de la salida del SNC. Como por ejemplo: En la Estimulación Eléctrica Funcional de músculos en una persona paralizada y estimulación de nervios periféricos para restaurar la función vesical.
3. Realzar la salida natural del SNC. Por ejemplo: la monitorización de la actividad cerebral durante prolongadas demandas de tareas tales como majear un automóvil y detectar lapsos de atención, el cual alerta a la persona y restaura la atención.
4. Suplir las salidas naturales del SNC. Como por ejemplo el de proveer un brazo robótico a una persona amputada.

5. Mejorar la salida natural del SNC. Por ejemplo: la utilización de una ICC en rehabilitación de ACV que detecte y mejore las señales de un área cortical dañada para estimular músculos o una órtesis para mejorar movimientos.
6. Pueden ser usados como herramienta de Investigación para estudiar las funciones del SNC en experimentos clínicos y no-clínicos.

Las aplicaciones mencionadas anteriormente tienen contexto biomédico. Sin embargo, las ICC se pueden adaptar utilizando los mismos principios en aplicaciones en otros campos, como en la industria de los videojuegos o desarrollos con fines bélicos.

En este trabajo se emplea procesamiento desarrollado en base a la detección del potencial P300 para la implementación de un deletreador de Donchin, explicado en la sección 2.1.

1.1.3.4. Realimentación

Las ICC también proveen una realimentación entre el dispositivo y el usuario que se puede usar tanto para ratificar como para optimizar la comunicación deseada.

Cabe mencionar que las aplicaciones de las ICC deberían ser optimizadas para cada usuario o para cada grupo de usuarios. Al mismo tiempo, el proceso de optimización debería ser tan objetivo como estandarizado posible. Los desarrolladores de ICC requieren de una fuerte cooperación interdisciplinaria, entre neurocientíficos, ingenieros, kinesiólogos, programadores y especialistas en rehabilitación [4].

1.2. Microcontrolador LPC 4337: Basado en el procesador Cortex M4-F

En este trabajo se utilizó como plataforma de desarrollo la EDU-CIAA³ (Figura 1.4) para implementar los algoritmos de procesamiento especificados en la sección 2.3. Esta plataforma se basa en el chip LPC4337 JDB144, cuyo fabricante es NXP Semiconductors.



Figura 1.4. Imagen de la placa de desarrollo EDU-CIAA.

El microcontrolador LPC4337 posee un procesador ARM Cortex-M4 y un ARM Cortex-M0 como coprocesador. Posee además, periféricos configurables tales como el State Configurable Timer (SCT) y el Serial General Purpose Entrada Output (SGPIO) interface, 2 controladores USB de alta velocidad, Ethernet, un controlador de memoria externa y múltiples periféricos analógicos y digitales.

El microcontrolador LPC4337 opera a frecuencias del CPU que llegan hasta 204 MHz, posee 136 KB de memoria RAM para uso de datos y ejecución de código, y 1024 KB de memoria flash en doble banco de memoria con acelerador flash [12].

³ EDU-CIAA, es la versión académica de la CIAA (Computadora Industrial Abierta Argentina), para mayor información visite la página <http://www.proyecto-ciaa.com.ar/>

Se optó por realizar el procesamiento para la implementación de una ICC en un procesador con arquitectura ARM Cortex-M4 ya que presenta características que lo convierten en una opción interesante para ello.

1.2.1. Procesador Cortex-M4F

El procesador Cortex-M4 es un procesador de 32 bits de alto rendimiento, de arquitectura ARMv7, descrito brevemente en la siguiente sección. Este procesador ofrece beneficios significativos para el procesamiento digital de señales (DSP, del inglés *Digital Signal Processing*), incluyendo [13]:

- Multiplicador por hardware en un ciclo de clock
- Divisor por hardware
- Extensiones DSP: *Single Instruction-Multiple Data* (SIMD), hardware que posibilita la operación aritmética de registros en simultáneo, y *Float Point Unit* (FPU), hardware especializado para el procesamiento de datos tipo punto flotante.
- Instrucciones de uso habitual en DSP, tales como: aritmética de saturación e instrucciones MAC (*Multiply and Accumulate*) en un solo ciclo [13].

El procesador también posee las siguientes características que resultan útiles en la implementación de sistemas dedicados, entre otras:

- Emplea un set de instrucciones compacto, lo que implica un costo menor de memoria de código para realizar una tarea, impactando en el costo de las memorias y su consumo eléctrico.
- Posee un set de instrucciones potente y simple. Por lo tanto, los compiladores de lenguajes de alto nivel generan código más compacto, reforzando el punto anterior.
- Presenta un gran número de instrucciones de un solo ciclo, mejorando la velocidad de procesamiento (o performance).
- Posee una pipeline de 3 etapas compacta, simple y eficiente, que logra una ejecución sostenida a alta velocidad.
- Arquitectura de bus Harvard, el cual separa la memoria de programa y la memoria de datos para poder operar simultáneamente sobre éstos.
- Acceso a registros alineados y no alineados, permitiendo almacenar y acceder datos en cualquier posición de memoria.
- Presenta un bajo costo en relación a su performance.
- Set de instrucciones pensado en facilitar la compilación de lenguajes de programación como el C.

1.2.1.1. Arquitectura ARMv7-M

La arquitectura ARMv7-M posee un diseño pensado en una operación altamente determinística, con mínima latencia de interrupciones. Esto implica contar con una pipeline mínima (véase Pipelining en la sección 1.2.1.2) y prescindir del uso de memorias caches [13].

Esta arquitectura incorpora como extensión opcional la FPU, que es un coprocesador que provee soporte para números de precisión simple en punto flotante. De acuerdo a lo especificado en la documentación técnica, cuando el procesador Cortex-M4 posee FPU, el procesador se denomina Cortex-M4F [13].

1.2.1.2. Tiempo de ejecución

Las tareas que debe llevar a cabo el microcontrolador, en términos temporales, están sujetas a restricciones tecnológicas. Sin embargo, también emplean componentes de hardware dedicados para mejorar el rendimiento del procesador. A continuación se describen factores que influyen en el tiempo de ejecución de tareas llevadas a cabo por el procesador ARM Cortex-M4F.

1.2.1.2.1. Tiempos de acceso

La suma de los tiempos involucrados en la decodificación de la información en los buses del microprocesador (retardo de propagación, control, operación,) se denomina “tiempo de acceso”. El tiempo de acceso del microprocesador en un ciclo de lectura o escritura debe ser mayor que el que necesita la memoria para poner o poder leer los datos requeridos en o del bus, respectivamente. Si esto no es así, se deben insertar ciclos de espera, denominados “*wait-states*”.

Los microprocesadores modernos, están desacoplados de la memoria o periféricos mediante un bus intermedio. En este caso, el concepto de tiempo de acceso sigue siendo el mismo, sólo que quien lidia con el tiempo de acceso de la memoria es el controlador de dicho bus (*bus controller*), mientras que el microprocesador sólo debe respetar el protocolo impuesto por este último [13].

1.2.1.2.2. Buses internos y timing

Dada la diferencia considerable de velocidades entre el procesador y las memorias, posiblemente también con los periféricos, y la creciente inclinación a emplear bloques de diversa propiedad intelectual que se interconectan entre sí, los procesadores modernos ya no utilizan un esquema simple de un bus acoplado directamente al procesador (que podemos denominar “esquema tradicional”), como se mencionó en la sección anterior. En su lugar, el procesador utiliza un tipo de bus estándar para interconectar el procesador y sus periféricos.

En síntesis, en un microprocesador de los años '80, con una arquitectura que llamamos tradicional, contamos con la certeza de que cada operación se realiza en un número específico de ciclos de clock. Así, sabemos que por ejemplo la operación sobre un pin de entrada/salida demora cuatro ciclos de clock, lo que en un micro de 40MHz son 100 ns. Otros microprocesadores podrán emplear más o menos ciclos, pero sabemos que siempre serán la misma cantidad y la sincronización (*timing*) es determinística. Lo mismo ocurre para los accesos a memoria, ya sean datos o instrucciones, dado que la memoria está conectada al bus del procesador y podrá tener a lo sumo algunos ciclos de espera (“*wait-states*”).

Sin embargo, en un procesador moderno de 32-bits, tenemos varios buses intermedios, y el timing se va haciendo menos determinístico y predecible, dado que debemos analizar qué está sucediendo en esos buses, pues otros periféricos pueden estar utilizándolos [13].

1.2.1.2.3. Pipelining

Un microprocesador realiza, al menos tres operaciones elementales con las instrucciones: leer, decodificar y ejecutar. Un procesador diseñado de manera que una parte del hardware lee, otra decodifica y otra ejecuta las instrucciones, permite que las operaciones, puedan realizarse en paralelo. El hardware diseñado para realizar estas operaciones denomina “pipeline”.

En la Figura 1.5 se puede apreciar el funcionamiento temporal de una pipeline de 3 etapas. En el tiempo t se lee la primera instrucción, al ser la primera en la pipeline, no se decodifica ni se ejecuta ninguna otra. En $t+1$, se lee la siguiente instrucción y se decodifica la instrucción leída en t . En $t+2$ Se ejecuta la instrucción decodificada en t , se decodifica la instrucción leída en $t+1$ y se lee una nueva instrucción. A partir de este punto, la pipeline se considera “llena” y en esta situación, se opera sobre 3 instrucciones distintas de manera simultánea.

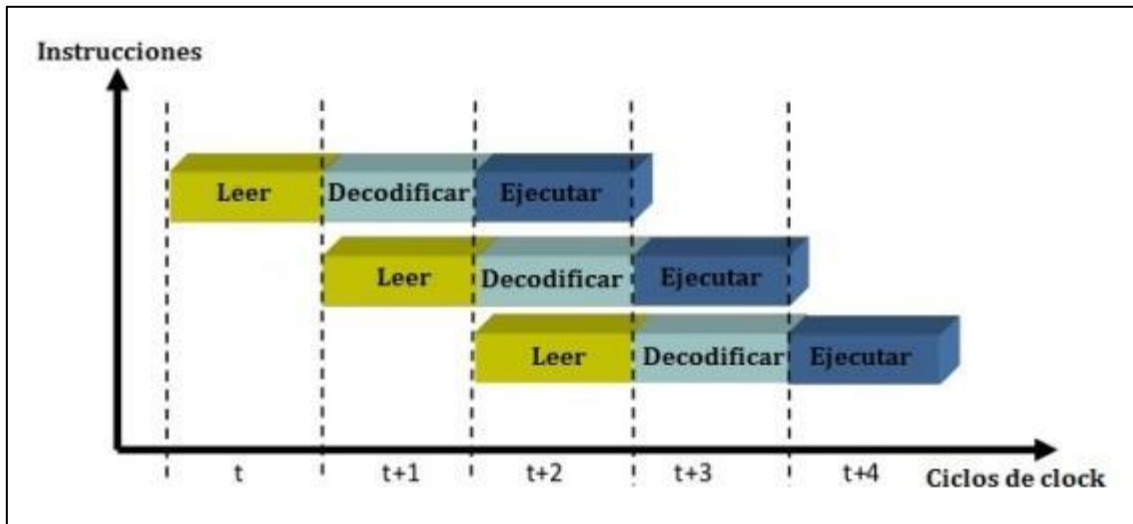


Figura 1.5. Diagrama temporal de funcionamiento de una pipeline de 3 etapas.

Cabe mencionar que, el tiempo de ejecución de cada instrucción es el mismo, este tiempo no se ha mejorado mediante esta estrategia. Lo que se ha mejorado es la cantidad de instrucciones ejecutadas en un mismo tiempo dado. Esto será así siempre y cuando la pipeline funcione sin interrupciones.

Sin embargo, al momento de encontrarse con un salto condicional, el procesador sólo tiene en la pipeline las instrucciones siguientes en el orden en que se encuentran en la memoria; por lo que, si el salto se ejecuta, se ve obligado a descartar esas instrucciones e ir a procesar las nuevas, lo cual emplea un tiempo extra. Es decir, debe purgar y recargar la pipeline.

Una pipeline más larga, entonces trae algunos beneficios, pero también resulta perjudicial en algunos casos. Existen algunas formas de optimizar este tipo de problemas, por ejemplo, los procesadores de alta performance más modernos utilizan algoritmos especulativos y/o de predicción, de modo de poder minimizar el impacto de los saltos.

La pipeline que presenta el procesador Cortex-M4F presenta estas 3 etapas: Lectura, Decodificación y Ejecución de la instrucción [13].

1.2.1.3. Herramientas para DSP

El procesador Cortex-M4F presenta diversas herramientas, tanto en el set de instrucciones como en el hardware disponible, que facilitan el procesamiento de datos. A continuación se detalla el funcionamiento general estas herramientas.

1.2.1.3.1. MAC (Multiply ACcumulate)

Es muy común en algoritmos DSP el multiplicar una serie de muestras por un grupo de coeficientes y sumar resultados parciales:

$$y[i] = \sum_{k=0}^{N-1} x[i-k] \times a_k$$

Este tipo de operación no es otra cosa que la acumulación de sucesivas multiplicaciones, es decir, un bucle donde se repite la operación:

$$y = y + x \times a$$

Es decir, una multiplicación y la acumulación de los resultados. Este tipo de operaciones son útiles para algoritmos de filtrado (como los filtros tipo FIR o IIR), correlación producto punto e incluso en aplicaciones como la FFT (*Fast Fourier Transform*) [25].

Este tipo de operaciones se verán aceleradas notablemente si se dispone de un hardware eficiente que realice la operación elemental de forma rápida. Este hardware es denominado MAC [13].

1.2.1.3.2. Aritmética Saturada

Es de gran aplicación en procesamiento digital de señales, debido a que muchos algoritmos deben imitar el funcionamiento de un sistema analógico, saturándose al salirse de rango en lugar de “reciclarse” numéricamente, como cualquier ALU. Si bien es posible detectar el resultado y saturar los registros manualmente, la existencia de estas instrucciones permite que la operación se realiza en la misma instrucción y en un ciclo [13].

1.2.1.3.3. Barrel shifter

Un Barrel shifter es un circuito secuencial capaz de desplazar una palabra binaria un número determinado de bits en un ciclo de clock. Como tal, posee la particularidad de acelerar notablemente operaciones de normalización y la generación de potencias de 2 [13].

ARM incorpora un barrel shifter en uno de los operandos de la ALU, de modo que en las operaciones de procesamiento es posible afectar a uno de los operandos por un desplazamiento lógico o aritmético, o incluso una rotación. Debido a la estructura interna, el barrel shifter también es aplicable a las operaciones de carga, de modo de soportar el direccionamiento offset con registro escalado [13].

1.2.1.3.4. SIMD Avanzado (Advanced SIMD)

SIMD (*Single Instruction-Multiple Data*) es una extensión del procesador que provee instrucciones que hacen posible operar sobre registros de manera vectorial⁴. La cantidad y variedad de las instrucciones es notable, y operan en paralelo sobre registros que son tratados como vectores.

La Figura 1.6 muestra un ejemplo de una operación de suma mediante SIMD. En este caso se suman de dos vectores de 32-bits ($X[]$ y $Y[]$) conteniendo cuatro variables de 8-bits (para el vector X : $x0$, $x1$, $x2$ y $x3$; Para el vector Y : $y0$, $y1$, $y2$ y $y3$). La velocidad de operación dependerá de la implementación del procesador en particular [13].

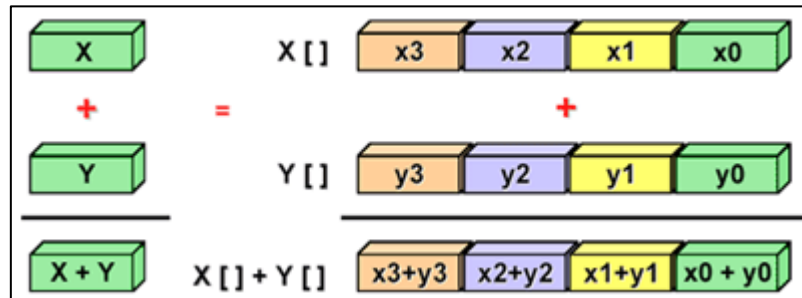


Figura 1.6. Esquema de operación de suma mediante instrucciones SIMD.

La extensión SIMD presenta las siguientes características [14].

- Cálculo simultáneo de 2 operandos de 16-bits o 4 operandos de 8-bits.
- Doble multiplicación/suma/resta de datos de 16-bits utilizando MAC
- Se realiza el procesamiento en paralelo con un incremento muy bajo de la energía empleada.

Esta extensión agrega un juego 32 registros de 64-bits, que pueden verse como 16 registros de 128-bits. Cada registro contiene uno o más elementos empaquetados de un determinado tipo que constituye un vector.

La extensión SIMD se integra con la FPU, compartiendo el mismo sistema de registros, pudiendo utilizarse ambos para proveer soporte SIMD en operaciones con números en punto flotante.

1.2.1.3.5. FPU (Float Point Unit)

La FPU es una unidad opcional en un dispositivo Cortex-M4 que provee funciones computacionales para datos de tipo punto flotante compatibles con el estándar ANSI/IEEE 754. Este estándar, especifica formatos y operaciones del tipo de dato punto flotante [13] [14].

La FPU soporta suma, multiplicación, división, multiplicación y acumulación, y operaciones de raíz cuadrada de datos de punto flotante de precisión simple. Para ello, la FPU opera, internamente, con una pipeline de 3 etapas totalmente independiente [13]. También provee conversiones entre datos de punto fijo y punto flotante. La FPU contiene

⁴ Esta notación hace referencia a la capacidad de procesamiento que poseen los “procesadores vectoriales”, por analogía con el procesamiento de los componentes de un vector. Un procesador simple, que opera sobre un dato a la vez, es denominado entonces “procesador escalar” [13].

32 registros de extensión de precisión simple, el cual se puede acceder como 16 registros de doble palabra para operaciones de carga, almacenamiento y movimiento [15].

La FPU provee entonces más de treinta instrucciones adicionales para soportar números de punto flotante, entre las cuales encontramos suma y multiplicación en un ciclo de clock, división en catorce ciclos y multiplicación y acumulación en tres ciclos de clock [13].

1.2.1.4. Bytes, media palabra, palabra, palabra doble.

En el contexto de este trabajo, nos referiremos a “palabra” (*word*) a un registro de 32 bits, dado que trabajamos con un procesador de 32 bits. Así como el término media-palabra (*half-word*) corresponde a uno de 16 bits y palabra doble (*double-word*) corresponde a uno de 64 bits.

Capítulo 2: Especificaciones y consideraciones

Como se mencionó en la sección 1.1.3.3, en este trabajo se implementaron algoritmos que conforman la etapa de procesamiento de una ICC. Tales algoritmos se extrajeron de una tesis de Maestría en Ingeniería Biomédica denominada “Técnicas de inteligencia computacional aplicadas a la detección de potenciales relacionados a eventos en interfaces cerebro computadoras”. Su autora, es una investigadora del Laboratorio de Ingeniería en Rehabilitación e Investigaciones Neuromusculares y Sensoriales (LIRINS) de la Universidad Nacional de Entre Ríos.

Estos algoritmos fueron seleccionados, de un conjunto de ellos, dado que presentaban una muy buena relación de efectividad (tasa de aciertos, sensibilidad y especificidad) respecto al costo computacional requerido para su ejecución.

En la instancia de investigación mencionada anteriormente se utilizó una base de datos de señales de EEG registradas de sujetos que empleaban un deletreador de Donchin como aplicación de la ICC.

Cabe mencionar que el procesamiento utilizado tiene como objetivo la detección de potencial P300 generado por estímulos visuales en una señal de EEG, por lo que los resultados obtenidos, son extensibles a cualquier otra aplicación que utilice como herramienta la detección del potencial P300.

2.1. El Deletreador de Donchin.

El deletreador de Donchin es un sistema basado en una matriz de 6x6 elementos que contiene las letras del abecedario y números, como la que se observa en la Figura 2.1. También pueden existir, además de letras y números, otros comandos (por ejemplo: espacio, *backspace*, signos, flechas etc.).

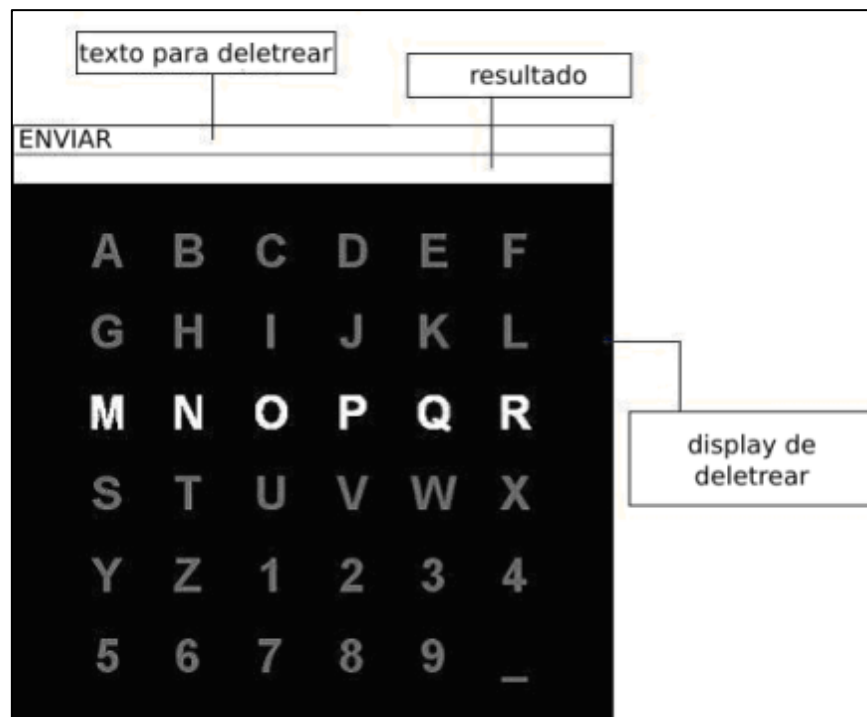


Figura 2.1. Esquema del deletreador de Donchin.

El estímulo visual consiste en iluminar los caracteres de una fila o columna de la matriz por vez en forma aleatoria como se puede observar en la Figura. 2.1. En total, la cantidad de eventos para la identificación del caracter contenido en una celda son 12 (6 filas y 6 columnas) [9].

A medida que el sujeto centra su atención en una celda de dicha matriz, la iluminación de la fila y de la columna a la que pertenece esa celda se convierte en el evento relevante. Sólo dos de estos doce estímulos son los relevantes en relación con la tarea y éstos son percibidos como estímulos raros según el paradigma oddball, por lo que deberían generar un potencial P300.

En este sistema se sincroniza el inicio del estímulo visual con el inicio del registro del EEG, la amplitud del P300 después de cada estímulo visual es evaluado y la celda es identificada como la intersección de la fila y la columna de los P300 detectados.

En la base de datos empleada en la investigación del LIRINS el tiempo de estímulo es de 62,5 mseg y 125 mseg el del intervalo inter-estímulo (ISI por sus siglas en inglés *Inter-Stimulus Interval*) [9].

El ISI se mide desde el comienzo de la intensificación de cada fila o columna y el comienzo de la intensificación de la siguiente fila o columna en un orden aleatorio. Figura 2.2 [6]

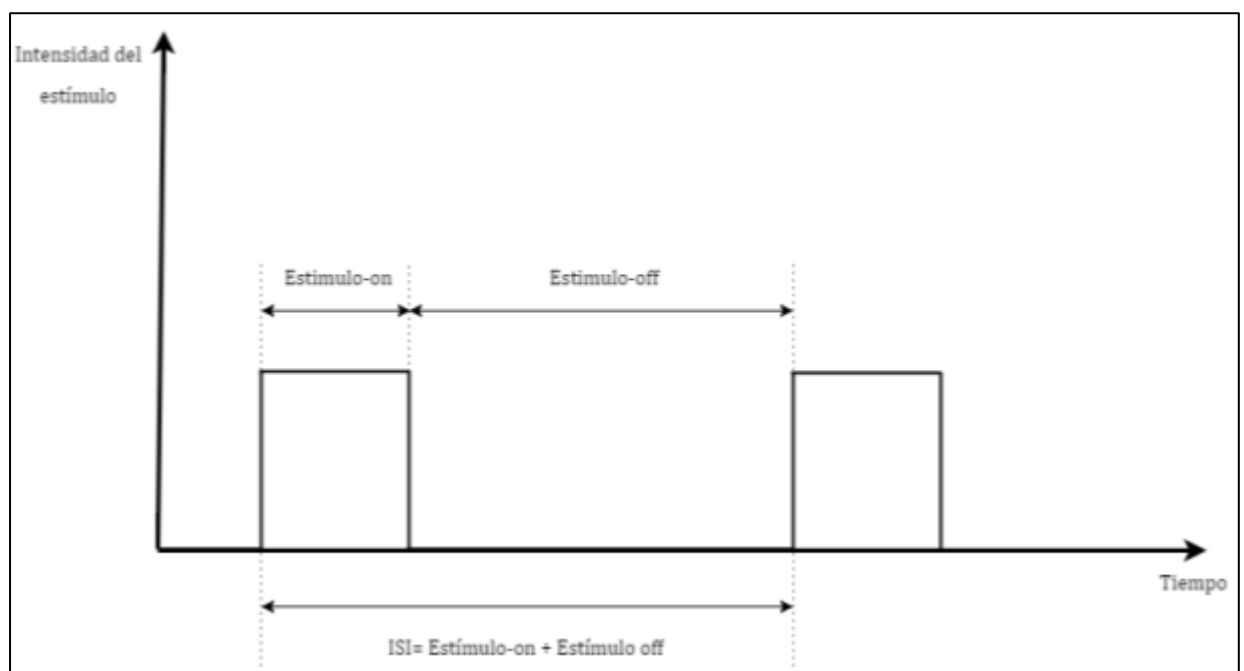


Figura 2.2: Diagrama temporal de la activación de los estímulos y los nombres de los períodos según la bibliografía.

Es importante tener en cuenta, la información temporal de los estímulos visuales, así como también la de la respuesta fisiológica (ver sección 1.1.2) dado que ésta, en cierto modo, nos limita el tiempo de respuesta del sistema, que incluye el tiempo de ejecución del procesamiento evaluado en este trabajo.

2.2. Recursos a evaluar

El rendimiento de un microprocesador ante una determinada tarea lo determina la efectividad de la misma en relación al consumo de recursos disponibles. Se tendrá un mayor rendimiento al tener resultados que se consideren aceptables con un menor uso de recursos.

Asumiendo una eficacia aceptable de un procesamiento determinado, para determinar el rendimiento del procesador se evalúan recursos claves utilizados por éste para una realizar tarea: Tiempo de Ejecución, Memoria RAM y Memoria ROM. Ya que conociendo la utilización de estos recursos se puede inferir acerca del funcionamiento general del procesador al ejecutar una determinada tarea.

2.2.1. Tiempo de ejecución.

La aplicación de la ICC planteada, tiene como requerimiento que sea un sistema en tiempo real. Es por ello que se debe conocer el tiempo de respuesta del sistema. Dado que este sistema debe interactuar con su entorno y responder a los estímulos que proporciona éste dentro de un plazo de tiempo determinado, no basta con que las acciones del sistema sean correctas, sino que además, tienen que ejecutarse dentro de un intervalo de tiempo determinado [17].

2.2.2. Memoria RAM

En un microprocesador la memoria RAM se utiliza para la carga de datos y también para la ejecución de código en ella [13]. Los microcontroladores, tienen una limitada cantidad memoria RAM. Esto se debe principalmente a que para su construcción requiere amplia superficie silicio y esto trae como consecuencia un aumento de los costos en su producción. Además, arreglos grandes de memoria RAM tienen mayores probabilidades de desarrollar fallas simplemente por el hecho de su amplia superficie ocupada [19].

Cuando la cantidad de memoria RAM (*stack*) utilizada por un hilo de ejecución excede la cantidad de memoria disponible, suceden eventos indeseados. Esta condición se conoce como desbordamiento del stack (*stack overflow*) y las consecuencias exactas dependen del entorno operativo y del contexto de ejecución. Existen técnicas de detección de estos desbordamientos en tiempo de ejecución, así como mecanismos de recuperación ante el desbordamiento del stack, pero dejan al programa en condiciones poco óptimas. [19]

El desbordamiento del stack es una gran amenaza para muchas aplicaciones computacionales. Es por esto que es de suma importancia prevenir que las condiciones de desbordamiento se produzcan. Para esto se requiere la evaluar la peor condición de requerimiento de memoria del stack de una aplicación. En cualquier caso, tener metodologías de evaluación de estos requerimientos en forma temprana evita comportamientos erráticos en tiempos de ejecución que traen como consecuencia costos altos e implícitos dado que son descubiertos tarde en el proceso de desarrollo y son difíciles de caracterizar. [19]

Hay 2 grandes enfoques para abordar la evaluar el uso de memoria del stack: Basados en pruebas y basados en análisis estáticos.

Enfoque basados en pruebas (*Testing based*).

El enfoque basado en pruebas consiste en medir la máxima cantidad de memoria utilizada al ejecutar o simularse una determinada aplicación. Una técnica comúnmente utilizada consiste en inicializar la memoria con un patrón repetido y calcular el tamaño de la memoria utilizada al ejecutarse una tarea, verificando cuanta memoria ya no presenta el patrón establecido inicialmente.

Como ventajas, estas mediciones proveen información proveniente de la simulación o ejecución del código. Cada software es probado varias veces a lo largo de su desarrollo, y obtener una realimentación adicional por parte de estas pruebas siempre es valioso.

Sin embargo, los escenarios de prueba en su mayoría solo cubren un subconjunto de posibles contextos de ejecución del programa, y por lo tanto no exponen a la aplicación al peor caso absoluto de consumo del stack. Entonces es crucial probar tantos contextos de ejecución como sean posibles, y esto implica costos significativos porque elaborar y ejecutar un apropiado conjunto de pruebas requiere mucha cantidad de recursos. [19]

Además, los mecanismos requeridos para realizar estas mediciones pueden tener efectos colaterales indeseados, como sutiles influencias de timing desvirtuando la observación o con consecuencias críticas en caso de sistema en tiempo real.

Por último, como se mencionó anteriormente, el enfoque basado en pruebas usualmente necesita ser lo menos intrusivo posible para evitar perturbaciones en la ejecución y perder preciados recursos temporales. Para satisfacer estas restricciones, sólo se permitiría obtener poca información acerca del contexto evaluado, y esto es una restricción desfavorable para la evaluación del consumo de este recurso.

Enfoque basado en análisis estático (*Static Analysis*)

Este enfoque consiste en utilizar herramientas para analizar patrones de consumo de la memoria del stack y computar límites de peores casos antes de la ejecución del programa.

Este tipo de análisis provee resultados precisos sin mucho esfuerzo, rápidamente y de manera temprana en el proceso de desarrollo. Además, los análisis estáticos no tienen efectos colaterales en la ejecución del programa dado que no utilizan recursos en esta instancia, por lo que tienen más lugar para ofrecer tanta información de realimentación como se desea [19].

Compiladores como herramienta para el análisis estático del requerimiento del stack

Existen extensiones del compilador que producen salidas especializadas para propósitos de análisis de requerimiento del stack. Este método aprovecha la capacidad de traducción del compilador a lenguaje de máquina para poder informar acerca de la cantidad de memoria utilizada por el stack.

Sin embargo, existen otras herramientas para este tipo de evaluación, entre ellas "Absint Stack Analyzer y "AVR Simulation and Analysis Framework, entre otras), y éstas trabajan a nivel de analizadores binarios o de assembler. [19]

2.2.3. Memoria ROM

En la memoria ROM se almacenan las series de instrucciones a ejecutar cuando el programa corra en el procesador. En la memoria ROM también se almacenan datos que se deseen mantener de manera permanente [13].

Es de interés conocer cuánta memoria ROM ocupa el código de los procesamiento que se deban implementar en el microprocesador, dado que éste indica cuanto es el espacio de código empleado por el procesador para llevar a cabo el procesamiento de interés, así como también los datos constantes necesarios para ejecutarlos (umbrales, pesos, índices, etc.).

2.3. Algoritmos a Implementar

2.3.1. Promediación coherente.

Es una técnica aplicada para mejorar la relación señal ruido de las señales adquiridas, en la cual se promedian un determinado número de épocas⁵. Este número de épocas elegido (N) es un parámetro de este procesamiento e influye en su resultado como se puede apreciar en la Figura 2.3. Se denomina coherente a esta promediación, dada su característica fundamental de que los inicios de todas las épocas a promediar deben estar sincronizadas de alguna manera, para el caso de los registros de los P300 se las sincroniza con el inicio del estímulo.

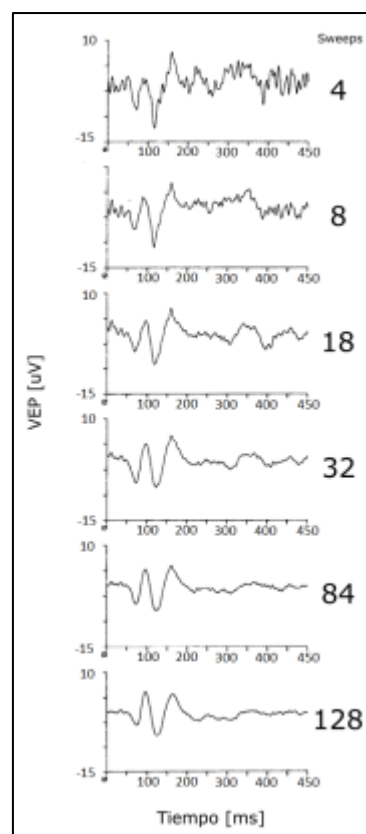


Figura 2.3. Ejemplo de PRE en el análisis de EEG, la reducción del ruido en función de la cantidad de promediaciones efectuadas (*sweeps* o barridos). Imagen extraída de [21].

⁵ Una época, en este contexto, es una señal registrada en un determinado lapso de tiempo el cual contiene información importante para el estudio planteado.

La promediación coherente es un algoritmo útil para la eliminación de ruido de cuantización [20], disminuye la varianza en un factor de N [21] [22], Aumenta la Relación Señal Ruido en un factor de $N^{1/2}$ y a su vez se puede considerar como un equivalente a un filtro [21]

En este trabajo se implementaron algoritmos que realizan promediaciones coherentes de 2 y 4 señales [9].

2.3.2. Extractor de Características

Las señales a procesar pueden contener una gran cantidad de datos (características), cuya dimensión depende de la cantidad de canales de registros considerados y de la frecuencia de muestreo utilizada. Es por esto que esta etapa consiste en elegir un subconjunto relevante de características de la señal para conformar el patrón a clasificar [9].

2.3.2.1. Submuestreo Equitemporal

En el procesamiento de señales, el muestreo de una señal es la transformación de una señal analógica en una señal discreta. Esta señal discretizada tendrá solo valores para instantes de tiempo determinados y equiespaciados establecidos por la frecuencia de muestreo. Esta discretización de la señal se denomina muestreo uniforme y se realiza sin pérdida de información si se respetan las condiciones establecidas por el Teorema del muestreo [9].

El submuestreo equitemporal implica una redefinición de la frecuencia de muestreo, de manera que, de la señal original se toma un subconjunto de valores para representar la señal y estos valores también se toman respetando el intervalo temporal que establece la nueva frecuencia de muestreo [9].

Como se verá más adelante, en la sección 2.4.1.1, se emplearon señales muestreadas a 64 Hz, y los submuestreos implementados son de 32 y 16 Hz.

2.3.2.2. Submuestreo basado en Eliminación Recursiva de Características

El submuestreo basado en Eliminación Recursiva de Características (submuestreo ERC), es la denominación que se emplea para implementar un muestreo de la señal que elige un subconjunto de características basándose en conocer cuáles de éstas son más importantes para su posterior clasificación. El criterio de selección de características se estableció en la etapa de investigación previa utilizando la técnica de ERC

La técnica ERC pertenece a un grupo de algoritmos de extracción de características denominado “algoritmos integrados”, donde la selección de características y el aprendizaje no pueden estar separadas, como se representa en la Figura 2.4. Los algoritmos integrados realizan la selección de características durante el entrenamiento del clasificador y generalmente son específicos para cada tipo de máquina de aprendizaje. Por otro lado, la ERC es un tipo de algoritmo de eliminación hacia atrás, en los cuales estos métodos comienzan incluyendo todas las características y de forma iterativa remueven una o varias características de la señal.

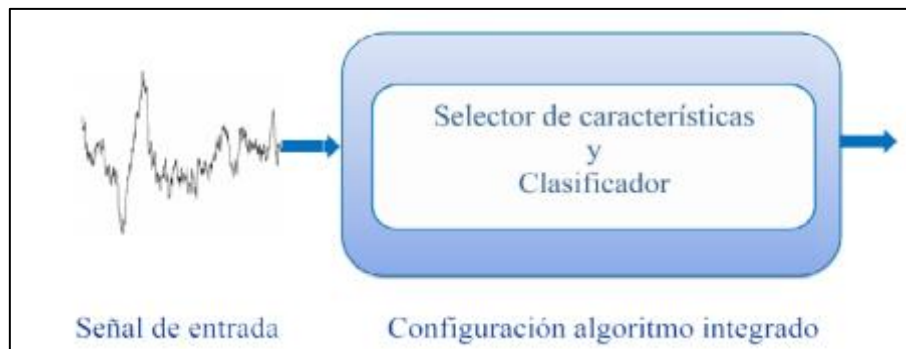


Figura 2.4 Diagrama de los componentes de un algoritmo del modelo integrado. Imagen extraída de [9].

La ERC consiste en un proceso iterativo que se puede resumir en 3 pasos:

- Entrenamiento del clasificador
- Cálculo del criterio para establecer un ranking entre las características.
- Eliminación de las características que se encuentran en el parte inferior del ranking.

El algoritmo finaliza cuando el criterio de detención se ha alcanzado o cuando no hay más características que evaluar.

La aplicación de la ERC empleada en la investigación en la cual se basa este trabajo, toma como clasificador el Discriminador Lineal de Fisher, empleando como criterio de eliminación el valor absoluto de los pesos, que son los coeficientes que conforman la combinación lineal que posibilita la clasificación. Es otras palabras, se van eliminando las características que sean menos relevantes, indicados por los pesos de menor módulo que se le asocia en la etapa de clasificación [9].

2.3.3. Clasificador LDA

El clasificador que se utiliza es el Discriminante Lineal de Fisher, también conocido como análisis de discriminación lineal (LDA, *Linear Discriminant Analysis*) es un método empírico de clasificación que se basa solamente en los vectores de atributos, es decir, en las características de los patrones. El objetivo del LDA es encontrar una transformación tal que la distancia entre clases sea máxima y la distancia entre los elementos de una misma clase sea mínima en el espacio transformado, que tiene una dimensionalidad menor que el de origen. Estas distancias se miden utilizando matrices de dispersión, entre clases y dentro de cada clase.

Se eligió emplear este tipo de clasificadores dado que tiene bajos requerimientos computacionales para su implementación y es estable ante variaciones en las señales de entrenamiento [9].

La implementación del LDA consiste en un producto punto entre la señal de EEG (pre-procesada) y un vector de pesos. El resultado de este producto punto es comparado con un umbral para poder determinar la presencia o no del potencial P300.

2.4. Modificación de parámetros en la implementación

2.4.1. 10 Canales vs. 1 canal

Uno de los interrogantes al implementar una ICC en un sistema dedicado es la conveniencia, respecto al rendimiento del procesador, de aplicar el procesamiento para la detección del potencial P300 a cada canal registrado por separado o a los 10 canales concatenados de como se muestra en la Figura 2.5.

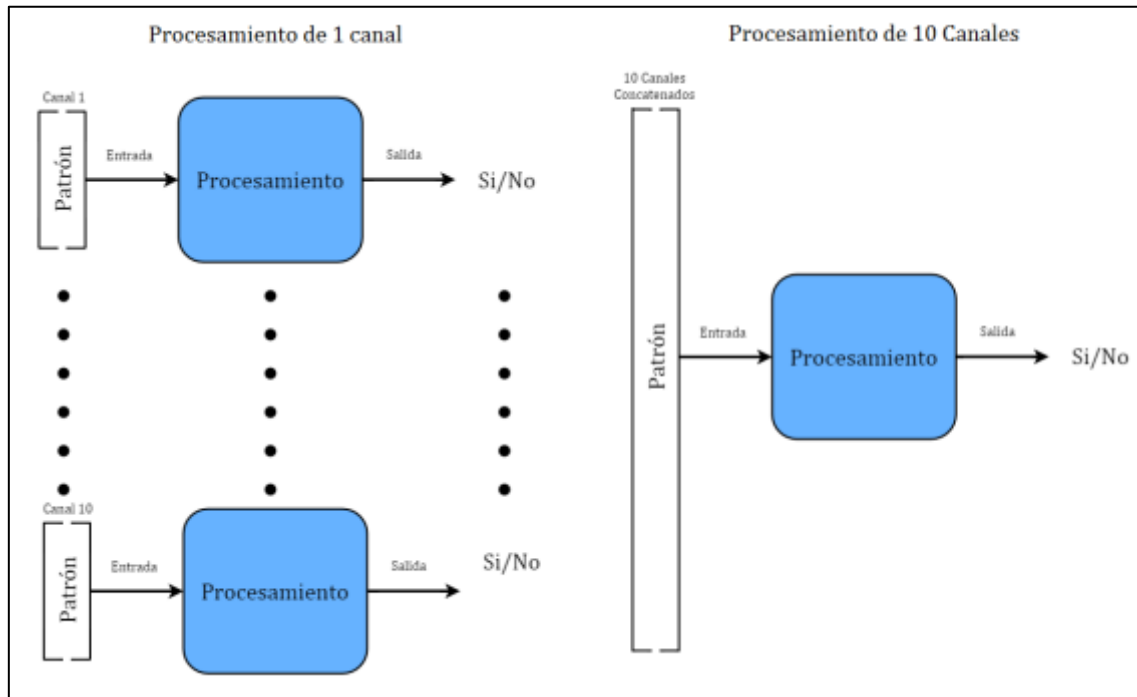


Figura 2.5. Diagrama que muestra el procesamiento de los patrones de 10 canales por separado y el procesamiento de los patrones de 10 canales concatenados.

Cabe destacar que, para cada uno de estos casos la entrada al sistema es distinta. Por lo tanto, los coeficientes empleados (pesos, umbrales, etc.) así como los índices de eficacia que estos posean pueden ser distintos. Estos índices, así como el desempeño del procesador que ejecute el procesamiento propuesto, son factores a evaluar en una instancia previa a la implementación de estos sistemas.

2.4.2. Longitud de las Señales

En la etapa de investigación, se utilizaron señales electroencefalográficas adquiridas a una frecuencia de muestreo de 256 Hz. Sin embargo, antes de desarrollar los algoritmos estas señales fueron submuestreadas a 64 Hz, lo cual no impacta sobre el resultado de la aplicación de las técnicas de procesamiento ya que el potencial P300 a detectar es una señal de baja frecuencia. Esto permite disminuir la cantidad de muestras que ingresan a los algoritmos implementados [9].

La bibliografía indica que el pico del potencial P300 se encuentra entre los 250 y 500 mseg posteriores al estímulo, aunque este rango puede variar dependiendo de varios factores: la modalidad del estímulo, condiciones de la tarea, la edad del sujeto, etc. [23] [7].

Se decidió tomar 700 mseg de la señal registrada para la implementación de los algoritmos, de manera de obtener un intervalo de 200 mseg de seguridad ante eventuales desplazamientos de la información contenida en la señal.

Por lo tanto si se toman los primeros 700 mseg de la señal registrada, implicaría trabajar con las primeras 45 muestras de las 64 que se tienen en un segundo de registro. Esta estrategia se aplica en vistas de disminuir aún más los datos de entrada al procesamiento ejecutado, sin afectar de alguna manera los resultados, basándose en información que reporta la bibliografía.

Si tomamos el mismo concepto para el procesamiento de los 10 canales concatenados tenemos 450 muestras (45 muestras x 10 canales) como entrada al procesamiento.

Es por ello que es de interés saber los recursos empleados en el procesamiento cuando las señales tienen un tamaño de 45 y 450 muestras.

Con intención de generar información respecto a los recursos empleados por los algoritmos en función la longitud de la señal también se realizarán mediciones para otros tamaños, estas serán: 50, 100, 200, 300 y 400 muestras.

2.4.3. Tipos de datos

En hardware digital, los números son almacenados como números binarios. Un número binario es una secuencia de bits con una longitud fija. Cómo los componentes del hardware o funciones de software interpretan esta secuencia de bits es definida como el tipo de dato [24].

El tipo de dato es otra variable que tiene impacto sobre el desempeño del procesamiento sobre un procesador. El conjunto de instrucciones utilizadas por el microprocesador, así como extensiones especializadas para distintos tipos de procesamiento, son dependientes del tipo de dato. Por lo que evidentemente variaciones en este parámetro repercutirá en el uso de recursos evaluados en este trabajo para conocer el rendimiento del microprocesador.

Los tipos de datos que se utilizan habitualmente para el procesamiento de señales son: Enteros de 8, 16 y 32-bits; punto fijo de 8, 16 y 32-bits; y Punto flotante de precisión simple (32-bits). De los cuales, los tipos datos de 8 bits no se utilizaron en este trabajo ya que se considera que el procesamiento que se puede realizar con el rango que estos ofrecen es limitado.

Por lo tanto se implementaron algoritmos con los siguientes 5 tipos de datos: Enteros de 32 bits (int32), enteros de 16 bits (int16), punto fijo de 32 bits (q31), punto fijo de 16 bits (q15) y Punto flotante de precisión simple (float32 o single).

2.4.3.1. Punto Fijo de 16 y 32 bits

Los formatos de punto fijo son formas de representar números racionales utilizando números enteros para la parte entera y para la parte fraccional. Se debe especificar cuantos bits representan la parte entera del número y la cantidad de bits que representa la parte fraccionaria. Es decir, donde se coloca la marca (la coma) que separa la parte entera de la fraccional. La notación más general utilizada en la bibliografía es $Q_n.m$, donde n es el número de bits que representan la parte entera, y m el número de bits que representan la parte fraccionaria [25].

Un tipo de dato de punto fijo es caracterizado por 3 factores: una longitud de palabra en bits, la posición del punto binario y si se usa un dato con o sin signo. La posición del punto binario es el medio por el cual los valores del punto fijo son interpretados.

El número de bits de la parte entera (n) más el número de la parte fraccionaria (m) debe ser el total de números de bits usados para representar el número, en este contexto serían la longitud de una palabra o media palabra (32 o 16-bits) [25].

En el caso de que la parte entera sea de un solo bit, usualmente éste se suele omitir en la denominación tal valor, llamándolos q15 y q31 en lugar de q1.15 y q1.31 respectivamente. Así como también, al tener un solo bit de la parte entera generalmente se entiende que este es el bit de signo.

Tomando el tipo de dato q15 como ejemplo, en la Figura 2.6 se muestra la representación de un número aproximadamente igual a 0,937744.

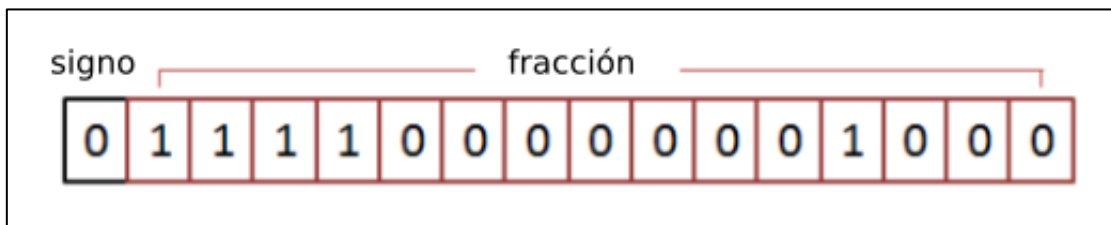


Figura 2.6. Representación de un tipo de dato punto fijo (q15), equivalente a 0,937744.

El número en decimal se representa como el número expresado en la parte fraccional dividido el máximo número que puede representar éste [25].

En la Tabla 2.1, se muestra como es el cálculo del valor decimal representado en el tipo de dato q15. Para el ejemplo de la Figura 2.6., así como también para el mínimo y para el máximo valor representable en este formato.

Concepto	Hexadecimal	Binario	Valor decimal	Cálculo de la representación de punto fijo	Número decimal representado en q15
Ejemplo de la Figura 2.6.	0x7808	0111100000001000	30728	$\frac{30728}{32768}$	0,937744
Mínimo valor representable	0xFFFF	1111111111111111	-32768	$\frac{-32768}{32768}$	-1
Máximo valor representable	0x7FFF	0111111111111111	32767	$\frac{32767}{32768}$	0,999969

Tabla 2.1. Ejemplos de Representación hexadecimal, binaria de tipo de dato q15 y su conversión para obtener la representación decimal.

Nótese en la Tabla 2.1 que en el cálculo de la representación del punto fijo el denominador es $32768 = 2^{15}$, donde 15 es la cantidad de bits de la parte fraccional.

De la Tabla 2.1 también se desprende que el rango de representación de este tipo de dato es de -1 a ≈ 1 . Y el paso entre un número y el siguiente (precisión) es constante y de una magnitud de $\frac{1}{2^{15}}$ ($3,0517 \times 10^{-5}$)

De igual manera la representación de q_{15} se extrapola al formato de punto fijo de 31 bits con un solo bit en la parte entera (q_{31}). Donde el rango de representación de éste también es de -1 a ≈ 1 pero la precisión de este tipo de dato es de $\frac{1}{2^{31}}$ ($4,6566 \times 10^{-10}$).

2.4.3.2. Enteros de 16 y 32 bits

La representación de enteros es el caso más sencillo del formato de punto fijo, en el cual el punto decimal se ubica en al final del último bit de la media palabra o palabra, según sean 16 o 32 bits, en otras palabras, esta representación de punto fijo no posee parte fraccionaria [26].

El tipo de variable utilizado en este trabajo es enteros con signo. De manera que tenemos 1 bit de signo y 15 o 31 bits de parte entera dependiendo si el dato representado es de 16 o 32 bits.

Por lo tanto, tomando como ejemplo un tipo de dato de 16 bits, en la Tabla 2.2 se muestra como es el cálculo del valor decimal representado en el tipo de dato entero de 16 bits para el valor del ejemplo de la Figura 2.6, así como también para el mínimo y para el máximo valor representable en este formato:

Concepto	Hexadecimal	Binario	Valor decimal (entero con signo)	Cálculo de la representación del entero	Número decimal representado con enteros
Valor de Figura 2.6	0x7808	0111100000001000	30728	$\frac{30728}{1}$	30728
Mínimo valor representable	0xFFFF	1111111111111111	-32768	$\frac{-32768}{1}$	-32768
Máximo valor representable	0x7FFF	0111111111111111	32767	$\frac{32767}{1}$	32767

Tabla 2.2. Ejemplos de Representación hexadecimal, binaria de tipo de dato int16 y su conversión para obtener la representación decimal.

Nótese que en la Tabla 2.1, en el cálculo de la representación de entero el denominador es $1 = 2^0$, donde 0 (cero) es la cantidad de bits de la parte fraccional.

De igual manera la representación de los enteros de 16 bits se extrapola a los enteros de 32-bits.

2.4.3.3. Punto flotante de precisión simple

El tipo de dato de punto flotante de precisión simple (denominado Single o float32 en este trabajo) es un método de representar una aproximación a los números reales en una manera que puede soportar un amplio rango de valores [26].

El estándar ANSI-IEEE 754 define el formato para números punto flotante de 32-bits llamados de “precisión simple”, así como también de 64 bits denominados de “precisión doble”. Sin embargo, dado que la unidad de punto flotante en el procesador Cortex-M4F sólo es simple precisión, se brindará información exclusivamente sobre este tipo de dato.

La codificación de los números de punto flotante es más compleja que la de los punto fijo. La idea básica consiste en usar una suerte de notación científica, donde una mantisa es multiplicada por 10 a la potencia de un exponente. Por ejemplo 5.4321×10^6 , donde 5.4321 es la mantisa y 6 es el exponente. Este tipo de notación es muy útil para representar números muy grandes así como también números muy pequeños [26].

La representación de punto flotante es similar a la notación científica, excepto que todo es llevado a cabo en base 2 en lugar de en base 10. Los números son representados aproximadamente a un número entero de un determinado número de dígitos significativos (la mantisa) y escalados utilizando un exponente.

Un número de punto flotante de precisión simple se almacena como se muestra en la Figura 2.7.

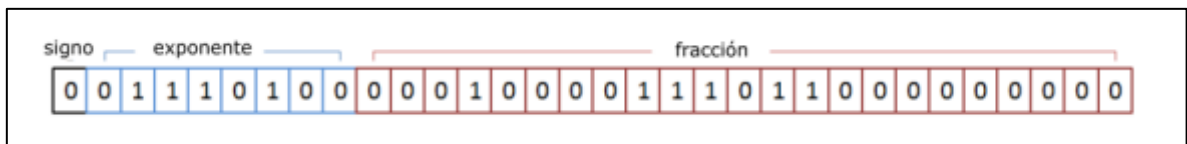


Figura 2.7. Representación de un número de punto flotante de precisión simple según ANSI-IEEE 754.

El primer bit es el bit de signo (S), a continuación, hay ocho bits para el exponente (E), y los 23 bits restantes se utilizan para la mantisa (M).

Estos bits forman el Número Punto Flotante (NPF) con la siguiente relación:

$$\text{NPF} = (-1)^S \times M \times 2^{E-127}$$

Donde el término $(-1)^S$ significa que si S es 0 el número es positivo y si es 1 el número es negativo. La variable E es un número entre 0 y 255, dado que se representa con 8 bits. Restando 127 a este número permite que el rango del término del exponente vaya de 2^{-127} a 2^{128} . En otras palabras, el exponente se encuentra almacenado en binario con offset, con un offset de 127.

La mantisa está formada por 23 bits en binario fraccional. Por ejemplo 2,783 es interpretado como:

$$2 + \frac{7}{10} + \frac{8}{100} + \frac{3}{1000} = 2,783d$$

En binario fraccional 1.0101 significa:

$$1 + \frac{0}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} = 1.0101b = 1,3125d$$

Para más información respecto a tipos de dato de punto fijo o flotante, referirse a [26] y [22].

Contar con un hardware especializado para operar estos tipos de datos ayudará a reducir el costo en materia de recursos al ejecutar el procesamiento. Cabe mencionar que el punto flotante libera al programador de la tediosa tarea de averiguar si el resultado de un cálculo estará dentro de tamaño razonable en comparación con el tamaño de la variable, también conocido como desbordamiento⁶ [25].

2.3.4. Implementación

A continuación se presenta la ecuación para determinar cuántas variaciones de procesamiento se implementaron.

N° Variaciones de algoritmo x N° Tipo de datos x N° de longitudes de señales medidas

Promediación Coherente:	$2 \times 6 \times 7 = 84$
Extractores de Características:	$2 \times 5 \times 7 + 1 \times 5 \times 2 = 70 + 10 = 80$
Clasificadores:	$1 \times 6 \times 7 + 1 \times 6 \times 2 = 42 + 12 = 54$
Total:	$84 + 80 + 54 = 218$ variaciones de procesamiento.

Las variaciones de los algoritmos de la Promediación coherente son 2, una que promedia 2 y otra que promedia 4 señales. Los extractores de características ofrecen 3 variaciones: 2 submuestreos equitemporales (32 y 16 Hz) y el submuestreo ERC. Los clasificadores son de un solo tipo, LDA, que se aplica para señales de longitudes de señales previamente establecidas en este trabajo ($1 \times 6 \times 7$) y otra tomando para las longitudes de las señales (2 determinadas longitudes para ser exactos) después de haber sido procesada por el algoritmo de submuestreo ERC ($1 \times 6 \times 2$).

La cantidad de tipos de datos evaluados son 5: single, int32, int16, q15 y q16. Sin embargo, en las técnicas de promediación coherente y clasificadores se observa que hay 6 tipos. Esto se debe a que también se implementó procesamiento de variables de tipo int16 pero utilizando herramientas de SIMD. Es importante aclarar que, el procesamiento mediante instrucciones SIMD no es llevado a cabo por un nuevo tipo de dato, esta variación se incluye en este campo solamente para facilitar la visualización de la diversificación del procesamiento abordado.

Como se mencionó en la sección 2.4.1.1, la cantidad de longitudes de señales empleadas son 7: 45, 50, 100, 200, 300, 400 y 450 muestras. No obstante, en el submuestreo ERC y la posterior etapa de clasificación, estas longitudes varían dependiendo si el procesamiento se llevó a cabo utilizando un solo canal o 10 canales. Por esta razón aparece el factor “2” en el segundo término de los submuestreos y clasificadores.

Para esclarecer las variaciones de procesamiento implementadas se presentan los siguientes diagramas en la Figura 2.8, 2.9 y 2.10:

⁶ No se debe confundir este tipo de desbordamiento, que es producto de las limitaciones de la representación con una cantidad determinada de bits, con el desbordamiento producido en el stack descrito en la sección 2.2.2., que se debe a la capacidad de memoria RAM del dispositivo.

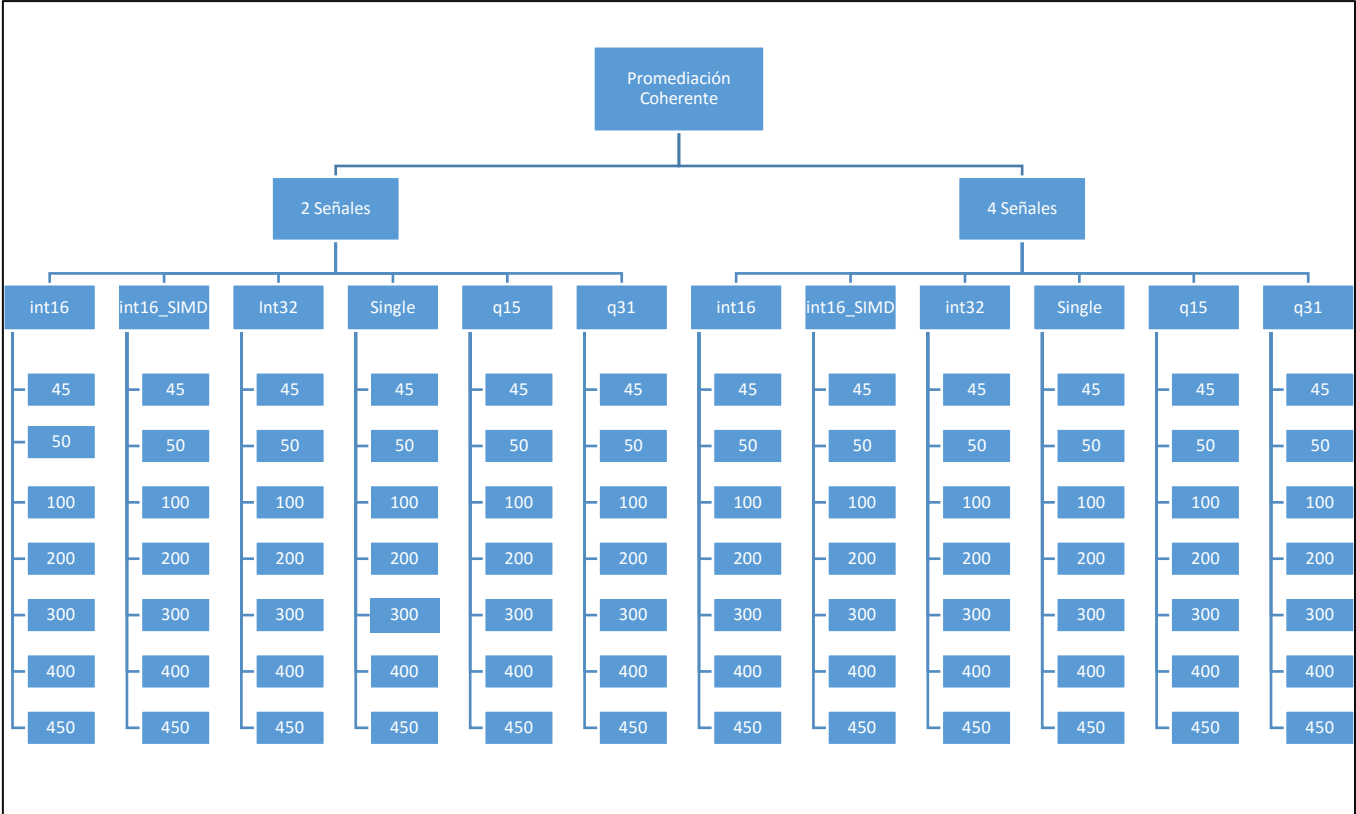


Figura 2.8. Variaciones de procesamiento implementadas para la Promediación Coherente.

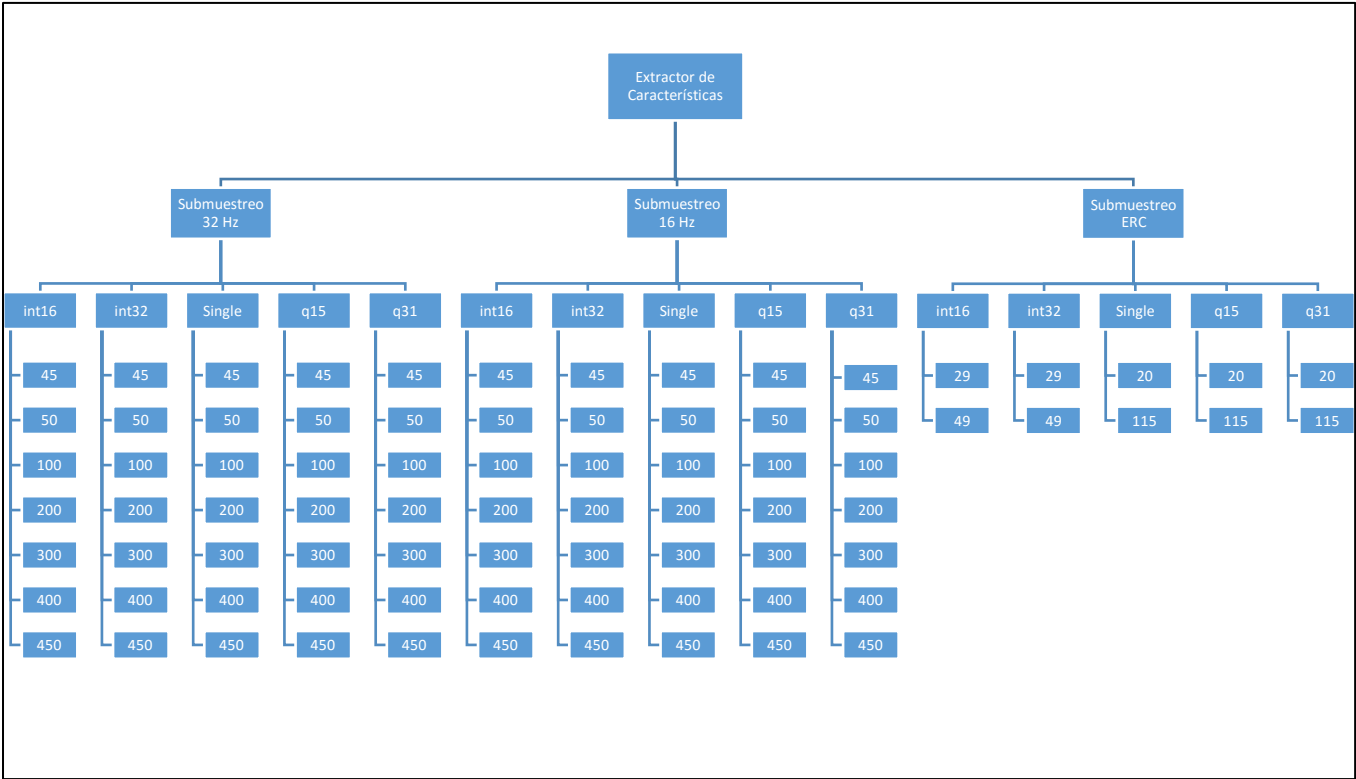


Figura 2.9. Variaciones de procesamiento implementadas para la Extracción de Características.

Cabe mencionar que para el submuestreo ERC, la investigación del procesamiento de señales arrojó diferentes valores de longitud de la señal a la salida de esta etapa para enteros y para single. Para los datos de tipo punto fijo se adoptaron las longitudes de los punto fijo como referencia. Esto se puede ver en la Figura 2.9.



Figura 2.10. Variaciones de procesamiento implementadas para el Clasificador.

En la Figura 2.10 se observa que además de las longitudes de señal planteadas inicialmente para la evaluación se agregan longitudes de señales que tienen éstas al ser previamente procesadas por el submuestreo ERC (bloques en color verde).

Capítulo 3: Metodología

3.1. Generación de herramientas

3.1.1. Comunicación uC-PC

Además de la conexión requerida entre el microcontrolador (μ C) y la PC para cargar las rutinas a ejecutar en él, se necesitó establecer otro tipo de comunicación para extraer datos del uC. Como por ejemplo, para extraer el tiempo de ejecución de determinado algoritmo medido con herramientas del propio μ C o los resultados de procesamiento ejecutado en el μ C para hacer una verificación de éstos de manera externa. Por estos motivos fue necesario generar una herramienta para la transmisión de datos del μ C a la PC.

Se eligió una comunicación mediante el UART (Universal Asynchronous Receiver-Transmitter) del μ C dada la relativa simplicidad de su implementación. Para ello se programó una rutina donde se establecen: el puerto UART a utilizar (UART 2 de la EDU-CIAA), el modo FIFO (*First In- First Out*) de la pila, el *baud rate* y se habilita la UART para poner en funcionamiento la transmisión de datos desde el μ C.

Para la recepción de los datos desde la PC se utilizó el software “HyperTerminal” el cual sirve como gestor de transferencia de archivos entre dispositivos a través de un puerto serie. Este programa también permite la captura de los datos recibidos en archivos de texto.

3.1.2. Algoritmos de conversión de datos

Para que la comunicación entre el μ C y la PC mediante el HyperTerminal sea más sencilla de visualizar, se requiere que los datos enviados sean una cadena de caracteres (datos tipo char). Por ese motivo fue necesaria la programación de rutinas de conversión de datos a cadena de caracteres que se emplean inexorablemente para enviar los datos por el UART.

Por otro lado, para la etapa de verificación de resultados se realizaron algoritmos de conversión de estos valores a su representación binaria con el objetivo de no perder precisión en los datos de (particularmente en los de tipo punto flotante) en el envío de datos por la UART.

3.2. Método de medición de parámetros.

3.2.1. Tiempo de Ejecución

Se elaboraron dos maneras diferentes de medir intervalos de tiempo utilizando herramientas del propio microprocesador mediante:

- Un Timer.
- Un registro contador de ciclos de reloj (CYCCNT, *Clock Cycles Counter*).

El CYCCNT es un registro de 32-bits que forma parte del *Data Watchpoint and Trace* (DWT), que es una herramienta de depuración del procesador que provee entre otras funcionalidades, el monitoreo de operación del mismo sin utilizar ciclos de reloj adicionales.

Se realizó una verificación de las mediciones temporales para corroborar la confiabilidad de las mismas. Esta verificación se llevó a cabo de la siguiente manera:

Se implementó una rutina en la cual se establece una frecuencia de encendido y apagado de un led (*blinking*) de la EDU-CIAA (Figura 3.1); ésta emplea las funciones de mediciones temporales como cronómetro para su funcionamiento. En esta rutina el parámetro a establecer es el tiempo periodo de la señal (ciclo de trabajo del 50%). Luego se midió una serie de señales cuadradas sobre el pin correspondiente mediante un osciloscopio digital de manera tal de poder evaluar si los tiempos medidos se condecían con los valores establecidos previamente. Se utilizó un osciloscopio digital Tektronix TBS 1052B-EDU (Figura 3.2).

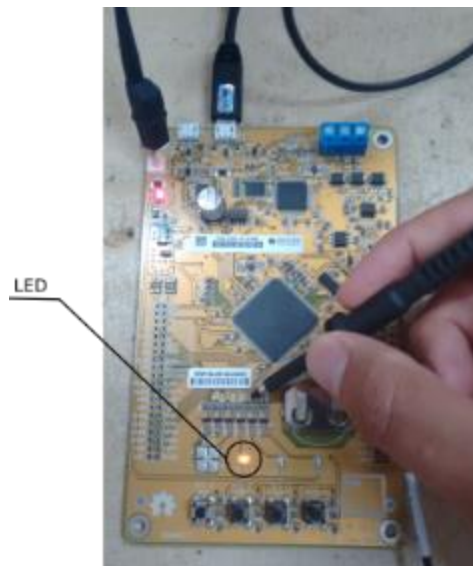


Figura 3.1: Imagen de la EDU-CIAA al registrar las mediciones del *blinking*.

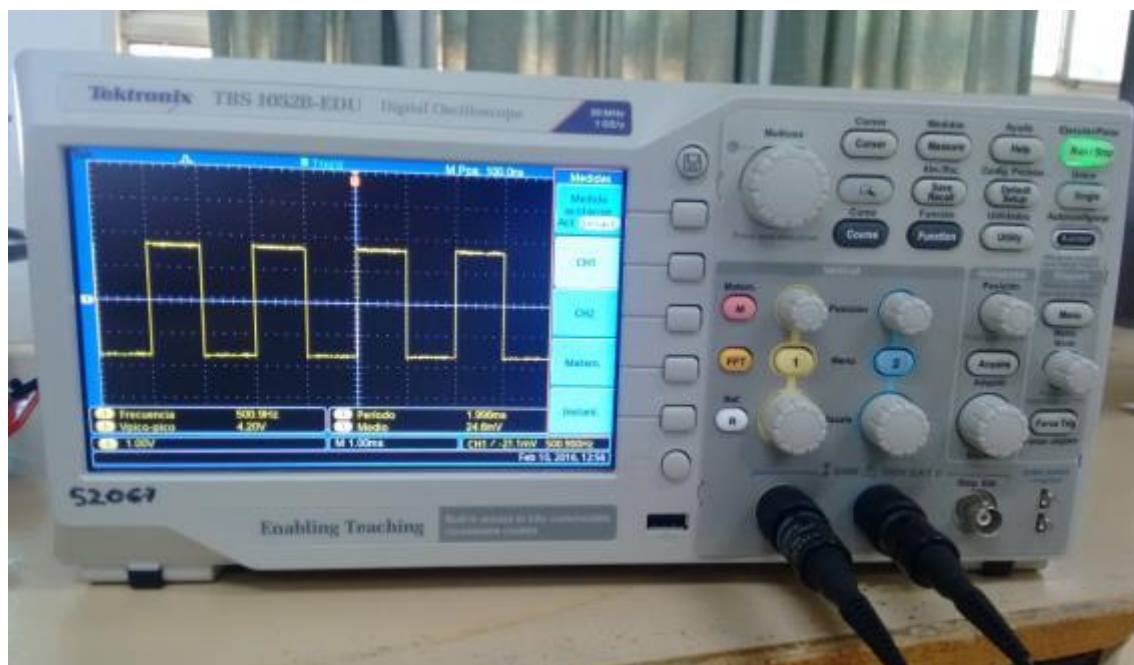


Figura 3.2: Imagen del osciloscopio digital Tektronix TBS 1052B-EDU registrando las señales sobre el LED.

3.2.1.1. Medición mediante Timer

Se programó una función donde se habilita el Timer a utilizar, se establece un Prescaler⁷ y se indica el clock máximo del Timer, que en este caso es el mismo que el clock del procesador (204MHz).

Los resultados de la siguiente medición se detallan en la Tabla 3.1

Periodo establecido	Periodo medido
2,00[ms]	1,99[ms]
1,00[ms]	1,00[ms]
200,00[μs]	202,30[μs]
100,00[μs]	103,00[μs]
20,00[μs]	23,20[μs]
10,00[μs]	12,79[μs]

Tabla 3.1. Resultados de las mediciones realizadas mediante el Timer.

En las siguientes Figuras (3.3, 3.4 y 3.5) se observan capturas de pantalla del osciloscopio en la medición de las señales sobre el LED.

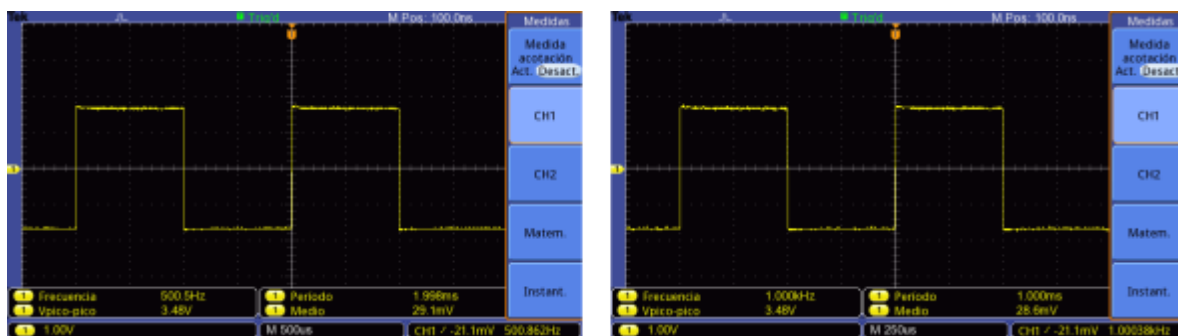


Figura. 3.3. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el Timer. Señal cuadrada de 500 Hz (izquierda) y una de 1 KHz (derecha).

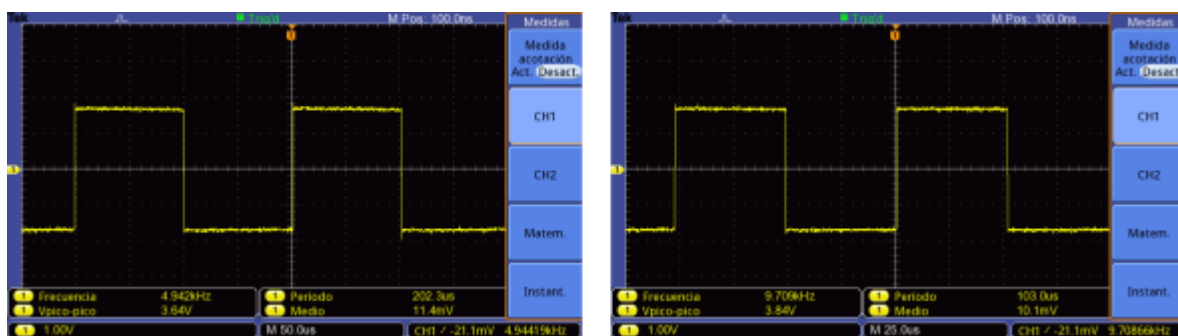


Figura. 3.4. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el Timer. Señal cuadrada de 5 KHz (izquierda) y una de 10 KHz (derecha).

⁷ El Prescaler es un número entero por el cual se divide la frecuencia del clock del procesador para tomar como base de tiempo del timer. En este caso, el prescaler se estableció en 1 y la frecuencia utilizada para la medición es la misma que la del clock del procesador.

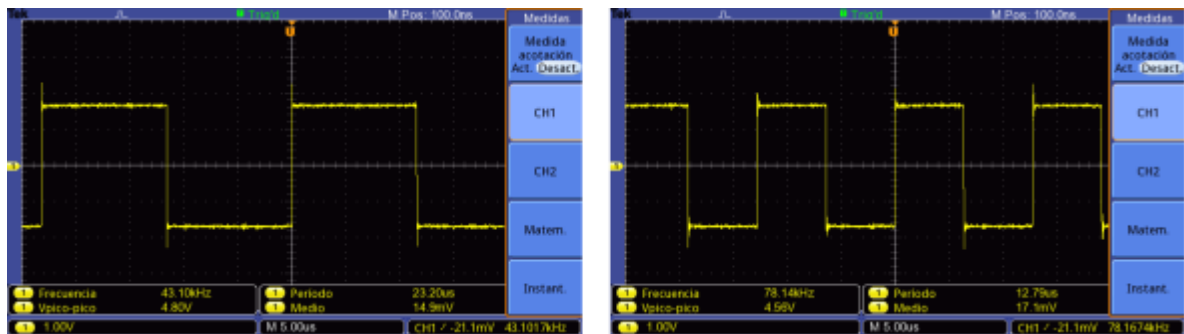


Figura. 3.4. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el Timer. Señal cuadrada de 50 KHz (izquierda) y una de 100 KHz (derecha).

3.2.1.2. Medición mediante DWT

Se programó una función el cual lee el registro CYCCNT de la DWT que oficia de “cronómetro digital”. Este método también se verificó de igual manera que con el Timer. Los resultados se muestran en la Tabla 3.2.

Periodo establecido	Periodo medido
2,00[ms]	1,99[ms]
1,00[ms]	1,00[ms]
200,00[μs]	201,40[μs]
100,00[μs]	101,80 [μs]
20,00[μs]	21,85[μs]
10[μs]	12,20[μs]

Tabla 3.2. Resultados de las mediciones realizadas mediante el CYCCNT.

En las siguientes Figuras (3.6, 3.7 y 3.8) se observan capturas de pantalla del osciloscopio en la medición de las señales sobre el LED.

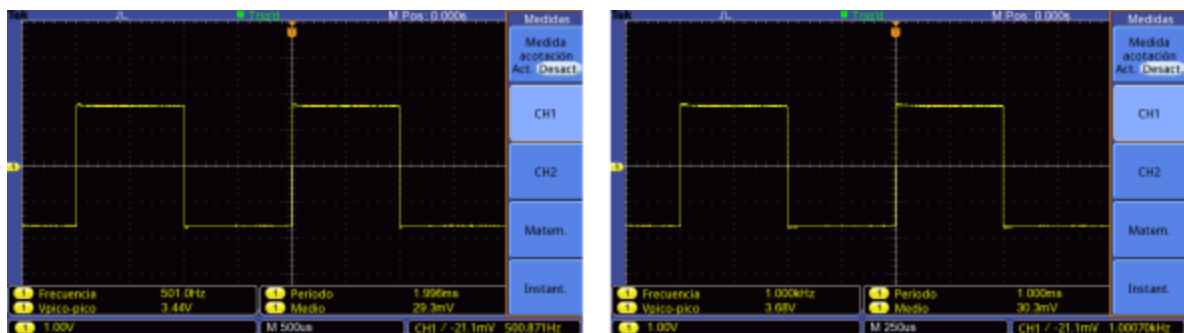


Figura 3.6. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el CYCCNT. Señal cuadrada de 500 Hz (izquierda) y una de 1 KHz (derecha).

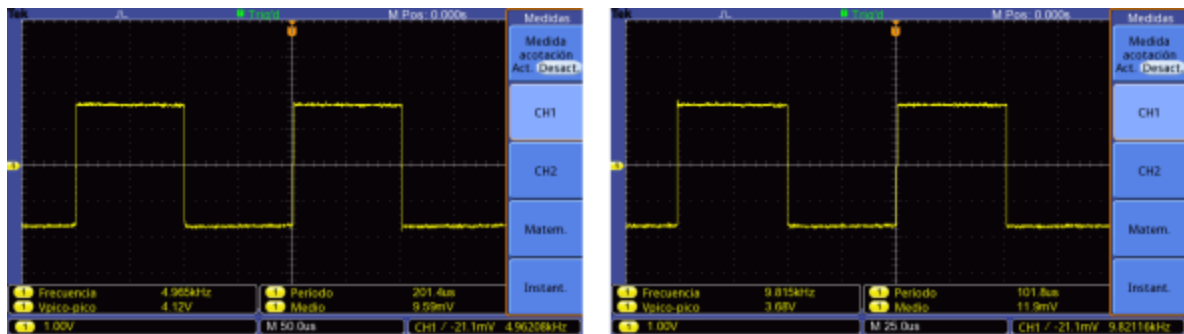


Figura 3.7. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el CYCCNT. Señal cuadrada de 5 KHz (izquierda) y una de 10 KHz (derecha).

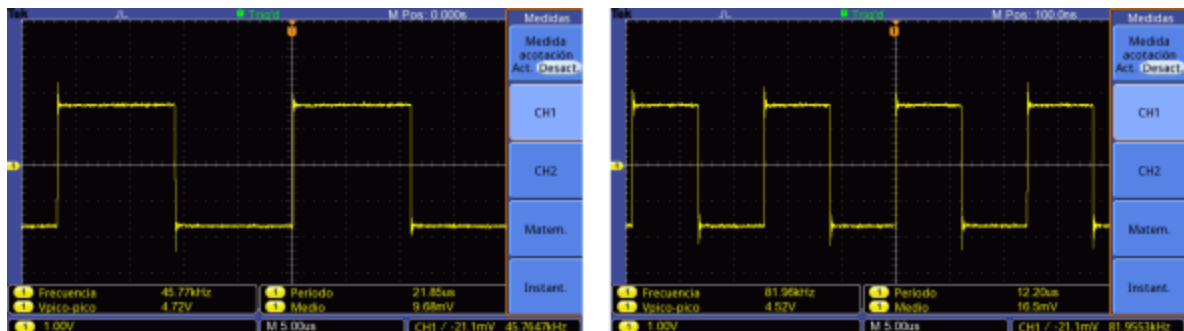


Figura 3.8. Imágenes adquiridas del osciloscopio al realizar las mediciones mediante el CYCCNT. Señal cuadrada de 50 KHz (izquierda) y una de 100 KHz (derecha).

En la Tabla 3.3 se comparan las mediciones y los errores obtenidos de las mediciones por ambos métodos.

Periodo establecido	Periodo medido DWT	Periodo medido Timer	Error DWT	Error Timer
2,00[ms]	1,99[ms]	1,99[ms]	0,10[ms]	0,10[ms]
1,00[ms]	1,00[ms]	1,00[ms]	0[ms]	0[ms]
200,00[µs]	201,40[µs]	202,30[µs]	1,40[ms]	2,30[ms]
100,00[µs]	101,80 [µs]	103,00[µs]	1,80[µs]	3,00[µs]
20,00[µs]	21,85[µs]	23,20[µs]	1,85[µs]	3,20[µs]
10,00[µs]	12,20[µs]	12,79[µs]	2,20[µs]	2,79[µs]

Tabla 3.3. Comparación de resultados de los métodos de medición temporal y sus errores respecto al valor esperado.

Cabe mencionar que estas mediciones se realizaron para corroborar que los métodos desarrollados no tuvieran errores groseros en sus mediciones. La magnitud de los errores encontrados en las mediciones eran esperables dado que el algoritmo de blinking implementado no estaba orientado a la precisión del mismo.

Como se puede observar en la Tabla 3.3, las mediciones llevadas a cabo no reportan errores significativos, las diferencias encontradas se deben al uso de recursos del procesador que hacen menos determinística la ejecución de sus tareas.

Sin embargo, el método mediante el DWT presenta menores errores que mediante el Timer. Es por esto que las mediciones posteriores se llevaron a cabo utilizando este método.

3.2.1.3. Repetición de las mediciones

Respecto a los tres parámetros que se proponen medir en este trabajo (tiempo de ejecución, memoria RAM y memoria ROM), las mediciones de uso de memorias son de tipo estáticas, lo que significa que si se lleva a cabo la medición repetidas veces sin modificar el código ejecutado los resultados seguirán siendo los mismos.

Por otro lado, en las mediciones temporales no sucede lo mismo, los resultados obtenidos varían al ejecutarse las mediciones repetidas veces aunque no se modifique el código.

A pesar que el microprocesador elegido se encuentra orientado a tener un comportamiento temporal determinístico, es posible que los resultados varíen debido a retardos introducidos de controlador de buses y pipeline, como se mencionó en el capítulo 1 en la sección “Buses internos y timing”.

Las rutinas para extraer información del μC ocupan recursos de hardware que repercuten en los datos obtenidos análogamente a lo explicado en el capítulo 2 en la sección “Memoria RAM”. Es por esto que se realizaron ensayos de repetibilidad las mediciones temporales para un posterior análisis.

Se determinó el valor de repeticiones en 1000 (mil) de manera empírica, dado que se llevaron a cabo las mediciones con otros valores y los resultados arrojaban patrones independientes de la cantidad de repeticiones.

3.2.2. Memoria RAM

Para la medición del uso de memoria RAM se utilizó la opción de compilador “*-fstack-usage*”, que soporta el compilador GCC. Esta opción genera un archivo extra que especifica la máxima cantidad de stack usada por una función. El contenido del archivo generado se muestra en la Figura 3.9. Cada línea de este archivo esta compuesto por 3 campos:

- El nombre de la función
- Una cantidad de bytes
- Uno o más calificadores.

```

1 arm_math.h:469:25:clip_q63_to_q31 32 static
2 arm_add_f32.c:73:6:arm_add_f32 64 static
3 arm_scale_f32.c:89:6:arm_scale_f32 48 static
4 arm_add_int32.c:76:6:arm_add_int32 64 static
5 arm_shift_int32.c:84:6:arm_shift_int32 64 static
6 arm_add_int16.c:76:6:arm_add_int16 48 static
7 arm_add_int16_SIMD.c:66:6:arm_add_int16_SIMD 80 static
8 arm_shift_int16.c:84:6:arm_shift_int16 48 static
9 arm_add_q31.c:67:6:arm_add_q31 120 static
10 arm_shift_q31.c:84:6:arm_shift_q31 80 static
11 arm_add_q15.c:66:6:arm_add_q15 80 static
12 arm_shift_q15.c:66:6:arm_shift_q15 128 static
13 scu_18xx_43xx.h:135:20:Chip_SCU_PinMuxSet 16 static
14 scu_18xx_43xx.h:150:20:Chip_SCU_PinMux 24 static
15 uart_18xx_43xx.h:306:20:Chip_UART_TXEnable 16 static
16 uart_18xx_43xx.h:329:20:Chip_UART_SendByte 16 static
17 uart_18xx_43xx.h:410:20:Chip_UART_SetupFIFOS 16 static
18 uart_18xx_43xx.h:512:24:Chip_UART_ReadLineStatus 16 static
19 prom_coherente.c:99:6:Prom_Coherente_two_signals_f32 24 static
20 prom_coherente.c:124:7:Prom_Coherente_four_signals_f32 24 static
21 prom_coherente.c:142:6:Prom_Coherente_two_signals_int32 24 static
22 prom_coherente.c:157:7:Prom_Coherente_four_signals_int32 24 static
23 prom_coherente.c:174:6:Prom_Coherente_two_signals_int16 24 static
24 prom_coherente.c:187:7:Prom_Coherente_four_signals_int16 24 static
25 prom_coherente.c:205:7:Prom_Coherente_two_signals_int16_SIMD 24 static
26 prom_coherente.c:219:7:Prom_Coherente_four_signals_int16_SIMD 24 static
27 prom_coherente.c:236:7:Prom_Coherente_two_signals_q31 24 static
28 prom_coherente.c:249:8:Prom_Coherente_four_signals_q31 24 static
29 prom_coherente.c:266:7:Prom_Coherente_two_signals_q15 24 static
30 prom_coherente.c:279:8:Prom_Coherente_four_signals_q15 24 static
31 submuestreo.c:83:6:submuestreo_32Hz_f32 32 static
32 submuestreo.c:97:6:submuestreo_16Hz_f32 40 static
33 submuestreo.c:119:6:submuestreo_32Hz_int32 32 static

```

Figura 3.9. Captura de pantalla de un archivo tipo “su” en el cual se observan en cada línea los campos descriptos (nombre, cantidad de bytes máximos utilizados y un clasificador).

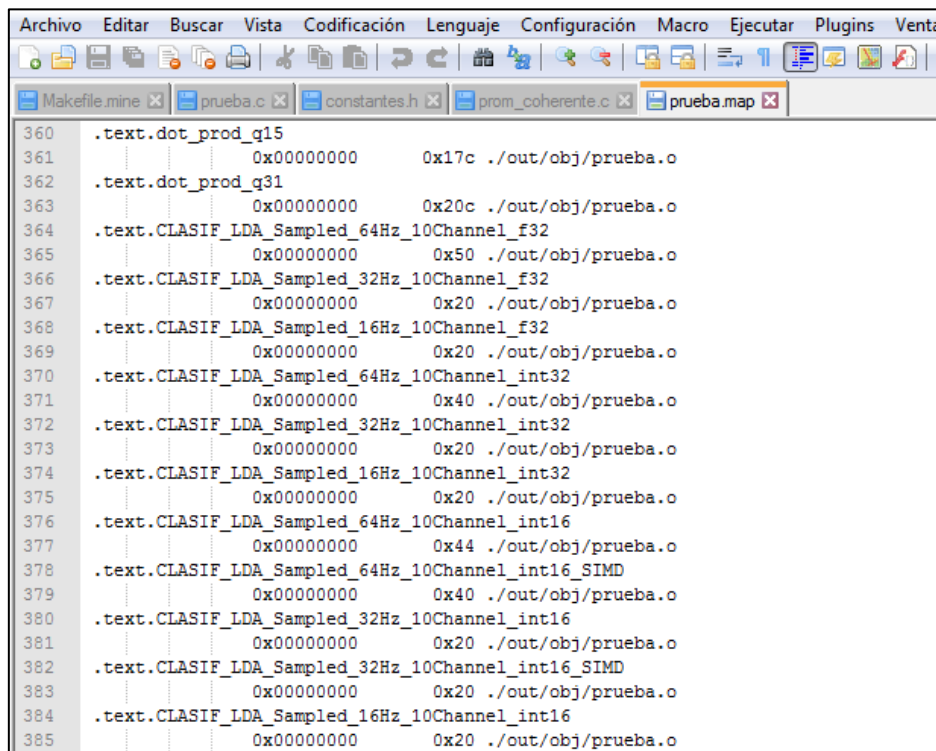
El segundo campo corresponde al tamaño máximo de stack que utiliza la función sin tener en cuenta el consumo de stack de llamadas a subfunciones dentro de ella.

El calificador “*static*” indica que los datos asignados en memoria por la función son estáticos. En este caso el tamaño máximo del stack utilizado por la función presenta una medida confiable de la utilización de esta función, y este valor nunca excede del valor presentado en esta columna.

El calificador “*dynamic*” indica que los datos asignados en la función son dinámicos. Cuando este calificador aparece, la medida del segundo campo no es confiable, dado que el uso del stack puede superar el valor presentado en el segundo campo [27] [19].

3.2.3. Memoria ROM

La memoria ROM que ocupan los algoritmos en el μ C fue mensurado utilizando como herramienta el archivo con extensión “.map”. Este archivo es generado por el linker en la etapa de compilación y, su formato se muestra en la Figura 3.10. Este archivo contiene la información de la localización en el mapa de memoria de los algoritmos, así como también el espacio que éstos ocupan.



```

360 .text.dot_prod_q15
361      0x00000000      0x17c ./out/obj/prueba.o
362 .text.dot_prod_q31
363      0x00000000      0x20c ./out/obj/prueba.o
364 .text.CLASIF_LDA_Sampled_64Hz_10Channel_f32
365      0x00000000      0x50 ./out/obj/prueba.o
366 .text.CLASIF_LDA_Sampled_32Hz_10Channel_f32
367      0x00000000      0x20 ./out/obj/prueba.o
368 .text.CLASIF_LDA_Sampled_16Hz_10Channel_f32
369      0x00000000      0x20 ./out/obj/prueba.o
370 .text.CLASIF_LDA_Sampled_64Hz_10Channel_int32
371      0x00000000      0x40 ./out/obj/prueba.o
372 .text.CLASIF_LDA_Sampled_32Hz_10Channel_int32
373      0x00000000      0x20 ./out/obj/prueba.o
374 .text.CLASIF_LDA_Sampled_16Hz_10Channel_int32
375      0x00000000      0x20 ./out/obj/prueba.o
376 .text.CLASIF_LDA_Sampled_64Hz_10Channel_int16
377      0x00000000      0x44 ./out/obj/prueba.o
378 .text.CLASIF_LDA_Sampled_64Hz_10Channel_int16_SIMD
379      0x00000000      0x40 ./out/obj/prueba.o
380 .text.CLASIF_LDA_Sampled_32Hz_10Channel_int16
381      0x00000000      0x20 ./out/obj/prueba.o
382 .text.CLASIF_LDA_Sampled_32Hz_10Channel_int16_SIMD
383      0x00000000      0x20 ./out/obj/prueba.o
384 .text.CLASIF_LDA_Sampled_16Hz_10Channel_int16
385      0x00000000      0x20 ./out/obj/prueba.o

```

Figura 3.10 Captura de pantalla de un archivo tipo “map”, que indica en hexadecimal la cantidad de bytes que ocupa la función en memoria.

3.3. Verificación del procesamiento

Se realizó una verificación del procesamiento de señales realizado en el microprocesador, en vistas de realizar mediciones sobre algoritmos que se tengan certeza de que realizan un procesamiento correcto de la señal.

Este procedimiento consiste en comparar la salida del procesamiento ejecutado en el microprocesador con la salida del mismo procesamiento de un software de procesamiento de señales externo que ejecuta la misma operación con la misma entrada. En este caso el software que se utilizó es MATLAB®⁸ ya que es un software de uso ampliamente aceptado por el ámbito académico y la Facultad de Ingeniería de la Universidad Nacional de Entre Ríos provee licencias para su uso.

Para las Promediaciones coherentes se utilizaron 2 señales de 6750 muestras y 4 señales de 3375. Para el submuestreo equitemporal se utilizó una señal de 13500 muestras. Para los submuestreos ERC se utilizaron longitudes de señal establecidas por la cantidad de índices determinados en la instancia de investigación. Para la verificación de los

⁸ La versión utilizada en la Facultad de Ingeniería de la Universidad Nacional de Entre Ríos es MATLAB® R2013b (8.2.00.701) – 64-bits. August 13, 2013. Licencia número: 880991

clasificadores LDA se utilizaron 450 muestras, y para los clasificadores de las señales resultantes del submuestreo ERC se utilizaron longitudes de señal establecidas por la cantidad de pesos que también fueron determinadas también en la etapa de investigación.

Las longitudes de las señales empleadas en las verificaciones varían dado que en las promediaciones se quiso abarcar la mayor cantidad de cálculos para obtener más información en cuanto a los errores, por eso se utilizó información de todo un canal provisto de la etapa de investigación (13500 muestras).

En la verificación de los submuestreos, aunque no era clave el tamaño de las señales con las cuales se corroboraban dado que no se realizaban procesamientos aritméticos en esta etapa, se verificó con 13500 muestras también. Y por último, para los clasificadores se utilizaron señales de 450 muestras, ya que era la mayor cantidad de coeficientes disponibles de la etapa de investigación.

Capítulo 4: Implementación de algoritmos

4.1. Herramientas utilizadas

La implementación de los algoritmos de procesamiento para la detección de P300 sobre el microcontrolador se efectuó en lenguaje C, y el compilador utilizado fue gcc-arm-none-eabi.

Para la programación de los algoritmos se utilizaron librerías DSP⁹ incluidos en la Interface Estándar de Software para Microcontroladores Cortex (CMSIS del inglés *Cortex Microcontroller Software Interface Standard*).

4.1.1. CMSIS

La CMSIS es una capa de abstracción independiente del chip (*device-independent*) para la serie de procesadores Cortex-M (Cortex M-3, Cortex-M4, Cortex-M0 y Cortex-M0+) y define interfaces de herramientas genéricas del procesador, ver esquema de la Figura 4.1. La CMSIS fue definida en cooperación con varios fabricantes, diseñadores y vendedores de software, y provee un enfoque común de interfaz a periféricos, sistemas operativos de tiempo real (RTOS), y componentes de middleware (Ethernet, stack USB, etc.). Esta interfaz de software para el procesador y los periféricos, simplifica el re-uso del software y reduce la curva de aprendizaje para desarrolladores en microcontroladores.

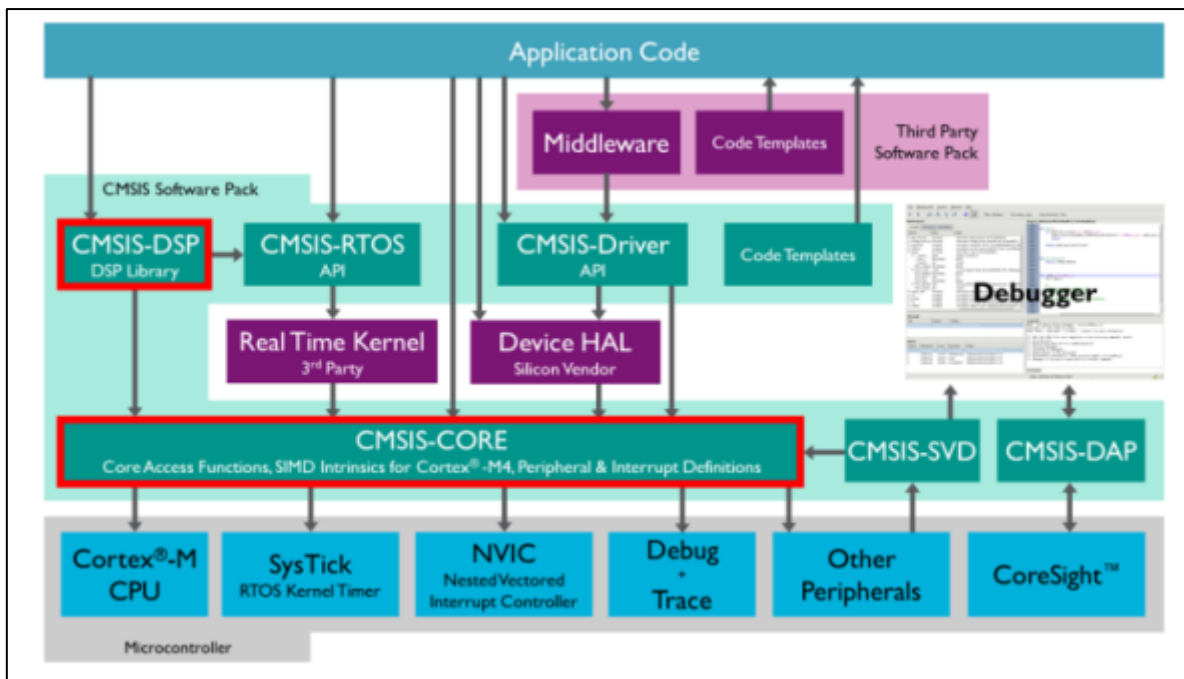


Figura 4.1. Diagrama de los bloques del CMSIS y su interacción con los componentes del microcontrolador y el software. Se encuentra recuadrado en rojo las partes del CMSIS utilizadas en este trabajo.

En la Figura 4.1 se ve como la aplicación interactúa con el hardware a través de una capa de abstracción. En la base se encuentra el procesador, la CMSIS-CORE, un conjunto de

⁹ En el siguiente link se encontrará información de las librerías así como también el acceso a ellas: <http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html>

funciones y definiciones que permiten mantener la uniformidad de criterios que mencionamos anteriormente.

La CMSIS puede verse como un conjunto de reglas y un grupo de archivos que conforman una capa que permite acceder a microcontroladores de diferentes fabricantes de una forma común; siempre y cuando utilicen un procesador de la serie Cortex-M. Las reglas de programación que esta interfaz impone, aseguran que las interfaces provistas por diferentes fabricantes tengan una filosofía y una forma común, de modo de poder cambiar de uno a otro sin mayores inconvenientes. El usuario gana así la libertad de poder migrar de fabricante, tanto para el microcontrolador como para el compilador, con una mínima intervención.

Los componentes del CMSIS son: CMSIS-CORE, CMSIS-Driver, CMSIS-DSP, CMSIS-RTOS API, CMSIS-DAP, CMSIS-Pack y CMSIS-SVD. De los cuales los que se utilizaron en este trabajo fueron la CMSIS-CORE y la CMSIS-DSP.

CMSIS-CORE: Es la estructura más importante que se encarga de definir:

- Una capa de abstracción (HAL de sus siglas en inglés *Hardware Abstraction Layer*).
- Una forma de organizar los header files
- Una forma de inicializar el sistema.

Internamente posee una serie de estructuras y los siguientes módulos, entre otros:

- Acceso a Registros (*Core Register Access*)
- Acceso a Periféricos (*Peripheral Access*)
- Acceso a herramientas de Depuración (*Debug Access*)
- Configuración del sistema y del clock

CMSIS-DSP: Es una colección de librerías con más de 60 funciones que resuelven problemas típicos de procesamiento de señales para varios tipos de datos: punto fijo (q7, q15 y q31) y punto flotante de precisión simple. La implementación para Cortex-M4 está optimizada para el conjunto de instrucciones SIMD.

4.1.2. SIMD

Para sacar provecho de la presencia de esta extensión, necesitamos escribir el código específicamente para ello y utilizar un compilador que sea capaz de detectar las operaciones transformables y emplear instrucciones específicas. Tanto el compilador GNU, GCC, como el de ARM, pueden realizar esta transformación si el programador respeta ciertas pautas en la escritura del código [13]. De esta manera el procesador empaqueta o desempaqueta (*pack* o *unpack*) y alinea los datos en registros SIMD mediante un grupo de instrucciones ad hoc [28].

4.1.3. MAC y FPU

En la ejecución de los algoritmos, utilizando herramientas de depuración se corroboró que efectivamente en los algoritmos de procesamiento se utilizan instrucciones de la MAC y de FPU, tales como: MUL (multiplicación de datos), ADD (suma de datos), VMUL (multiplicación de datos de tipo punto flotantes), VADD (suma de datos de tipo punto flotantes) y SMLAD (producto punto de registros de 16 bits utilizando SIMD).

4.2. Algoritmos de verificación del procesamiento

Para la verificación del procesamiento llevado a cabo en el μC se generaron funciones en Matlab, que calculan un error entre los resultados del procesamiento llevado a cabo en el μC con el realizado en Matlab. Las salidas de las funciones de verificación generalmente son vectores de error, dado que es una resta muestra a muestra de dos arreglos que se definen como:

$$\text{Error} = \text{Salida } \mu C - \text{Salida Matlab}$$

Para llevar a cabo esta verificación, las funciones de verificación reciben como parámetro las entradas de los algoritmos de procesamiento implementados en el μC , así como también la salida de los mismos.

Un diagrama de flujo genérico que representa las funciones para la verificación del procesamiento realizado en el μC se presenta a continuación en la Figura 4.2.

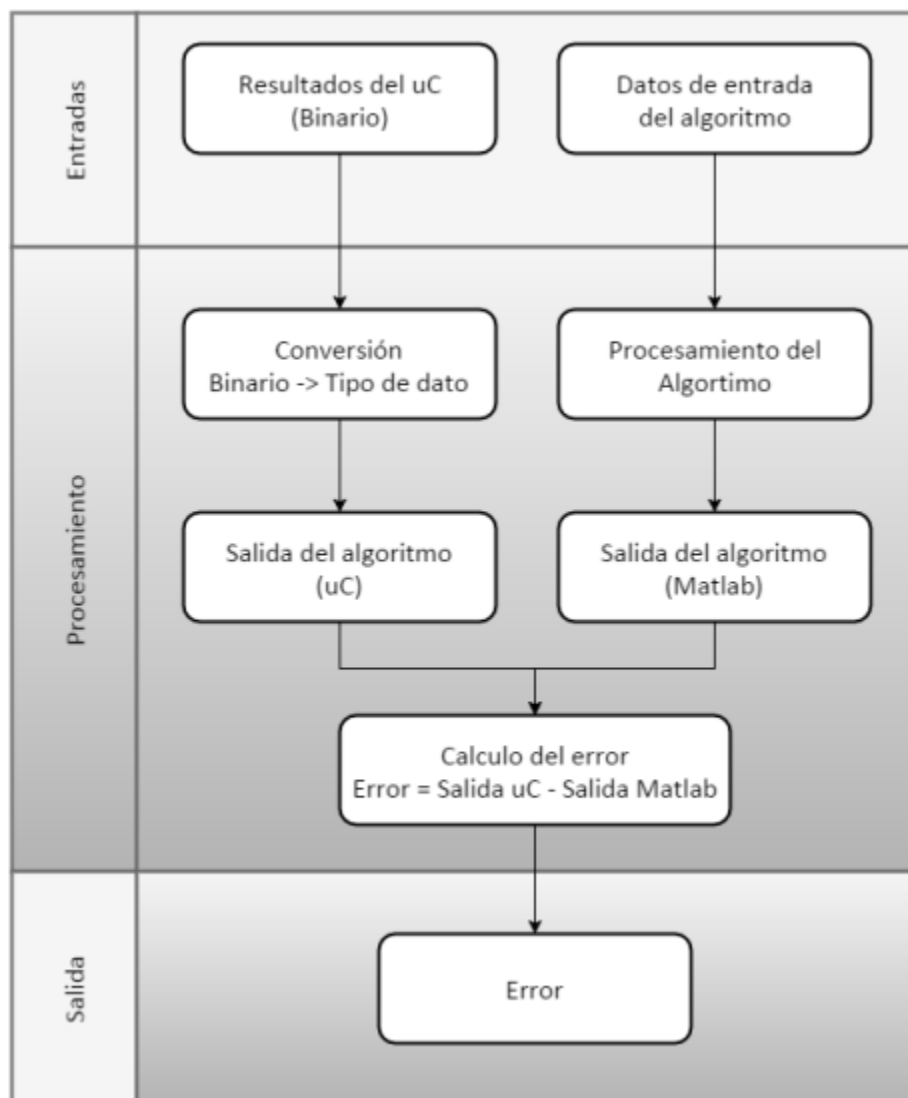


Figura 4.2. Diagrama de flujo esquemático de la verificación del procesamiento implementado.

En el diagrama de flujo se pueden diferenciar 3 partes, las entradas, el procesamiento y la salida. La entrada de las funciones son los resultados del procesamiento llevado a cabo en el μC , así como también las entradas que necesitó el μC para que este procesamiento pueda llevarse a cabo. El procesamiento consiste 3 etapas: llevar a cabo el algoritmo que se implementó en el μC ; convertir el resultado obtenido del μC (datos en binario) a un tipo de dato específico (punto flotante, entero de 32 o 16 bits); y por último, calcular un error en base a los resultados obtenidos en los pasos anteriores.

El esquema de verificación representado en la Figura 4.2 se llevó a cabo para las etapas de Promediación coherente, Extractor de Características, Clasificación y sus variaciones: Promediación coherente de 2 señales, Promediación coherente de 4 señales, Submuestreo 32Hz, Submuestreo 16Hz, Submuestreo basado en ERC y el Clasificador LDA. Cada uno de estos algoritmos para datos tipo punto flotante, enteros 32 y 16-bits.

Cabe mencionar que la verificación de los algoritmos que emplean punto flotante, enteros de 16 bits y enteros de 16 bits utilizando SIMD realizadas en Matlab se realizaron utilizando el mismo tipo de dato que las empleadas en el μC , con el objetivo de que no haya inconvenientes de precisión.

La verificación del procesamiento llevado a cabo con datos tipo punto fijo (q15 y q31) se llevó a cabo utilizando funciones de conversión de estos tipos de datos a tipo punto flotante de precisión simple, provistas en el paquete de CMSIS-DSP. Por lo que se utilizaron las herramientas generadas para punto flotante en esta etapa para estos tipos de datos.

4.3. Algoritmos de procesamiento

4.3.1. Promediación coherente

Para la promediación coherente se utilizaron las funciones de suma y división para cada tipo de dato, incluyendo enteros de 16 bits utilizando SIMD.

4.3.1.1. Promediación coherente – 2 Señales

El modelo de función de promediación coherente de 2 señales se presenta a continuación.

```
void Prom_Coherente_2_signals_TipoDeDato (    DataType *Psen_1,
                                              DataType *Psen_2,
                                              DataType *Pdest,
                                              uint32_t blockSize )
{
    /*Suma de señales*/
    arm_add_DataType (Psen_1, Psen_2, Pdest, blockSize);

    /*Escalado de la señal.
    /*El valor de factor_shift es de -1 para dividir el arreglo por un factor de 2 */
    arm_shift_TipoDeDato (Pdest, scale_factor_shift, Pdest, blockSize);

    /*arm_scale_f32 (Pdest, factor_scale, Pdest, blockSize);
    Si el tipo de dato es punto flotante se utiliza esta función en lugar de
    "arm_shift_TipoDeDato". Con un factor_scale de 0.5*/
```


}

La descripción se divide por partes de la función en: Nombre de la función (en negrita), parámetros de entrada (azul), definición de la función (rojo) y comentarios en la función (verde)

Nombre de la función

Prom_Coherente_2_signals_TipoDeDato: Nombre de la Función

TipoDeDato: Hace referencia al tipo de dato empleado en la operación. Estos pueden ser "f32", "int32", "int16", "q15", "q31" o "int16_SIMD" si el procesamiento es efectuado utilizando tal extensión. Este sufijo es colocado al final del nombre de la función para poder reconocer que tipo de datos utiliza como parámetros de entrada.

Parámetros de entrada

DataType: Tipo de dato empleado reconocido por el lenguaje de programación. Estos pueden ser float32_t, int32_t, int16_t, q15_t o q31_t

uint32_t: es el formato utilizado para especificar la longitud de la señal (entero sin signo de 32 bits)

Psen_1: Puntero a la dirección de memoria del primer elemento de la señal_1 (entrada)

Psen_2: Puntero a la dirección de memoria del primer elemento de la señal_2 (entrada)

Pdest: Puntero a la dirección de memoria del primer elemento de la señal de destino (salida)

blockSize: Longitud de la señal

Por ejemplo, para la declaración de una función de promediación coherente de 2 señales para enteros de 32 bits sería:

```
void Prom_Coherente_2_signals_int32 (    int32_t *Psen_1,
                                         int32_t *Psen_2,
                                         int32_t *Pdest,
                                         uint32_t blockSize)
```

Definición de la función

arm_add_TipoDeDato: Función de suma de dos arreglos de datos. En orden, los parámetros de entrada son: un puntero al primer elemento del primer arreglo (Psen_1), un puntero al primer elemento del segundo arreglo (Psen_2), un puntero al primer elemento del arreglo destino de la suma (Pdest), y el tamaño de los arreglos (blockSize).

arm_shift_TipoDeDato: Función que desplaza una representación binaria de un número (entrada) un "scale_factor" de veces, esto implica la multiplicación o división del número por una potencia de dos ($2^{\text{factor_shift}}$). En orden, los argumentos de esta función son: Puntero al primer elemento del arreglo a desplazar (en este caso Pdest), el factor de desplazamiento, puntero al primer elemento del arreglo de destino (en este caso Pdest) y la longitud del arreglo (blocksize). Esta función se utilizó para dividir los elementos de un arreglo por 2.

factor_shift: Factor de escalamiento utilizado en la función "arm_shift_TipoDeDato", el valor que toma es un entero de 8 bits (int8_t). Si este factor es positivo el número binario se desplaza a la izquierda (multiplicación) y si es negativo el número se desplaza a la derecha (división). Toma el valor -1 para dividir el arreglo por 2.

La función “**arm_shift_TipoDeDato**” se utiliza siempre y cuando el tipo de dato sea punto fijo. Se recuerda que los tipos de dato enteros son una clase de formato punto fijo, como se vio en la sección 2.4.1.3. Sin embargo, esta función no se puede utilizar cuando el tipo de dato es de punto flotante, dada su forma de representación binaria por el estándar IEEE-754 (ver sección 2.4.1.3.3).

Por lo tanto, al implementar la promediación coherente con datos de tipo punto flotante se utiliza la función “**arm_scale_f32**”. Los parámetros de entrada son los mismos que para “**arm_shift_TipoDeDato**”, con excepción del “factor_shift”, esta función tiene un factor que denominamos “**factor_scale**”. Este parámetro de entrada es la constante a la cual se va a multiplicar la señal de interés, en este caso el factor_scale tiene un valor de 0,5.

Cabe aclarar que, se utiliza una función de escalamiento porque el CMSIS no presenta funciones para la división de una señal por un determinado valor.

Tanto en el uso de las funciones “**arm_scale_f32**” o “**arm_shift_TipoDeDato**” las funciones tienen como parámetros 2 veces a Pdest (como entrada y como salida), esto se debe a que tanto la señal de entrada como de salida de esta función es la misma, de manera de optimizar el uso de memoria RAM.

Cabe destacar que la función que corresponde a la promediación coherente de enteros de 16 bits (Prom_Coherente_2_signals_int16_SIMD) tiene una ligera diferencia, dado que la instrucción que genera este procesamiento en el microprocesador almacena la suma de dos arreglos de 16 bits en un arreglo de 32 bits. Por lo tanto la declaración de la función sería la siguiente.

```
void Prom_Coherente_2_signals_int16_SIMD(    int16_t *Psen_1,
                                             int16_t *Psen_2,
                                             int32_t *Pdest,
                                             uint32_t blockSize )
```

4.3.1.2. Promediación coherente – 4 Señales

Si la promediación coherente se realiza con 4 señales el modelo de la función es el que se presenta a continuación:

```
void Prom_Coherente_4_signals_TipoDeDato(    DataType *Psen_1,
                                             DataType *Psen_2,
                                             DataType *Psen_3,
                                             DataType *Psen_4,
                                             DataType *Pdest,
                                             uint32_t blockSize )
{
    /*Suma de señales*/
    arm_add_DataType (Psen_1, Psen_2, Pdest, blockSize);
    arm_add_DataType (Pdest, Psen_3, Pdest, blockSize);
    arm_add_DataType (Pdest, Psen_4, Pdest, blockSize);
    /*Escalado de la señal.
    /*El valor de factor_shift es de -2 para dividir el arreglo por un factor de 4 */
    arm_shift_TipoDeDato (Pdest, scale_factor_shift, Pdest, blockSize);
```

```

/*arm_scale_f32 (Pdest, factor_scale, Pdest, blockSize);
Si el tipo de dato es punto flotante se utiliza esta función en lugar de
"arm_shift_TipoDeDato". Con un factor_scale de 0.25*/
}

```

Donde se respetan todas las referencias establecidas en la promediación coherente de 2 señales. Se observa que en la etapa de suma de señales, ésta se acumula en el arreglo de destino (salida), manteniendo la intención de minimizar el consumo de memoria RAM. Luego, el resultado de la suma se divide con el método correspondiente según el tipo de dato utilizado ("arm_scale_f32" o "arm_shift_TipoDeDato").

4.3.2. Extractor de características

Para la implementación de una etapa de extracción de características de la señal se emplearon funciones de submuestreo para cada tipo de dato. Cabe aclarar que para este procesamiento no se utiliza herramientas SIMD, dado que el submuestreo no implica procesamiento aritmético sobre las señales.

4.3.2.1. Submuestreo Equitemporal

4.3.2.1.1. Submuestreo 32Hz

Teniendo en cuenta que la frecuencia de muestreo de la señal es de 64 Hz, el modelo de la implementación del submuestreo equitemporal a 32 Hz se presenta a continuación:

```

void Submuestreo_32Hz_TipoDeDato (   DataType *Psen,
                                     DataType *Pdest,
                                     uint32_t blockSize )
{
    uint32_t i, aux;
    aux= blockSize/2;
    for (i=0; i<aux; i++)
    {
        *Pdest = *(Psen_1 + i*2);
        Pdest += 1u;
    }
}

```

La descripción se divide por partes de la función en: Nombre de la función (en negrita), parámetros de entrada (azul) y la definición de la función (rojo).

Nombre de la función

Submuestreo_32Hz_TipoDeDato: Nombre de la Función

TipoDeDato: Hace referencia al tipo de dato empleado en la operación. Estos pueden ser "f32", "int32", "int16", "q15" o "q31"

Parámetros de entrada

DataType: Tipo de dato empleado reconocido por el lenguaje de programación. Estos pueden ser float32_t, int32_t, int16_t, q15_t o q31_t

uint32_t: es el formato utilizado para especificar la longitud de la señal

Psen: Puntero a la dirección de memoria del primer elemento de la señal a submuestrear (entrada).

Pdest: Puntero a la dirección de memoria del primer elemento de la señal de destino (salida).

blockSize: Longitud de la señal.

Por ejemplo, para la declaración de una función de submuestreo a 32 Hz para punto flotante sería:

```
void Submuestreo_32Hz_f32 (float32_t *Psen,  
                           float32_t *Pdest,  
                           uint32_t blockSize)
```

Definición de la función

Psen apunta a una determinada posición de memoria, el cual denominaremos **puntero base de la señal a submuestrear**, donde se le suma a esa dirección de memoria un offset dado por $i*2$. En cada iteración se asigna el valor del a la que apunta el puntero base de la señal a submuestrear más el offset ($i*2$) a una posición de memoria indicada por Pdest, que oficia de **puntero base de la señal submuestreada**. Luego de cada asignación, la dirección de memoria es aumentada en 1 de manera tal que en la siguiente iteración el valor sea asignado a una dirección de memoria consecutiva. Por lo que en la ejecución del bucle se asignan los valores de cada 2 posiciones de memoria de la señal a submuestrear a cada valor consecutivo de la señal de destino (salida). De esta manera se obtiene una señal de la mitad de longitud de la señal de entrada.

Si la longitud de la señal a submuestrear es impar (tamaño N), el tamaño del arreglo de salida (salida) es de $(N-1)/2$.

4.3.2.1.2. Submuestreo 16Hz

El modelo de la implementación del submuestreo equitemporal a 16 Hz se presenta a continuación:

```
void Submuestreo_16Hz_TipoDeDato (    DataType *Psen,
                                     DataType *Pdest,
                                     uint32_t blockSize )
{
    uint32_t i, aux_entero, aux_resto;
    aux_resto=0;
    aux_entero= blockSize/4;
    aux_resto= blockSize%4;
    if (aux_resto)
    {
        aux_entero+=1;
    }
    for (i=0; i<aux_entero; i++)
    {
        *Pdest = *(Psen_1 + i*4);
        Pdest += 1u;
    }
}
```

En donde se respeta las mismas referencias y se utiliza la misma estrategia que para el submuestreo a 32 Hz, con la diferencia que los datos del arreglo a submuestrear se toman cada 4 posiciones de memoria en lugar de cada 2.

Otra diferencia es que en este caso, si la longitud del arreglo a submuestrear(N) presenta un resto distinto de cero al dividirlo por 4, el arreglo de salida (salida) tiene longitud $N/4+1$. Esto se debe a que en las señales obtenidas del grupo de trabajo del LIRINS, las señales de longitud de 450 submuestreadas a 16 Hz tenían 113 muestras, por lo que este algoritmo se adaptó para tener la misma longitud y no tener inconvenientes en las verificaciones posteriores.

4.3.2.2. Submuestreo ERC

Como hemos visto anteriormente en la sección 2.3.2, el submuestreo ERC se basa en submuestrear la señal con un determinado criterio.

El modelo de la implementación de submuestreo ERC se presenta a continuación:

```
void Submuestreo_ERC_TipoDeDato (DataType *Psen_1,
                                DataType *Pdest)
{
    uint32_t i;
    for (i=0;i<Tam_Sen_ERC_TipoDeDato; i++)
    {
        *(Pdest + i) = *(Psen_1 + Indices_ERC_TipoDeDato [i]) ;
    }
}
```

La descripción se divide por partes de la función en: Nombre de la función (en negrita), parámetros de entrada (azul) y la definición de la función (rojo).

Nombre de la función

Submuestreo_ERC_TipoDeDato: Nombre de la Función

TipoDeDato: Hace referencia al tipo de dato empleado en la operación. Estos pueden ser "f32", "int32", "int16", "q15" o "q31".

Parámetros de entrada

DataType: Tipo de dato empleado reconocido por el lenguaje de programación. Estos pueden ser float32_t, int32_t, int16_t, q15_t o q31_t

Psen: Puntero a la dirección de memoria del primer elemento de la señal a submuestrear (entrada).

Pdest: Puntero a la dirección de memoria del primer elemento de la señal de destino (salida).

Por ejemplo, para la declaración de una función de submuestreo ERC para punto flotante de 32 bits sería:

```
void Submuestreo_ERC_q31 (q31_t *Psen,
                           q31_t_t *Pdest)
```

Definición de la función

i: Valor utilizado en el bucle, aumenta en cada iteración.

Tam_Sen_ERC_TipoDeDato: Valor que indica la longitud que debe tener la señal submuestreada, el cual es utilizado en la condición de salida del bucle. Este valor se almacena en memoria no volátil.

Indices_ERC_TipoDeDato: Arreglo que contiene índices, que contiene las características más relevantes de la señal. Su longitud es de "Tam_Sen_ERC_TipoDeDato". Este arreglo se almacena en memoria no volátil.

En este algoritmo se utiliza la misma estrategia que en el submuestreo equitemporal, con la diferencia que el offset del puntero base de la señal a submuestrear en este caso es el valor indicado en el arreglo de índices (Indices_ERC_TipoDeDato). De esta manera el algoritmo selecciona elementos relevantes de la señal y los almacena en el arreglo de salida (salida).

4.3.3. Clasificador LDA

Para la clasificación lineal se utilizaron funciones de producto punto para cada tipo de dato, incluyendo enteros de 16 bits utilizando SIMD.

El modelo de implementación de clasificador lineal se presenta a continuación:

```
Void Clasif_TipoDedato (   DataType *Psen,
                          DataType *resultado,
                          DataType *clasif )
{
dot_prod_TipoDeDato (Psen, Pesos_TipoDeDato, Tam_Sen, resultado);

if (*resultado < Umbral_TipoDeDato)
{   *clasif=0;   }
else
{   *clasif=1;   }

}
```

La descripción se divide por partes de la función en: Nombre de la función (en negrita), parámetros de entrada (azul), definición de la función (rojo).

Nombre de la función

Clasif_TipoDedato: Nombre de la Función

TipoDeDato: Hace referencia al tipo de dato empleado en la operación. Estos pueden ser “f32”, “int32”, “int16”, “q15”, “q31” o “int16_SIMD” si el procesamiento es efectuado utilizando tal extensión

Parámetros de entrada

DataType: Tipo de dato empleado reconocido por el lenguaje de programación. Estos pueden ser float32_t, int32_t, int16_t, q15_t o q31_t

uint32_t: es el formato utilizado para especificar la longitud de la señal

Psen: Puntero a la dirección de memoria del primer elemento de la señal a clasificar (entrada)

resultado: Puntero a la dirección de memoria donde se almacena el resultado del producto punto (salida).

clasif: Puntero a la dirección de memoria del donde se almacena el resultado de la clasificación (salida).

Definición de la función

dot_prod_TipoDeDato: Función que realiza el producto punto de 2 vectores. Los parámetros de entrada en orden son: un puntero al primer elemento del primer arreglo

(Psen_1), un puntero al primer elemento del segundo arreglo (Pesos_TipoDeDato), la longitud de la señales (Tam_Sen) y el resultado del producto punto (resultado).

Umbral_TipoDeDato: Es un valor de umbral el cual se compara con el resultado del producto punto. Este valor se almacena en memoria no volátil.

Esta función realiza un producto punto entre una señal de EEG con un arreglo de pesos calculados en la etapa de investigación correspondiente. El resultado es comparado con un umbral determinado para determinar la detección del P300 en la señal de EEG o no. Para ello, esta función tiene como salida la variable “**clasif**” que actúa de bandera “Si/No”. A su vez, devuelve el valor del resultado del producto punto por referencia en caso de ser requerido para un posterior procesamiento.

Capítulo 5: Resultados

5.1. Verificaciones

En la verificación de la Promediación coherente y de la Extracción de Características, se obtiene un vector error. Dado que, según lo mencionado en la sección 4.2, en la verificación de estos algoritmos de procesamiento, el error es el resultado de una resta de vectores productos de un determinado procesamiento en el μC y del mismo procesamiento realizado en Matlab.

Por otro lado, en la verificación de los Clasificadores se obtienen sólo 2 valores: el resultado del producto punto y un valor que indica la detección o no del potencial P300 (se reporta cero si no se detecta y un valor distinto a cero si se detecta).

5.1.1. Promediación coherente

Para los datos de tipo punto flotante el error dio igual a cero en los algoritmos de promediación de 2 y 4 señales.

Para los datos de tipo enteros (int32, int16 e int16_SIMD) el error máximo encontrado es de -1. A su vez, este es el único valor distinto de cero encontrado para estos tipos de dato.

Para los enteros de 32 bits, este error representa un $-2,33 \times 10^{-10} \%$ de error considerando todo el rango de representación de este tipo de dato. Para enteros de 16 bits, el error representa un $-1,52 \times 10^{-05} \%$ para todo su rango de representación.

Este error se debe a que en la implementación de la Promediación coherente para estos tipos de datos, se utiliza la función "arm_shift_TipoDeDato" que, desplaza los números binarios a la derecha para efectuar la división. Este procesamiento produce el resultado esperado siempre y cuando los números sean pares. Si los números son impares, esta división tiene efectos distintos dependiendo si el número en cuestión es positivo o negativo. Si es negativo el número se redondea alejándose del cero, es decir, aumentando el módulo del resultado; y si es positivo el número se redondea acercándose del cero, es decir disminuyendo el módulo. En otras palabras, esta división siempre redondea "hacia la izquierda".

En la Figura 5.1 se representa un ejemplo de desplazamiento de un registro de 8 bits en el cual se realiza un desplazamiento sobre el número 7 y -7.

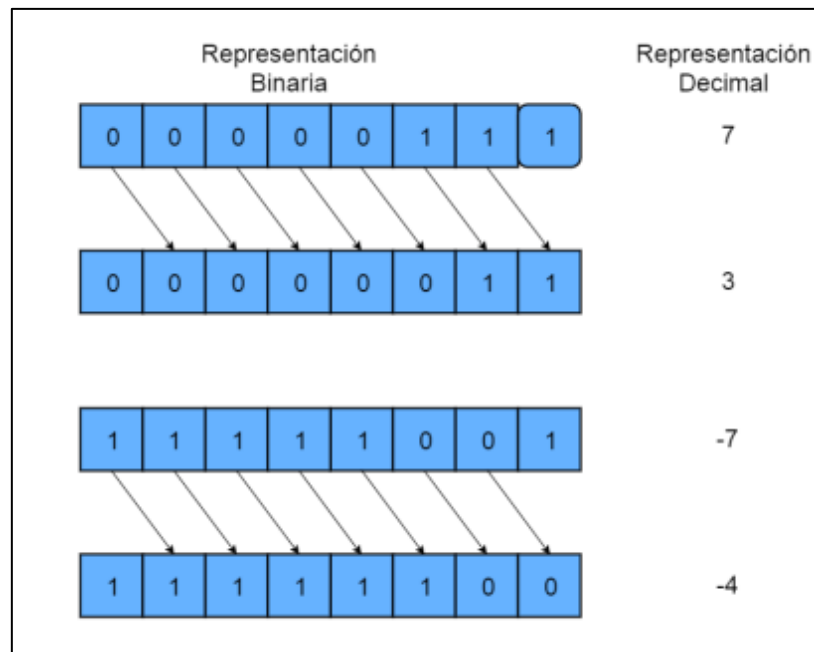


Figura 5.1. Ejemplo de cómo influye el desplazamiento binario en un número impar positivo y negativo.

Por otro lado, en el proceso de verificación, Matlab al convertir un número con decimales a entero, éste aumenta el módulo del dígito en cuestión hasta el entero más próximo. Por ejemplo: `int32 (2,3)` es igual a 3 o `int32 (-4,6)` es igual a -5. Lo cual es igual al comportamiento que posee la división implementada para números negativos, pero no para positivos.

Por lo tanto, se tiene la hipótesis que los valores los cuales se reportan los errores serían positivos e impares debido a que éstos el microcontrolador los redondea hacia el entero más próximo en dirección hacia el cero.

Para corroborar esto, se procedió de la siguiente manera:

1. Se extrajeron del vector de error los índices de los valores que reportaban error.
2. Se calcularon los valores de la promediación antes de la división.
3. Se seleccionaron los valores del vector antes de la división utilizando los índices extraídos anteriormente
4. Se confirmó que todos valores eran únicamente positivos e impares.

La verificación de la promediación coherente de 2 señales de los datos tipo q31 de arrojan solo 10 valores (de 6750 muestras en el vector de error) distintos de cero, de los cuales el error máximo por exceso registrado es $9,3132 \times 10^{-10}$ y por defecto $-9,3132 \times 10^{-10}$. Para la promediación coherente de 4 señales se encontraron 436 valores distintos a cero.

El error presente a este tipo de dato se debe a redondeos y truncamientos en la conversión de single a q31 para realizar el procesamiento y de q31 a single para enviar los datos a la PC. Esto se corrobora en la verificación del extractor de características.

5.1.2. Extractor de Características

Los errores registrados para el submuestreo equitemporal y ERC para datos tipo punto flotante, enteros de 16 y 32 bits son iguales a cero. No así para los tipos de dato q31 y q15.

Los submuestreos de tipos de dato q31 arrojan errores máximos por defecto de $-4,3656 \times 10^{-10}$ y $4,3656 \times 10^{-10}$ por exceso. Presentan 10 y 7 valores con error distinto de cero para el submuestreo equitemporal de 32 y 16 Hz respectivamente.

El error para el caso del submuestreo de tipo de datos q15, arroja valores máximos por defecto de $-3,0413 \times 10^{-5}$ y $3,0510 \times 10^{-5}$ por exceso y prácticamente todos los valores en el vector de error sin distintos a cero.

Según se mencionó en la sección 2.4.3.1, la precisión de los datos de tipo punto fijo q31 es de $4,6566 \times 10^{-10}$ y para los q15 es de $3,0517 \times 10^{-5}$. Por lo que los errores máximos encontrados se encuentran en el rango de error mínimo posible en este tipo de representación.

Cabe destacar que la cantidad de errores encontrados en los de tipo q31 es significativamente menor, así como también en módulo respecto a las de q15. Esto se debe a que los errores de redondeo y truncamiento se dan en los bits menos significativos de los registros empleados. Al castear un tipo dato de 32 bits a otro tipo de dato de 32 bits, estos errores son más pequeños que al castear un dato de 16 bits a uno de 32 bits, dado que la posición de los bits menos significativos de un registro de 16 bits no son los menos significativos en uno de 32 bits.

Al no realizarse procesamiento aritmético en esta etapa y al obtenerse errores del orden de los obtenidos en la promediación coherente y clasificación, como veremos más adelante, se puede inferir que el error presente en el procesamiento llevado a cabo con los datos de tipo q31 y q15 definitivamente se deben a errores de redondeo y truncamiento en las conversiones llevadas a cabo.

5.1.3. Clasificador LDA

Los errores registrados para todos los clasificadores LDA para datos tipo punto flotante, enteros de 16 y 32 bits son iguales a cero. Los valores que informan la detección del P300 son todos correctos. El error reportado para los clasificadores LDA para int16 utilizando SIMD es de 1 (error de $1,52 \times 10^{-05} \%$).

Los errores absolutos y relativos, a todo su rango de representación (de -1 a ≈ 1), para los tipos de datos q15 y q31 se detallan a continuación en la Tabla 5.1.

Tipo de dato	LDA		ERC - 1 canal		ERC - 10 canales	
	Absoluto	Relativo	Absoluto	Relativo	Absoluto	Relativo
q15	-0,5958	-29,79%	-0,2316	-11,58%	0.0269	1,35%
q31	0,2185	10,93%	0,7858	39,29%	0,9375	46,88%

Tabla 5.1. Errores absolutos y relativos reportados por las verificaciones para los Clasificadores que utilizan datos de tipo q15 y q31.

Como se puede ver en los errores relativos de la Tabla 5.1, los errores que este tipo de dato presentan son inaceptables para realizar este tipo de procesamiento.

Se estima que la fuente de este error para q31 se debe a la implementación de la función del producto punto este tipo de dato provista por CMSIS. El algoritmo almacena el resultado de los productos y sumas en un acumulador de 64 bits (q63_t) y en cada

iteración el acumulador desplaza (mediante shift) el valor acumulado hacia la derecha 14 veces, lo cual es equivalente a dividir el número por 2^{14} (16384).

No se tiene en claro el objetivo este desplazamiento dentro de la función, por lo que antes de continuar con la verificación se modificó tal rutina, de manera tal de evitar el desplazamiento, con el objetivo de verificar si el error disminuía a valores aceptables. Sin embargo, al implementar estos cambios a pesar de haber disminuido el error, éste se encontraba lejos de ser aceptable. Por este motivo se reporta el error de la rutina sin modificar.

Otro inconveniente con este procesamiento es el de poder reportar de manera eficiente el procesamiento dado que el dato se almacena en un registro de 64 bits de tipo q63, y no hay herramienta que permita la conversión de este tipo de dato a punto flotante, q31 o q15 provista por CMSIS. Por lo que se implementaron dos maneras de obtener estos valores:

- 1) Casteando el valor de q63 a q31
- 2) Utilizando union¹⁰, para poder extraer el número en binario en bloques de 32 bits.

Lo cual se llega a la conclusión de que castear el número y de q63 a q31 y extraer los 32 bits más significativos del registro de 64 bits da el mismo resultado. Este valor fue convertido a punto flotante, y éste valor fue el que se tomó como salida del algoritmo.

En conclusión, La gran mayoría de los algoritmos se implementaron correctamente, quedando demostrado con los resultados de esta etapa. Se destaca la necesidad de investigar la criticidad del error de una unidad reportado al emplear como división el shift, así como también el error reportado por el clasificador LDA utilizando enteros de 16 bits mediante SIMD.

Aun reportando el error importante en los clasificadores de tipo q31 y q15 se decidió realizar las mediciones de los recursos que utilizan los mismos para tener información del consumo de recursos que éstos demandan.

5.2. Mediciones de parámetros

5.2.1. Tiempo

En esta sección se presentan los resultados de las mediciones del tiempo de ejecución de los algoritmos que conforman el procesamiento de una ICC.

Dado a la cantidad de datos generados, en el Anexo N°1 “Tiempos de ejecución de algoritmos en función de la longitud de señal y el tipo de dato” se presentan los 218 resultados de todas las variaciones de procesamiento evaluados, descritas en la sección 2.3.4.

Los resultados reportados en el Anexo 1 son tiempos máximos de ejecución de procesamiento medidos en microsegundos. Tales valores reportados son producto del análisis de repetibilidad de los tiempos de ejecución que se explica en la siguiente sección.

¹⁰ En C y C++, las uniones son expresados casi igual que los struct excepto que cada miembro empieza en la misma dirección de memoria. Por lo que el tamaño de la variable tipo unión es el tamaño que tenga el tipo de dato de más grande en ella. Esta estructura permite almacenar elementos de diferentes tipos en las mismas posiciones de memoria

5.2.1.1. Repetibilidad

Acorde a lo señalado en la sección 3.2.1.3, para las mediciones temporales era necesario un estudio de repetibilidad de los mismos. Cada medición se llevó a cabo 1000 (mil) veces, lo cual denominamos como una serie de mediciones.

En la Figura 5.2 se presenta un ejemplo de los resultados de una serie de mediciones.

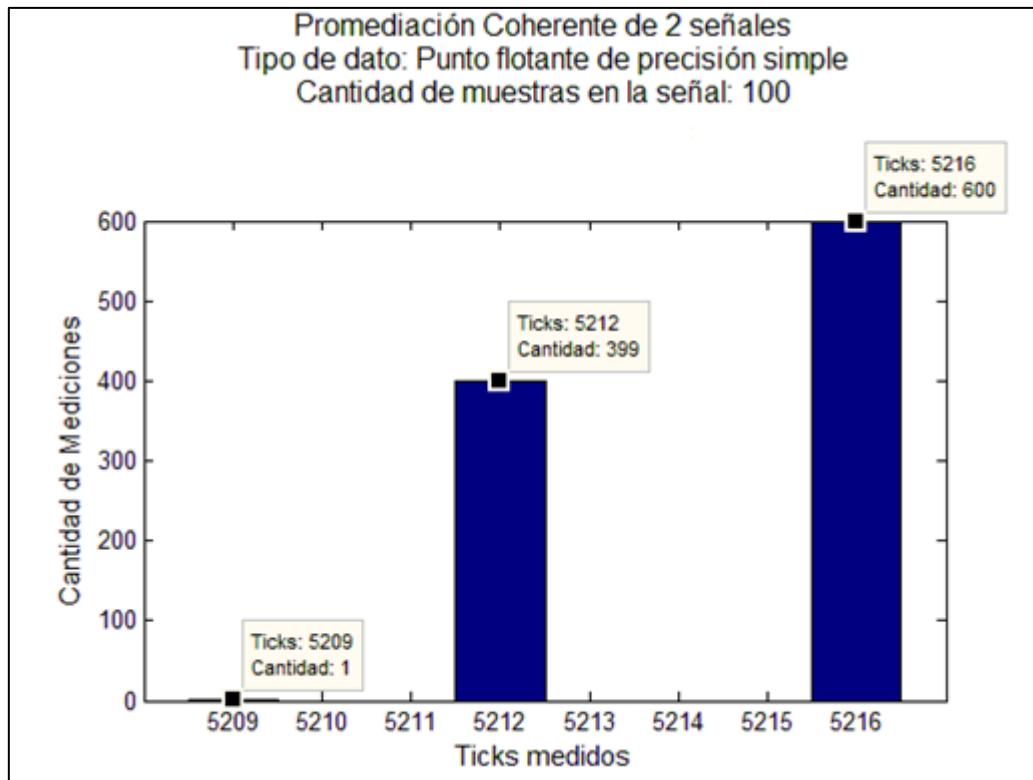


Figura 5.2. Histograma que presenta el resultado de la medición de tiempo (ticks) de una promediación Coherente de 2 señales de una señal de 100 muestras y tipo de dato single.

El tiempo fue medido en ciclos de reloj, denominados en este trabajo como “ticks”. Se recuerda lo mencionado en la sección 3.2.1 que 204 ticks (ciclos de reloj) equivalen a 1 microsegundo.

Dado que algunos de los resultados de las series de mediciones presentan dispersiones, si presentáramos cada histograma, tendríamos 218 de estos gráficos solo para esta sección. Por lo que se decidió, en principio, analizar los resultados de manera global de tener un panorama general de los resultados.

De todas las series de mediciones llevadas a cabo se analizó el rango de dispersión máxima presentan. El rango máximo de dispersión se calculó restando el valor mínimo al valor máximo de ticks medidos en cada serie realizada.

$$\text{Rango máximo de la serie } N = \text{Ticks máximo de la serie } N - \text{Ticks mínimo de la serie } N$$

Donde N es un entero de 1 a 218.

La Figura 5.3 ilustra la dispersión de rangos máximos en todas las mediciones realizadas.

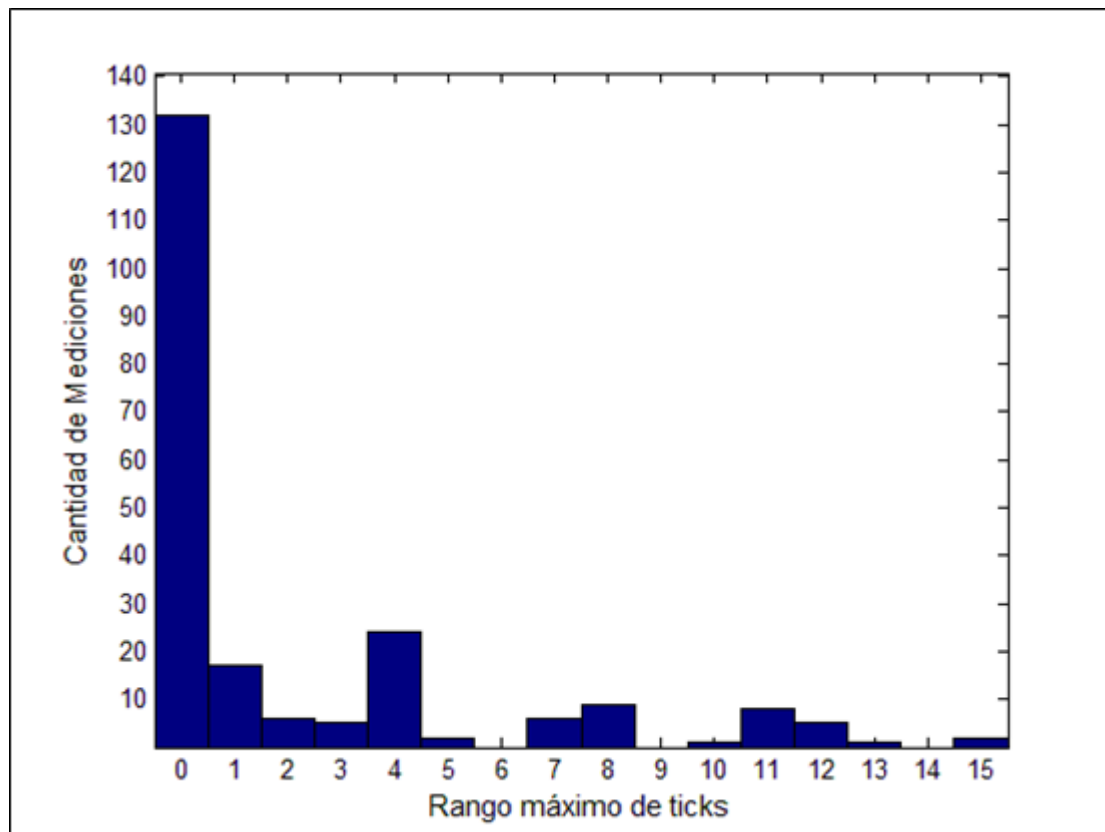


Figura 5.3. Distribución de rangos máximos de todas las mediciones (N=218)

Como se puede observar en la el histograma de la Figura 5.3, las mediciones presentan dispersión. Estudiando los datos podemos encontrar que:

- 60,55% presenta un rango máximo igual a cero, lo que significa que no hay dispersión en los resultados.
- 22,93% presentan un valor aislado. Denominamos valores aislados a los resultados en la medición que sólo se presentan una única vez en toda la serie, y generalmente suele ser la cota inferior en el rango máximo (valor mínimo), colaborando a que se aparente una distribución más amplia en los resultados.
- El 85, 32% de las variaciones se encuentran en un rango máximo de 0 a 5 ticks
- Las variaciones máximas registradas (2 de 218) presentan un rango máximo de 15 ticks.

Como se puede ver en la Figura 5.4, la dispersión la aporta mayoritariamente las series de las Promediaciones coherentes, tomando valores de rangos máximos entre 0 y 15. Los extractores de características aportan rangos máximos entre 0 y 2 (Figura 5.5); y los clasificadores aportan rangos máximos únicamente de 0 y 1 (Figura 5.6).

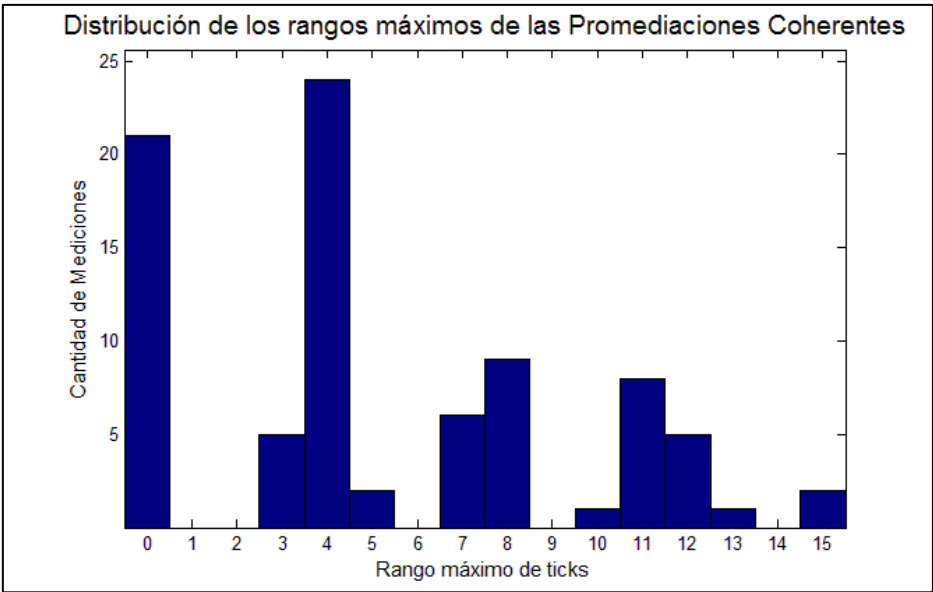


Figura 5.4. Histograma de rangos máximos de las promediaciones coherentes.

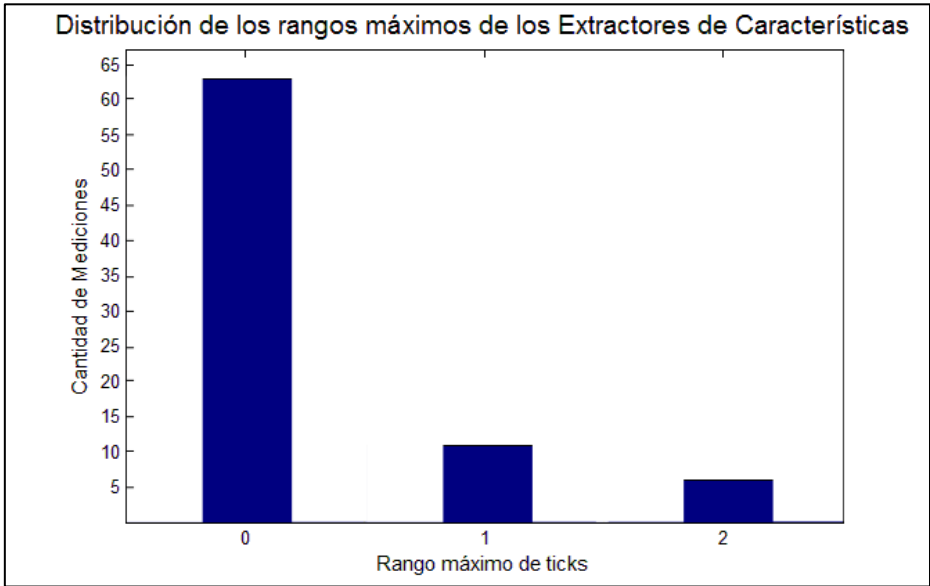


Figura 5.5. Histograma de rangos máximos de los extractores de características.

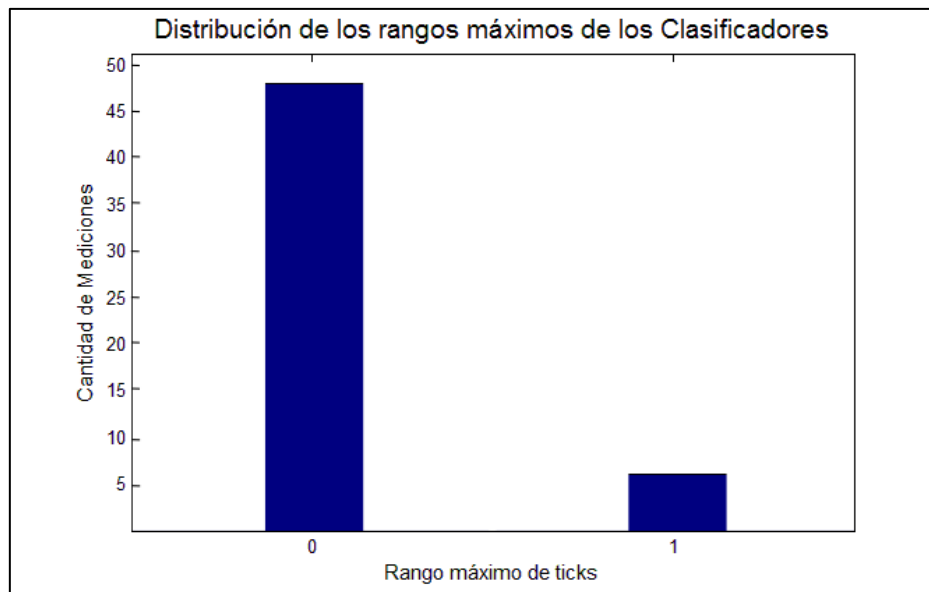


Figura 5.6. Histograma de rangos máximos de los clasificadores.

El rango máximo de mayor módulo encontrado es de 15 ticks que corresponden a Promediaciones coherentes de 2 señales, con una longitud de señal de 200 muestras y con tipo de datos de 16 bits con y sin el uso de SIMD (Figuras 5.7 y 5.8). Esta dispersión de 15 ticks equivale aproximadamente a 74 nanosegundos y es el rango más desfavorable de todos los casos medidos, que incluye un valor aislado, caracterizado anteriormente, que aporta 2 ticks a esta dispersión.

Por otro lado, estos 15 ticks de rango máximo representan un error máximo de 0,15% respecto al promedio de ticks medidos en sus correspondientes series.

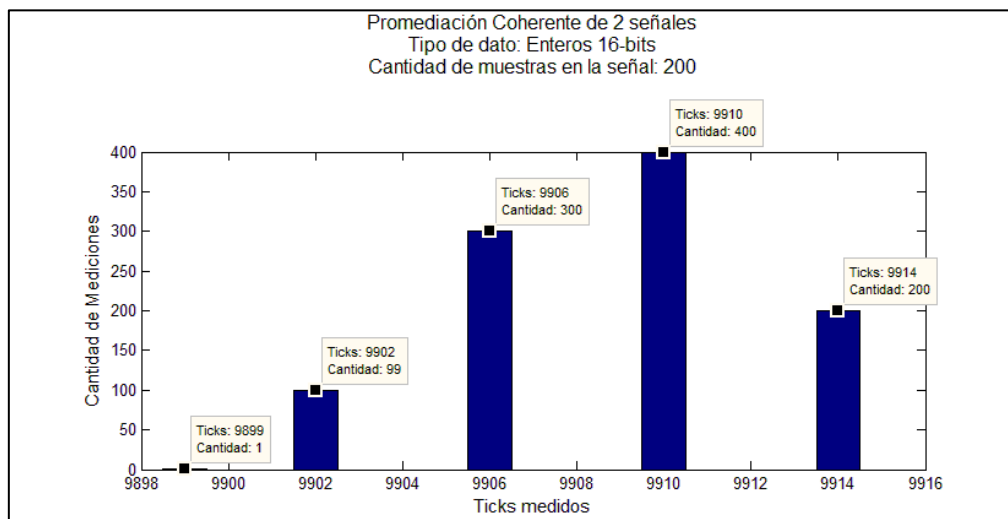


Figura 5.7. Histograma de rangos de la promediación coherente de 2 señales de 200 muestras y datos enteros de 16 bits de señales.

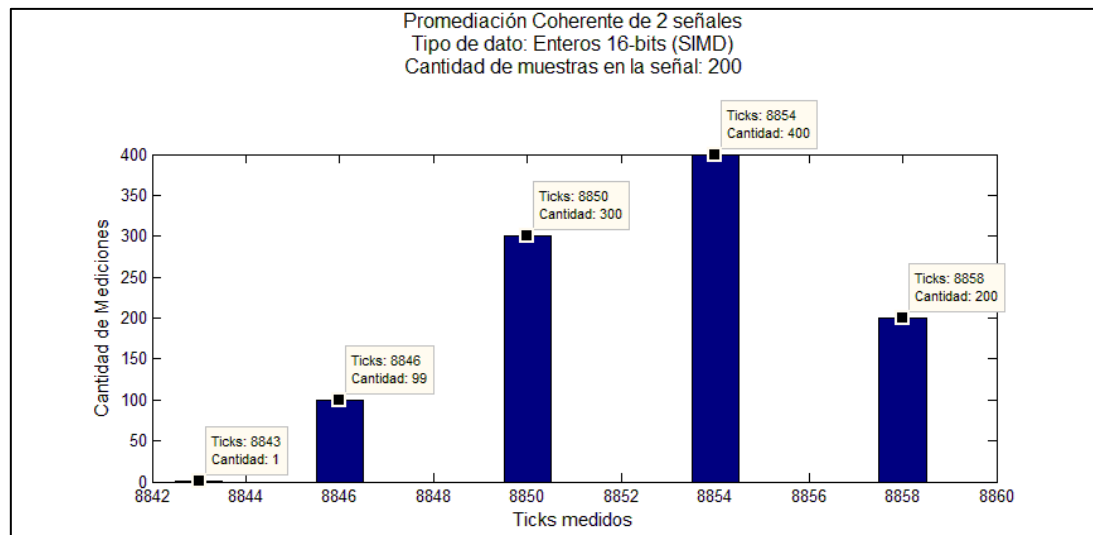


Figura 5.8. Histograma de rangos de promediación coherente de 2 señales con datos enteros de 16 bits (utilizando SIMD) y señales de longitud de 200 muestras.

Se considera que la dispersión de 15 ticks, siendo la condición más crítica encontrada, no aporta un error significativo para la aplicación que se intenta implementar, dado que como se verá más adelante los tiempos de ejecución del procesamiento completo se encuentran en el orden de centenas de microsegundos.

Por lo tanto, en el Anexo N°1 se reportan todos los tiempos medidos en microsegundos con los valores máximos medidos.

5.2.1.2. Tiempo de ejecución Vs. Longitud de la señal

A continuación se presentan algunos de los resultados de las mediciones al variar la longitud de la señal procesada (Figuras 5.9, 5.10 y 5.11).

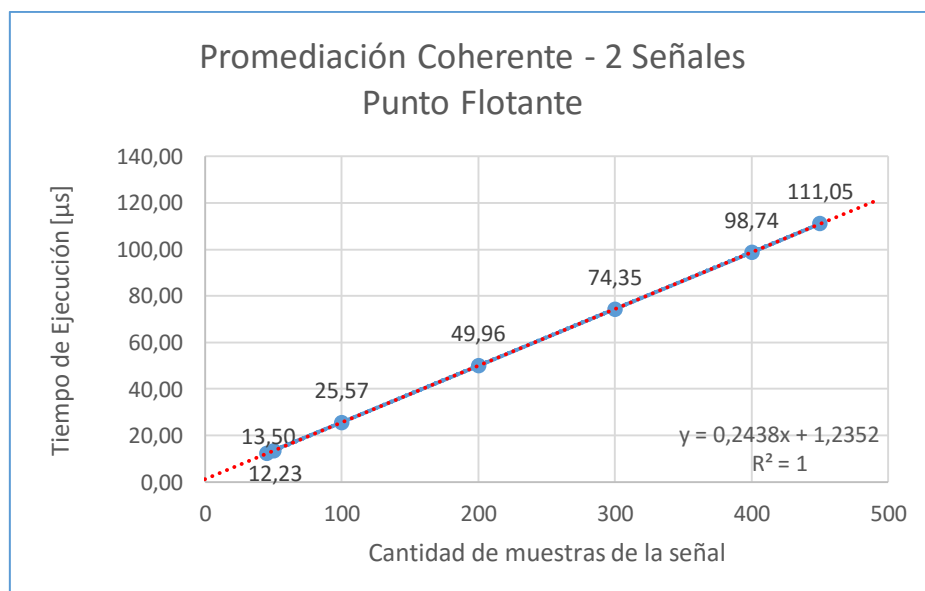


Figura 5.9. Tiempo de ejecución de la promediación coherente de 2 señales de datos tipo punto flotante en función de la cantidad de muestras de la señal.

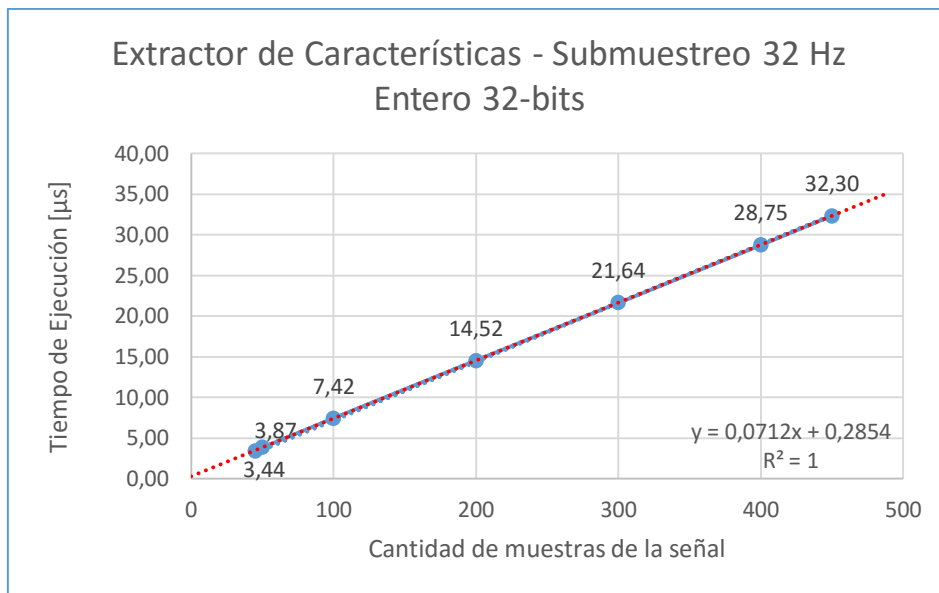


Figura 5.10. Tiempo de ejecución de submuestreo de 32 Hz de datos enteros de 32-bits en función de la cantidad de muestras de la señal.

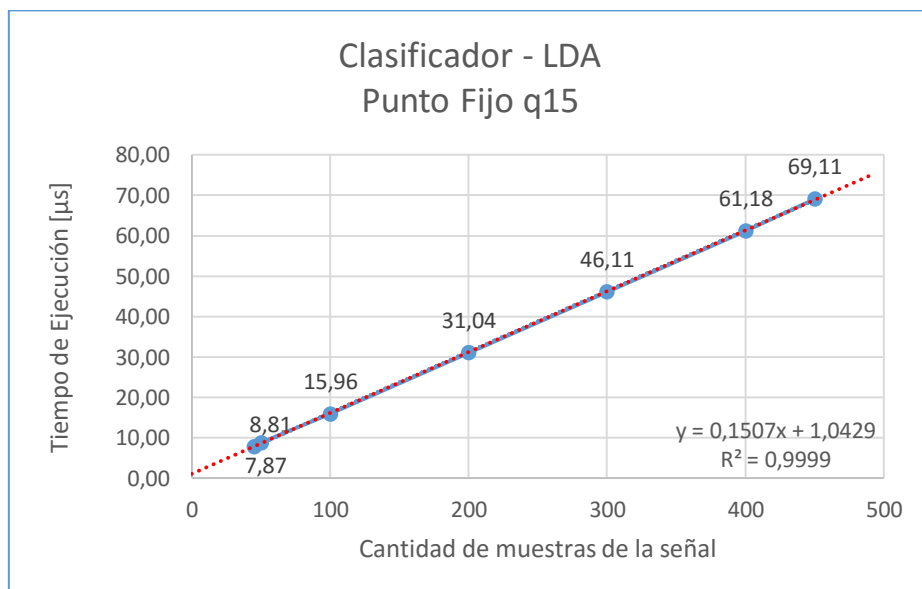


Figura 5.11. Tiempo de ejecución del clasificador LDS en función de la cantidad de muestras de la señal.

Teniendo en cuenta lo mencionado en la sección 2.3.4 respecto a las variaciones de procesamiento implementadas, se tienen 39 tipos de procesamiento que varían en función de la longitud de la señal:

2 x Prom. Coherente x 6 tipos de datos= 12

3 x Submuestreos x 5 tipos de datos= 15

2 x Clasificador x 6 tipos de datos= 12

Se recuerda que el factor “2” en el cálculo de variaciones de procesamiento del clasificador no hace referencia a que haya 2 tipos de clasificadores. Éste hace referencia a que existen variaciones en la longitud de la señal procesada según se haya procesado o no el submuestreo ERC previamente.

De estas 39 variaciones de procesamiento se analizó la linealidad que presentan los mismos en función de la cantidad de muestras contenidas en la señal con el coeficiente R^2 , el cual es un estadístico que indica cuan cercana es la estimación de los valores a la recta de aproximación lineal correspondientes a los datos que se tienen. Este valor oscila entre cero y 1, y mientras más cercano a 1 sea más precisa es la aproximación.

32 de los 39 algoritmos reportaron un R^2 igual a 1; 6 de ellos reportaron un valor de este coeficiente de 0.9999 y uno de ellos 0.9998. Lo cual evidencia una linealidad marcada en el tiempo de ejecución de los algoritmos en función del tamaño de la señal.

Como consecuencia de este análisis se elige una única longitud de señal para poder comparar en el impacto del tiempo de ejecución en función del tipo de dato que se utilice. Se elige una longitud de señal de 100 muestras para ello, dado que es un número con el cual se pueda escalar con mayor facilidad el tiempo de ejecución ante otra cantidad de muestras presentes en la señal.

5.2.1.3. Tiempo de ejecución Vs. Tipo de dato

Dada la cantidad de datos, se ordenaron los resultados de menor a mayor tiempo de ejecución del algoritmo. Cabe mencionar que los valores reportados en esta sección son los máximos medidos.

5.2.1.3.1. Promediación Coherente – 2 Señales

Tipo de Dato	Tiempo de Ejecución [μ s]
int16_SIMD	22,33
int32	24,10
q15	24,48
int16	24,94
Single	25,57
q31	29,81

Tabla 5.2. Tiempo de ejecución de la promediación coherente de 2 señales según el tipo de dato.

5.2.1.3.2. Promediación Coherente – 4 Señales

Tipo de Dato	Tiempo de Ejecución [μ s]
int16_SIMD	44,80
q15	47,00
int32	49,76
Single	52,25
int16	52,39
q31	67,34

Tabla 5.3. Tiempo de ejecución de la promediación coherente de 4 señales según el tipo de dato.

5.2.1.3.3. Submuestreo 32Hz

Tipo de Dato	Tiempo de Ejecución [μ s]
q15	7,17
q31	7,18
int16	7,30
Single	7,42
int32	7,42

Tabla 5.4. Tiempo de ejecución del submuestreo 32 Hz según el tipo de dato.

5.2.1.3.4. Submuestreo 16Hz

Tipo de Dato	Tiempo de Ejecución [μ s]
q15	3,80
q31	3,80
int16	3,94
Single	4,07
int32	4,07

Tabla 5.5. Tiempo de ejecución del submuestreo 32 Hz según el tipo de dato.

5.2.1.3.5. Submuestreo ERC

Dado que sólo se midieron tiempo de ejecución de 2 longitudes de señal para cada tipo de dato de este algoritmo, aprovechando la linealidad que éstos algoritmos presentan, se calculó el tiempo de ejecución de este algoritmo si la señal resultante tuviera 100 muestras.

Tipo de Dato	Tiempo de Ejecución [μ s]
q15	14,78
q31	14,78
Single	15,32
int32	16,22
int16	17,52

Tabla 5.6. Tiempo de ejecución del submuestreo ERC según el tipo de dato.

Cabe mencionar, que para comparar el tiempo de ejecución del submuestreo ERC con los submuestreos equitemporales, se debe tener en cuenta la longitud de la señal a la salida de éstos.

Es decir, Para comparar el submuestreo de 32 Hz, el cual tiene una señal de entrada de 100 muestras y una señal de salida de 50 muestras, con el submuestreo ERC, la salida de éste debe tener 50 muestras. Lo cual sería la mitad de los valores presentados en la Tabla 5.6.

De la misma manera con el submuestreo de 16 Hz, el cual tiene una señal de entrada de 100 muestras y una señal de salida de 25 muestras. Por lo que la comparación con el submuestreo ERC debería hacerse con los valores presentados en la Tabla 5.6 dividido cuatro.

5.2.1.3.6. Clasificador LDA

Tipo de Dato	Tiempo de Ejecución [μ s]
int16_SIMD	10,26
int32	10,81
Single	11,38
int16	13,01
q15	15,96
q31	27,28

Tabla 5.7. Tiempo de ejecución del Clasificador LDA según el tipo de dato.

5.2.1.3.7. Clasificador LDA luego del submuestreo ERC

Tipo de Dato	Tiempo de Ejecución [μ s]
int16_SIMD	10,53
int32	10,92
Single	11,49
int16	12,50
q15	15,50
q31	25,00

Tabla 5.8. Tiempo de ejecución calculado del Clasificador LDA según el tipo de dato.

Los valores reportados en la Tabla 5.8, así como también en los submuestreos ERC, son calculados en base a la linealidad que el procesamiento presenta en función de la longitud de la señal. En este caso se midió el tiempo de ejecución del clasificador para señales que contengan una cantidad de muestras determinadas por el procesamiento submuestreo ERC y luego, con esta información, se calculó cuanto tiempo tarda si la señal tuviera 100 muestras.

Se destaca también que los resultados presentados de los clasificadores LDA medidos (Tabla 5.7) y de los clasificadores LDA calculados (Tabla 5.8) son prácticamente iguales en todos los tipos de datos. Lo que indica que la linealidad que estos resultados presentan en función de la cantidad de muestras de la señal, sirven para poder estimar los tiempos de ejecución para otras longitudes de señal. El cual es un recurso utilizado más adelante.

5.2.1.3.8. 1 canal vs. 10 canales

En esta sección se muestran los resultados del tiempo de ejecución insumido por el procesamiento completo por cada canal (procesamiento de 10 canales y 45 muestras cada señal) y para 10 canales (1 procesamiento de una señal de 450 muestras, que son los 10 canales concatenados).

Para mostrar estos resultados de manera que el lector tenga un panorama general de los recursos temporales que lleva el procesamiento completo, dada todas las variables que se tienen, se optó por tomar la siguiente estrategia: reportar las condiciones límites de las mismas. Esto es, mostrar las situaciones en las cuales se insuma el menor y el mayor tiempo de ejecución. Se consideran las situaciones que empleen algoritmos que hayan tenido resultados satisfactorios en la etapa de verificación de procesamiento.

Por lo tanto, respecto a los tipos de dato, se muestran los resultados de los tiempos de ejecución efectuados con enteros de 16 bits utilizando SIMD y con punto flotante, que son los tipos de dato que se ejecutan más rápido y más lento respectivamente.

Respecto a la combinación de algoritmos para el procesamiento de las señales, se eligieron las combinaciones de algoritmos que insumen mayor y menor tiempo con los tipos de datos mencionados anteriormente.

De esta manera, se determinan los límites temporales que toma ejecutar estos procesamientos con distintos tipos de datos y combinaciones de algoritmos, siempre y cuando se utilice al menos uno de cada tipo de algoritmo.

Combinación de algoritmos con menor tiempo de procesamiento

1 canal

Tipo de dato	Promediación Coherente 2 Señales [μs]	Extractor de Características 16 Hz [μs]	Clasificador LDA [μs]	Total por un Canal [μs]
int16_SIMD	10,95	2,11	2,14	15,20
Single	12,23	3,44	2,14	17,80

Tabla 5.8. Resultados de la combinación de algoritmos que consumen menos tiempo para un procesamiento 1 canal con enteros de 16 bits y Single.

En la Tabla 5.8, en la columna “Total por un Canal” se puede observar cuánto tiempo lleva la ejecución del procesamiento de una señal de 45 muestras (un canal), ante una determinada combinación de algoritmos para 2 tipos de dato establecidos previamente. Para realizar el procesamiento completo se deben procesar 10 canales, lo cual tomaría 10 veces más del tiempo reportado en la Tabla 5.8: 150 μs para int16_SIMD y 178 μs para datos tipo Single.

10 canales

Tipo de dato	Promediación Coherente 2 Señales [μs]	Extractor de Características 16 Hz [μs]	Clasificador LDA [μs]	Total [μs]
int16_SIMD	96,49	16,48	11,66	124,62
Single	111,05	17,01	12,78	140,84

Tabla 5.9. Resultados de la combinación de algoritmos que consumen menor tiempo para procesamiento 10 canales con enteros de 16 bits y Single.

En la Tabla 5.9, en la columna “Total” se presenta el tiempo de ejecución total para la misma combinación de algoritmos de la Tabla 5.8 pero para un procesamiento de una sola señal de 450 muestras.

Combinación de algoritmos con mayor tiempo de procesamiento

1 canal

Tipo de dato	Promediación Coherente 4 Señales [μs]	Extractor de Características 32 Hz v	Clasificador LDA [μs]	Total por un Canal [μs]
int16_SIMD	21,63	3,40	3,08	28,11
Single	24,71	3,44	2,32	30,47

Tabla 5.10. Resultados de la combinación de algoritmos que consumen mayor tiempo para procesamiento 1 canal con enteros de 16 bits y Single.

De la misma manera que en la combinación de algoritmos con menor tiempo de procesamiento para un canal, para que el procesamiento se realice de manera completa se deben procesar 10 canales. Por lo tanto el tiempo que llevaría para int16_SIMD es 281,1 μs y 304,7 μs para tipo Single.

10 canales

Tipo de dato	Promediación Coherente 4 Señales [μs]	Extractor de Características ERC [μs]	Clasificador LDA [μs]	Total [μs]
int16_SIMD	195,30	31,75	22,22	249,27
Single	228,87	32,30	24,59	285,76

Tabla 5.11. Resultados de la combinación de algoritmos que consumen mayor tiempo para procesamiento 10 canales con enteros de 16 bits y Single.

Los valores de tiempo de ejecución de la etapa del clasificador fueron calculados ya que las señales a la salida de los extractores de características tienen un tamaño que no fue mensurado en este trabajo.

Con estos resultados se puede concluir que en términos temporales, es más conveniente emplear el procesamiento de una señal de 450 muestras en lugar del procesamiento por cada canal.

5.2.2. Memoria ROM

La medición de memoria ROM se realiza utilizando la información del espacio de memoria utilizado por la función evaluada, más el espacio utilizado por las funciones llamadas dentro de la función implementada y considerando también la memoria que ocupan eventuales índices o coeficientes que residen en memoria no volátil. La memoria que ocupan de todas las funciones está medidas en bytes.

El consumo de memoria ROM y memoria RAM **de las funciones implementadas en este trabajo** no depende de la longitud de la señal, por el tipo de procesamiento que estas realizan y por la manera en el cual están programadas. Es por ello que solo se compara el consumo de memoria ROM en función del tipo de dato utilizado.

5.2.2.1. Promediación Coherente

La memoria ROM utilizada por este algoritmo se determina sumando la memoria consumida por la función, más las funciones de suma y división empleadas en ellas.

5.2.2.1.1. Promediación Coherente – 2 Señales

Tipo de Dato	Función propia	Función de suma	Función shift (división)	Total
Punto Flotante	48	244	224	516
Entero 32-bits	48	204	476	728
Entero 16-bits	48	228	532	808
Entero 16-bits (SIMD)	48	228	532	808
q31	48	228	549	825
q15	48	524	524	1096

Tabla 5.12. Consumo de memoria ROM por la promediación coherente de 2 señales según el tipo de dato.

5.2.2.1.2. Promediación Coherente – 4 Señales

Tipo de Dato	Función propia	Función de suma	Función shift (división)	Total
Punto Flotante	72	244	224	540
Entero 32-bits	72	204	476	752
Entero 16-bits	72	228	532	832
Entero 16-bits (SIMD)	72	228	532	832
q31	72	228	549	849
q15	72	524	524	1120

Tabla 5.13. Consumo de memoria ROM por la promediación coherente de 4 señales según el tipo de dato.

5.2.2.2. Extractores de características

En el submuestreo equitemporal (32 y 16 Hz) la memoria ROM que se emplea es únicamente la que de la propia función. Por otro lado el submuestreo ERC emplea memoria, además de la de su propia función, índices que permanecen fijos.

5.2.2.2.1. Submuestreo 32 Hz

Tipo de Dato	Función propia
Punto Flotante	68
Entero 32-bits	68
Entero 16-bits	68
Q15	68
Q31	68

Tabla 5.14. Consumo de memoria ROM por el submuestreo 32Hz según el tipo de dato.

5.2.2.2.2. Submuestreo 16 Hz

Tipo de Dato	Función propia
Punto Flotante	92
Entero 32-bits	92
Entero 16-bits	92
Q15	92
Q31	92

Tabla 5.15. Consumo de memoria ROM por el submuestreo 16Hz según el tipo de dato.

5.2.2.2.3. Submuestreo ERC

Tipo de Dato	Función propia	Índices
Punto Flotante	68	$N*4$
Entero 32-bits	68	$N*4$
Q15	68	$N*2$
Q31	68	$N*4$
Entero 16-bits	76	$N*2$

Tabla 5.16. Consumo de memoria ROM por el submuestreo ERC según el tipo de dato.

Donde N es la longitud de la señal procesada. $N*4$ es una palabra y $N*2$ es media palabra.

Se recuerda al lector, según lo explicado en la sección 2.3.2.2, que este submuestreo emplea índices para poder extraer determinadas características de la señal de entrada. Tales índices en una implementación del sistema deberían ser almacenados en Memoria ROM y el espacio que éstos ocupan depende si el tipo de dato empleado ocupa una palabra o media palabra.

5.2.2.3. Clasificador LDA

En el clasificador LDA, la memoria ROM que se emplea es la que ocupa la propia función, así como también la memoria que requiere el vector de pesos para que pueda realizarse el producto punto.

Tipo de Dato	Función propia	Producto punto	Pesos	Total
Entero 32-bits	64	208	4xN	272 + 4xN
Entero 16-bits	68	248	2xN	316 + 2xN
Punto Flotante	80	260	4xN	340 + 4xN
Entero 16-bits (SIMD)	64	276	2xN	340 + 2xN
q31	76	380	4xN	456 + 4xN
q15	76	524	2xN	600 + 2xN

Tabla 5.17. Consumo de memoria ROM por el clasificador LDA según el tipo de dato.

Donde N es la longitud de la señal procesada, $N*4$ es la cantidad de bytes una palabra y $N*2$ es la cantidad de bytes en media palabra.

Según lo visto en la sección 2.3.3, el clasificador LDA emplea un vector de pesos para realizar el producto punto correspondiente a esta etapa. Este vector de pesos se almacena en memoria ROM y el tamaño de éstos depende del tamaño de la señal y de que si el tipo de dato se almacena en una palabra o en media palabra.

5.2.2.4. Procesamiento total: Condiciones límites

Siguiendo la misma estrategia implementada en la sección “1 canal vs. 10 canales” de la sección 5.2.13, en este apartado se reporta la combinación de algoritmos que presenta el menor y mayor consumo de memoria ROM para los tipos de dato que más y menos memoria ROM emplean. Utilizando algoritmos que pasaron satisfactoriamente la etapa de verificación de procesamiento.

Se recuerda que el objetivo de reportar las condiciones límites es mostrar un panorama de las condiciones de uso mínimo y máximo respecto al uso de memoria ROM del procesamiento completo de la señal.

Caso de menor consumo de memoria ROM

Tipo de dato	Promediación coherente 2 señales	Extractores de Características Submuestreo 32 Hz	Clasificador LDA	Total
Punto Flotante	516	68	340	924
Entero 16-bits (SIMD)	808	68	340	1216

Tabla 5.17. Consumo de memoria ROM por la combinación de menor consumo de memoria para datos de tipo punto flotante y entero de 16 bits utilizando SIMD.

Caso de mayor consumo de memoria ROM

Tipo de dato	Promediación coherente 4 señales	Extractores de Características Submuestreo 16 Hz	Clasificador LDA	Total
Punto Flotante	540	92	340	972
Entero 16-bits (SIMD)	832	92	340	1264

Tabla 5.18. Consumo de memoria ROM por la combinación de mayor consumo de memoria para datos de tipo punto flotante y entero de 16 bits utilizando SIMD.

Se debe aclarar que, existe la posibilidad de que un submuestreo ERC consuma más o incluso menos memoria ROM que los submuestreos equitemporales elegidos en la etapa de extracción de características, pero en este caso la memoria ROM empleada por el submuestreo ERC dependería del tamaño de la función, lo que dejaría a las condiciones límites presentadas en función de este parámetro. Lo cual no es beneficioso al entendimiento de los límites del uso de este recurso.

Por otro lado, el consumo de memoria ROM total reportado tiene en cuenta la memoria que ocupan las instrucciones necesarias para llevar a cabo el procesamiento, no se tiene en cuenta la memoria ROM necesaria en el vector de pesos que utiliza el clasificador, dado que ésta es dependiente de la longitud de la señal procesada y del tipo de dato utilizado (palabra o media palabra).

5.2.3. Memoria RAM

La medición de memoria RAM se realiza utilizando información del máximo espacio del stack utilizada por la función evaluada, más el máximo espacio del stack utilizado por sub-funciones llamadas dentro de ella. Y se calcula la peor condición para la memoria RAM consumida como la suma de la máxima memoria RAM máxima utilizada por la función propia más la máxima RAM consumida de todas sus sub-funciones. La memoria RAM también se mide en bytes.

5.2.3.1. Promediación Coherente

Promediación Coherente de 2 y 4 señales

Tipo de Dato	Función propia	Función de suma	Función de multiplicación	Peor Condición
Entero 16-bits	24	48	48	72
Punto Flotante	24	64	48	88
Entero 32-bits	24	64	64	88
Entero 16-bits (SIMD)	24	80	48	104
q31	24	120	80	144
q15	24	80	128	152

Tabla 5.19. Consumo de memoria RAM por la promediación coherente de 2 y 4 señales según el tipo de dato.

5.2.3.2. Submuestreo

Submuestreo 32Hz

Tipo de Dato	Función propia
Punto Flotante	32
Entero 32-bits	32
Entero 16-bits	32
Q15	32
Q31	32

Tabla 5.20. Consumo de memoria RAM por el submuestreo 32 Hz según el tipo de dato.

Submuestreo 16Hz

Tipo de Dato	Función propia
Punto Flotante	40
Entero 32-bits	40
Entero 16-bits	40
Q15	40
Q31	40

Tabla 5.21. Consumo de memoria RAM por el submuestreo 16 Hz según el tipo de dato.

Submuestreo ERC

Tipo de Dato	Función propia
Punto Flotante	24
Entero 32-bits	24
Entero 16-bits	24
Q15	24
Q31	24

Tabla 5.22. Consumo de memoria RAM por el submuestreo ERC según el tipo de dato.

5.2.3.3. Clasificador LDA

Tipo de Dato	Función propia	Producto punto	Peor condición
Entero 32-bits	24	32	56
Entero 16-bits	24	32	56
Punto Flotante	24	32	56
Entero 16-bits (SIMD)	24	96	120
q31	24	120	144
q15	24	136	136

Tabla 5.23. Consumo de memoria RAM por los clasificadores LDA y ERC según el tipo de dato

5.2.3.4. Procesamiento total: Condiciones límites

A diferencia de las condiciones límites de la memoria ROM, las condiciones límites de memoria RAM simplemente se obtienen analizando el proceso que más cantidad de RAM utiliza para un determinado tipo de dato debido ya que, respecto a los algoritmos implicados en el procesamiento, la utilización de este recurso se realiza de manera no simultánea.

Capítulo 6: Análisis económico

6.1. Introducción

En este capítulo se desarrollan aspectos económicos de una etapa de Investigación y Desarrollo de un proyecto que parte de los resultados obtenidos en este trabajo final. Donde el objetivo final sea obtener una Interfaz Cerebro-Computadora basado en la detección de potencial P300 por estimulación visual en dispositivos dedicados como producto comercial.

Para la obtención del producto comercial mencionado, se deben llevar a cabo diferentes etapas: Investigación, diseño, implementación, reglamentación, etc. En este capítulo se sólo se aborda la etapa de implementación de este tipo de sistemas. La cual es la etapa inmediata siguiente a la alcanzada en este trabajo, por lo que la estimación de los costos de la misma es viable. No se realizará un análisis de los costos de las demás etapas dado que éstos dependen de múltiples factores de diversa complejidad y variables que se desconocen en esta instancia.

6.2. Proyecto propuesto

6.2.1. Objetivo

Implementación y optimización de un deletreador de Donchin en un dispositivo dedicado.

6.2.2. Resumen

Los bloques fundamentales de una ICC son: Adquisición de Señales, Procesamiento, Aplicación y Realimentación.

La etapa de adquisición consta del registro de EEGs mediante diferentes canales y la digitalización y almacenamiento de los mismos. El procesamiento en un dispositivo dedicado requiere, valga la redundancia, de un procesador que ejecute los algoritmos que presenta esta etapa. Por último, por tratarse de un Deletreador de Donchin la finalidad de este sistema, la etapa de aplicación y realimentación constan del mismo recurso: Un display. Éste muestra caracteres como letras, números y otros caracteres, actuando como fuente del estímulo visual. Esta herramienta también muestra los resultados parciales del deletreo, oficiando como realimentación al usuario, cerrando el lazo del sistema.

Para tener este sistema en marcha se necesita tener los bloques mencionados anteriormente funcionales e integrados.

Con el objetivo de la optimización el funcionamiento del sistema, una vez integrados los bloques del mismo, se debe realizar un análisis del rendimiento del sistema general para luego trabajar sobre ello. En esa instancia se debe tener en cuenta el tiempo de respuesta del sistema; análisis de la cantidad de canales empleados en la adquisición y procesamiento; tasa de efectividad, sensibilidad y especificidad del procesamiento empleado; rendimiento del microcontrolador; El procesamiento se debe ajustar al tipo de dato que el procesador mejor responda; configuración del tiempo de estimulación visual y del tiempo inter-estímulo; entre otros.

6.2.3. Alcances y Limitaciones

Este proyecto no pretende centrarse en la construcción física de ninguno de los bloques necesarios para la implementación de este sistema. Por lo que se propone adquirir bloques idóneos para esta aplicación disponibles comercialmente y centrarse en la integración y optimización de la aplicación planteada.

Por otro lado, el objetivo del proyecto es implementar y optimizar sistemas de ICC en dispositivos dedicados desde un punto de vista académico. Es por esto que esta etapa no contempla regulaciones que aplican para dispositivos médicos comerciales, tales como las de funcionamiento esencial y seguridad eléctrica o las de software de producto médico.

6.2.4. Duración

El proyecto se plantea para 12 meses. Se cree que este plazo es coherente dado que el proyecto tiene como punto de partida los resultados y conclusiones de este trabajo. Así como también se establecen alcances y limitaciones para la realización del mismo.

6.2.6. Equipo de trabajo

El recurso humano involucrado en este proyecto se encuentra conformado por:

Carácter	Posición	Dedicación Horaria Semanal
Director	Bioingeniero Senior	20
Co-Director	Investigador en el campo de ICC	10
Integrantes	Bioingeniero Junior	40
Becario PID	Estudiante de grado de Bioingeniería	10

Tabla 6.1. Equipo de trabajo involucrado en el proyecto y su dedicación semanal al mismo.

6.2.7. Infraestructura

Para la realización de este proyecto se cuenta con las instalaciones de la Facultad de Ingeniería de la UNER, el cual cuenta con el lugar físico así como también con el soporte técnico necesario para llevar adelante este proyecto. Por ello no se considerarán costos de servicios como el del suministro eléctrico o servicio de internet, así como tampoco se considerarán costos de equipamiento electrónico (osciloscopios, generadores de señales, fuentes, etc.) disponibles en estas instalaciones.

6.2.8. Impacto

Al llevar a cabo el objetivo planteado, se tienen consecuencias en distintos campos.

Investigación: Ofrece una realimentación del desempeño del rendimiento del sistema a los investigadores de procesamiento de señales en ICC en vistas de mejorar el desempeño de la totalidad del sistema. A su vez brinda la potencialidad de obtener bases de datos propias mediante un deletreador de Donchin.

Social: Sería la primer ICC basada en un dispositivo dedicado a nivel nacional de relativo bajo costo, al no depender de una computadora para su funcionamiento, para la comunicación asistida de personas con patologías neuromotoras severas.

Económico: Por el motivo mencionado anteriormente, es posible captar inversores para el desarrollo de un prototipo comercial para este prototipo, tanto para el área biomédica, como en otras áreas como entretenimiento, aplicaciones de uso general, etc.

6.3. Financiamiento

Se evaluaron diferentes fuentes de financiamientos estatales para la realización del proyecto presentado anteriormente y dada las características de éste se optó por un la línea para Programa de Investigación y Desarrollo (PID) del a Universidad Nacional De Entre Ríos (UNER). Este programa mantiene una ventanilla abierta de manera constante para la presentación de proyectos.

Los proyectos de Investigación Científica, Desarrollo e Innovación Tecnológica son acreditados por la UNER a través de la aprobación de su consejo superior según la normativa vigente, Ordenanza 403 "CS" y Anexo único, que se establece el régimen de presentación, aprobación y seguimiento de los mismos.

6.3.1. Presupuesto

En la Tabla 6.1 se observa el presupuesto que requiere el proyecto propuesto, éste presupuesto contiene con los ítems clasificados como lo requieren los formularios de la PID UNER. Los montos se encuentran en Pesos Argentinos (AR\$).

Clasificación	Concepto	Monto
Bienes de consumo	Componentes electrónicos varios (cables, conectores, etc.)	\$3.000,00
	Artículos de librería (hojas A4, tinta de impresora, etc.)	\$1.000,00
Compuestos químicos	-	-
Servicios no personales	Costos de envío	\$4.500,00
Bienes de uso - Equipamiento	Módulo de adquisición de EEG (g.MOBilab+ 8 channel EEG version)	\$ 71.460,40
	Placa de desarrollo EDU-CIAA	\$600,00
	Matriz de LEDs RGB de 32 x 32	\$669,8,00
	Controlador del arreglo de LEDs	\$1.000,00
	Fuente para arreglo de LEDs (5V, 4A)	\$222,80
Bienes de uso - Bibliografía	-	-
Total		\$82.453,00

Tabla 6.1. Presupuesto de los recursos necesarios para llevar a cabo un Deletreador de Donchin basado en un sistema dedicado.

En el Anexo N°2 "Presupuestos y precios de lista para el análisis económico" se brinda información que soporta algunos montos del presupuesto presentado en la Tabla 6.1.

Respecto a cuestiones impositivas, dado que este se enmarca en una compra por parte de un grupo de investigación de la Universidad, las eventuales importaciones efectuadas no

se ven afectadas por impuestos según el Régimen de importaciones para insumos destinados a investigaciones científico-tecnológicas (ROECyT¹¹).

Por otro lado, se aclara que los precios de los productos en dólares se cotizaron al precio del Peso Argentino según el valor de cambio del Banco de la Nación Argentina (BNA) el día 18 de Marzo del 2016 (1 dólar = 14,9 AR\$).

6.4. Análisis de costos

La línea de financiamiento elegida exige presentar el equipo de trabajo el cual va a llevar adelante el proyecto para poder evaluar la viabilidad de la misma. Sin embargo, ésta no considera estipendios para recurso humano dedicado al proyecto.

No obstante, en esta sección se estiman los costos implicados en el recurso humano para luego poder comparar los costos de recursos humanos con los recursos materiales y poder sacar conclusiones al respecto.

Carácter	Posición	Estipendio/Beca
Director	Bioingeniero Senior	\$172.800,00
Co-Director	Investigador en el campo de ICC	\$132.397,92
Integrante	Bioingeniero Junior	\$153.600,00
Becario PID	Estudiante de grado de Bioingeniería	\$16.800,00
Total		\$475.597,92

Tabla 6.2. Presupuesto de los recursos necesarios para llevar a cabo un Deletreador de Donchin basado en un sistema dedicado.

Los montos especificados en la Tabla 6.2 están calculados para un año, que es el plazo de duración del proyecto.

Para el cálculo del estipendio al Bioingeniero Senior y Junior se tomó como referencia los aranceles mínimos dictaminados por el Colegio de Ingenieros Especialistas de Entre Ríos (CIEER) según la resolución de directorio N° 949/2012 y la última actualización del monto de la hora ingenio que esta utiliza en la resolución N° 1223/2015.

Para el monto destinado al investigador se basó en la beca del CONICET que esta destina a doctorandos para el corriente año (véase Anexo N°3: "Estipendio de Becas doctorales del CONICET según zona de trabajo"). El monto para el Becario PID actualmente se encuentra en \$1.400 mensual y esta se puede solicitar en la instancia de la presentación de financiamiento del proyecto a la UNER.

En la Figura 6.1 se presenta un gráfico el cual representa la distribución de los eventuales costos de recurso humano, materiales y servicios. En el mismo queda evidenciado que los

¹¹ Para mayor información visitar: www.roecyt.mincyt.gob.ar

costos de RRHH representa la mayor inversión dado el valor que éstos aportan en este tipo de proyectos.

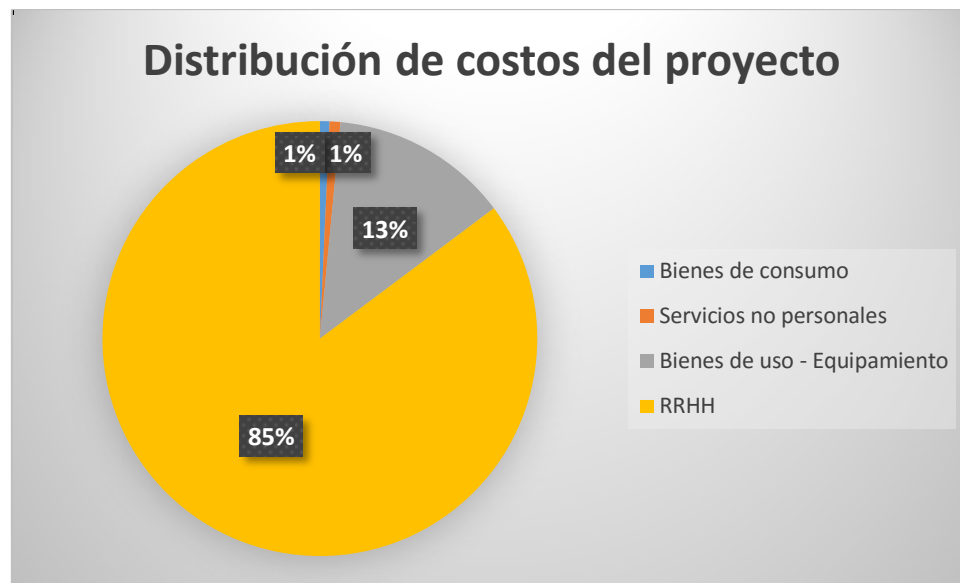


Figura 6.1. Distribución de los costos en recurso humano, materiales y servicios

F. Conclusiones

En líneas generales el rendimiento del procesador elegido para esta aplicación ha dado muy buenos resultados.

La memoria ROM utilizada en la peor condición de combinación de algoritmos para este parámetro es mínima respecto a la ofrecida por el microcontrolador elegido (LPC4337 de NXP). Cabe aclarar la memoria ROM, así como la memoria RAM, disponible para la aplicación lo determina el chip, éstas son características ajenas a las propiedades del procesador.

En cuanto a la memoria RAM, el consumo de esta también es bajo. Esto se debe en gran parte a que los algoritmos no utilizan simultáneamente este recurso, así como también se debe al uso eficiente de este recurso por parte de las librerías provistas por CMSIS-DSP. Sin embargo, éste es un parámetro muy importante a tener en cuenta cuando se implemente el sistema ICC completo. Dado que según lo investigado, el desbordamiento de memoria RAM produce efectos devastadores en las tareas llevadas a cabo por el procesador en tiempo de ejecución. Hay que tener en cuenta también que es un recurso que se vuelve crítico en función del aumento de la cantidad de muestras contenidas en las señales procesadas. Este trabajo presenta herramientas para poder monitorear el uso de este recurso en la etapa de implementación del sistema completo.

Con respecto al tiempo de ejecución de un procesamiento completo, los resultados muestran que el tiempo de procesamiento es 3 órdenes de magnitud menores que los tiempos involucrados en la estimulación visual y la respuesta fisiológica necesaria (potencial P300) para establecer la comunicación, lo cual indica que la etapa de procesamiento digital de la señal no es limitante en el desarrollo de los sistemas ICC sobre sistemas dedicados basados en procesadores Cortex-M4F.

Queda demostrado que se puede estimar con una precisión aceptable el tiempo de ejecución de los algoritmos en función de la longitud de la señal para todos los tipos de dato. Esta estimación serviría para tener noción de la utilización de estos recursos para aplicaciones donde el temporizado no es crítico, no así para los sistemas críticos en tiempo real.

La modificación de los tipos de datos al ejecutar los procesamientos muestra diferencias significativas en el consumo de recursos evaluados. Lo que demuestra que este parámetro es un factor importante a tener en cuenta en la implementación u optimización de estos sistemas. Ya que en estas instancias se debe encontrar un equilibrio de costo/beneficio entre el desempeño del procesador al ejecutar algoritmos con un determinado tipo de dato y las características que éstos presentan (rango, precisión, etc.).

De todos los tipos de dato evaluados, los de punto fijo (q15 y q31) son los que más recursos consumen, esto se debe a que el procesador debe emplear más instrucciones, que a su vez, implican más tiempo de ejecución, para realizar el mismo procesamiento que con otros tipo de datos. Los otros tipos de dato (enteros y punto flotante) consumen menor cantidad de recursos dado que, por un lado, los enteros son el tipo de dato de punto fijo más simple, por lo que emplearía menor cantidad de instrucciones. Y por otro lado, el procesador cuenta con un área especializada en DSP de tipos de dato punto flotante, lo cual simplificaría las instrucciones empleadas en procesamiento de este tipo de datos también.

Respecto al submuestreo equitemporal, la diferencia de la utilización de recursos es prácticamente nula respecto al tipo de dato. Dado que estos emplean instrucciones que operan moviendo y copiando información de registros (palabras o media palabra), no realizando procesamiento matemático sobre ellos. Tampoco hay diferencia significativa respecto al tiempo de ejecución entre los submuestreos equitemporales con los ERC, siempre y cuando el tamaño de la señal de salida de estos algoritmos sea el mismo.

Por último, en cuanto a la conveniencia del procesamiento por canal o de a 10 canales, queda evidenciado que en cuanto al rendimiento del procesador, el que consume menos recursos en relación tiempo-memoria es el de 10 canales. Esto se debe a que la cantidad de información procesada es la misma, pero en esta se ejecutan de manera más eficiente ya que, ejecuta menor cantidad de cálculos que la de 1 canal. Aunque puede haberse arribado a esta conclusión al analizar la situación teóricamente, el resultado de este trabajo cuantifica estas diferencias.

G. Bibliografía

- [1] Brunner, C., Birbaumer, N., Blankertz, B., Guger, C., Kübler, A., Mattia, D., Müller-Putz, G. R. (2015). BNCI horizon 2020: Towards a roadmap for the BCI community. *Brain-Computer Interfaces*, 2(1), 1–10. doi:10.1080/2326263x.2015.1008956
- [2] ARASAAC. (1982). *ARASAAC: ¿Qué son los Sistemas Aumentativos y Alternativos de Comunicación (SAAC)?*. Última visita 7 de Marzo, 2016, del Portal Aragonés de la Comunicación Aumentativa y Alternativa, <http://arasaac.org/aac.php>
- [3] Wolpaw, J. R., Birbaumer, N., McFarland, D. J., Pfurtscheller, G., & Vaughan, T. M. (2002). Brain-computer interfaces for communication and control. *Clinical Neurophysiology*, 113(6), 767–791. doi:10.1016/s1388-2457(02)00057-3
- [4] Wolpaw, J. R., Birbaumer, N., Heetderks, W. J., McFarland, D. J., Peckham, P. H., Schalk, G., ... Vaughan, T. M. (2000). Brain-computer interface technology: A review of the first international meeting. *IEEE Transactions on Rehabilitation Engineering*, 8(2), 164–173. doi:10.1109/tre.2000.847807
- [5] Montalvo, M. Estado del Arte: Interfaces Cerebro Computadora.
- [6] Farwell, L. A., & Donchin, E. (1988). Talking off the top of your head: Toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and Clinical Neurophysiology*, 70(6), 510–523. doi:10.1016/0013-4694(88)90149-6
- [7] Polich, J., & Criado, J. R. (2006). Neuropsychology and neuropharmacology of P3a and P3b. *International Journal of Psychophysiology*, 60(2), 172–185. doi:10.1016/j.ijpsycho.2005.12.012
- [8] Wikipedia Foundation. *Interfaz cerebro-computadora*. Última visita 7 de Marzo, 2016, de Wikipedia, http://es.wikipedia.org/wiki/Interfaz_Cerebro_Computadora
- [9] Atum, Y. (2013). Técnicas de inteligencia computacional aplicadas a la detección de potenciales relacionados a eventos en interfaces cerebro computadoras. Tesis de Maestría. Laboratorio de Ingeniería en Rehabilitación e Investigaciones Neuromusculares y Sensoriales (LIRINS). Universidad Nacional de Entre Ríos (UNER).
- [10] Spinelli, M. E. Amplificadores de Instrumentación en Aplicaciones Biomédica. Tesis de Doctorado. Departamento de Electrotecnia. Universidad Nacional de la Plata (UNLP).
- [11] Filomena, E., Aldonate, J., Acevedo, R., & Spinelli, E. Revisión sobre 11. In *XVII Congreso Argentino de Bioingeniería-VI Jornadas de Ingeniería Clínica*.
- [12] NXP Semiconductors. “LPC43xx ARM Cortex-M4M0 multi-core microcontroller”. Hoja de datos de LPC43xx, 18 de Febrero, 2015. Rev. 1.9.
- [13] Caprile, S. R. (2013). *Desarrollo con microcontroladores ARM cortex-M3* (1ra ed.). Buenos Aires: Sergio R. Caprile.
- [14] ARM Ltd. (2014, September 29). *DSP & SIMD*. Última visita 7 de Marzo, 2016, de ARM The Architecture for the Digital World, <https://www.arm.com/products/processors/technologies/dsp-simd.php>
- [15] ARM. “Cortex™ -M4 Devices”. Guía genérica de usuario del procesador Cortex-M4, (2010).

-
- [16] Lu, J., Speier, W., Hu, X., & Pouratian, N. (2013). The effects of stimulus timing features on P300 speller performance. *Clinical Neurophysiology*, 124(2), 306–314. doi:10.1016/j.clinph.2012.08.002
- [17] Advameg. (2016). *Comp.realtime: Frequently asked questions (FAQs) (version 3.5)*. Última visita 7 de Marzo, 2016, de <http://www.faqs.org/>, <http://www.faqs.org/faqs/realtime-computing/faq/>
- [18] Electrical Engineering Stack Exchange. (2016). *Why do microcontrollers have so little RAM?*. Última Visita 7 de Marzo, 2016, from <http://electronics.stackexchange.com/>, <http://electronics.stackexchange.com/questions/134496/why-do-microcontrollers-have-so-little-ram>
- [19] Botcazou, E., Comar, C., & Hainque, O. (2005, June). Compile-time stack requirements analysis with GCC. In *GCC Developers' Summit* (p. 93).
- [20] Goovaerts, H. G., & Rompelman, O. (1991). Coherent average technique — a tutorial review. *Journal of Biomedical Engineering*, 13(4), 275–280. doi:10.1016/0141-5425(91)90108
- [21] Rompelman, O., & Ros, H. H. (1986). Coherent averaging technique: A tutorial review part 1: Noise reduction and the equivalent filter. *Journal of Biomedical Engineering*, 8(1), 24–29. doi:10.1016/0141-5425(86)90026-9
- [22] Lyons, R. G. (2004). *Understanding digital signal processing* (2nd ed.). United States: Prentice Hall PTR.
- [23] Polich, J. (2007). Updating P300: An integrative theory of P3a and P3b. *Clinical Neurophysiology*, 118(10), 2128–2148. doi:10.1016/j.clinph.2007.04.019
- [24] MathWorks. (1994). *Fixed-point data types*. Última visita 7 de Marzo, 2016, extraído de <http://www.mathworks.com/>, <http://www.mathworks.com/help/fixedpoint/ug/fixed-point-data-types.html>
- [25] Silicon Labs. “Digital Signal Processing with the EFM32”. Nota de aplicación - AN0051, (2013). Rev.1.03.
- [26] Smith, S. W. (1997). The scientist and engineer's guide to digital signal processing.
- [27] Free Software Foundation, Inc. *Static stack usage analysis - GNAT user's guide for native platforms*. Última visita March 7, 2016, extraída de GCC, the GNU Compiler Collection, https://gcc.gnu.org/onlinedocs/gnat_ugn/Static-Stack-Usage-Analysis.html#Static-Stack-Usage-Analysis
- [28] Yiu, J. (2010). *The definitive guide to the ARM cortex-m3 TI*. United Kingdom: Newnes (an imprint of Butterworth-Heinemann Ltd).

H. ANEXOS

Anexo N°1: “Tiempos de ejecución de algoritmos en función de la longitud de señal y el tipo de dato”.

Algoritmo	Tipo	Tipo de dato	Muestras en la señal	Mínimo	Ticks Máximos	Tiempo [μs]
Promediación Coherente	2 Señales	Single	45	2487	2494	12,23
			50	2748	2755	13,50
			100	5209	5216	25,57
			200	10179	10191	49,96
			300	15155	15167	74,35
			400	20130	20142	98,74
			450	22644	22654	111,05
		int32	45	2370	2378	11,66
			50	2630	2634	12,91
			100	4912	4917	24,10
			200	9587	9591	47,01
			300	14266	14270	69,95
			400	18941	18945	92,87
			450	21334	21338	104,60
		int16	45	2450	2461	12,06
			50	2715	2726	13,36
			100	5076	5087	24,94
			200	9899	9914	48,60
			300	14729	14737	72,24
			400	19553	19560	95,88
			450	22012	22023	107,96
		int16 SIMD	45	2222	2233	10,95
			50	2475	2486	12,19
			100	4545	4556	22,33
			200	8843	8858	43,42
			300	13148	13156	64,49
			400	17447	17454	85,56
			450	19672	19683	96,49
		Q15	45	2412	2412	11,82
			50	2677	2677	13,12
			100	4993	4993	24,48
			200	9742	9742	47,75
			300	14492	14492	71,04
			400	19242	19242	94,32
			450	21677	21677	106,26

		Q31	45	2901	2901	14,22
			50	3208	3208	15,73
			100	6082	6082	29,81
			200	11907	11907	58,37
			300	17727	17731	86,92
			400	23555	23555	115,47
			450	26503	26507	129,94
	4 Señales	Single	45	5033	5041	24,71
			50	5574	5578	27,34
			100	10650	10658	52,25
			200	20930	20934	102,62
			300	31200	31208	152,98
			400	41475	41483	203,35
			450	46682	46690	228,87
		int32	45	4813	4821	23,63
			50	5346	5351	26,23
			100	10148	10152	49,76
			200	19918	19922	97,66
			300	29692	29696	145,57
			400	39467	39471	193,49
			450	44449	44453	217,91
		int16	45	5063	5067	24,84
			50	5618	5625	27,57
			100	10684	10688	52,39
			200	20955	20967	102,78
			300	31234	31238	153,13
			400	41509	41513	203,50
			450	46734	46738	229,11
		int16 SIMD	45	4408	4412	21,63
			50	4924	4937	24,20
			100	9135	9139	44,80
			200	17856	17868	87,59
			300	26585	26589	130,34
			400	35310	35314	173,11
			450	39837	39841	195,30
		Q15	45	4607	4607	22,58
			50	5141	5141	25,20
			100	9587	9587	47,00
			200	18787	18787	92,09
			300	27988	27988	137,20
			400	37188	37188	182,29
			450	41942	41942	205,60

		Q31	45	6454	6454	31,64
			50	7143	7146	35,03
			100	13735	13738	67,34
			200	27060	27063	132,66
			300	40386	40389	197,99
			400	53711	53711	263,29
			450	60444	60447	296,31
Submuestreo	32 Hz	Single	45	702	702	3,44
			50	790	790	3,87
			100	1514	1514	7,42
			200	2962	2962	14,52
			300	4414	4414	21,64
			400	5866	5866	28,75
			450	6590	6590	32,30
		int32	45	702	702	3,44
			50	790	790	3,87
			100	1514	1514	7,42
			200	2962	2962	14,52
			300	4414	4414	21,64
			400	5866	5866	28,75
			450	6590	6590	32,30
		int16	45	694	694	3,40
			50	778	778	3,81
			100	1490	1490	7,30
			200	2914	2914	14,28
			300	4338	4338	21,26
			400	5766	5766	28,26
			450	6478	6478	31,75
		q15	45	679	679	3,33
			50	763	763	3,74
			100	1463	1463	7,17
			200	2863	2863	14,03
			300	4263	4263	20,90
			400	5663	5663	27,76
			450	6363	6363	31,19
		q31	45	679	680	3,33
			50	763	764	3,75
			100	1463	1464	7,18
			200	2863	2864	14,04
			300	4263	4264	20,90
			400	5663	5664	27,76
			450	6367	6367	31,21


	16 Hz	Single	45	445	445	2,18
			50	473	473	2,32
			100	830	830	4,07
			200	1579	1579	7,74
			300	2327	2327	11,41
			400	3075	3075	15,07
			450	3470	3470	17,01
		int32	45	438	439	2,15
			50	470	471	2,31
			100	830	830	4,07
			200	1582	1582	7,75
			300	2327	2327	11,41
			400	3075	3075	15,07
			450	3470	3470	17,01
		int16	45	430	431	2,11
			50	458	459	2,25
			100	803	804	3,94
			200	1534	1534	7,52
			300	2258	2258	11,07
			400	2982	2982	14,62
			450	3361	3361	16,48
		q15	45	415	415	2,03
			50	443	443	2,17
			100	776	776	3,80
			200	1476	1476	7,24
			300	2176	2178	10,68
			400	2876	2878	14,11
			450	3243	3245	15,91
		q31	45	415	415	2,03
			50	443	443	2,17
			100	776	776	3,80
			200	1476	1476	7,24
			300	2176	2178	10,68
			400	2876	2878	14,11
			450	3243	3245	15,91
	ERC	Single	20	662	662	3,25
			115	3589	3589	17,59
		int32	29	939	939	4,60
			49	1607	1607	7,88
		in16	29	1041	1041	5,10
			49	1754	1754	8,60
		q15	20	655	655	3,21

			115	3458	3458	16,95
		q31	20	655	655	3,21
			115	3458	3458	16,95
Clasificadores	LDA	Single	45	1145	1145	5,61
			50	1264	1264	6,20
			100	2321	2321	11,38
			200	4476	4476	21,94
			300	6620	6620	32,45
			400	8775	8775	43,01
			450	9863	9863	48,35
		int32	45	1085	1085	5,32
			50	1199	1199	5,88
			100	2206	2206	10,81
			200	4256	4256	20,86
			300	6304	6304	30,90
			400	8358	8358	40,97
			450	9399	9399	46,07
		int16	45	1294	1294	6,34
			50	1425	1425	6,99
			100	2655	2655	13,01
			200	5153	5153	25,26
			300	7659	7659	37,54
			400	10157	10157	49,79
			450	11425	11425	56,00
		int16 SIMD	45	1081	1081	5,30
			50	1229	1229	6,02
			100	2093	2093	10,26
			200	4016	4016	19,69
			300	5939	5939	29,11
			400	7870	7870	38,58
			450	8929	8929	43,77
		q15	45	1605	1605	7,87
			50	1798	1798	8,81
			100	3255	3255	15,96
			200	6333	6333	31,04
			300	9407	9407	46,11
			400	12481	12481	61,18
			450	14098	14098	69,11
		q31	45	2584	2585	12,67
			50	2842	2843	13,94
			100	5565	5566	27,28
			200	10988	10989	53,87

			300	16415	16416	80,47
			400	21838	21839	107,05
			450	24542	24542	120,30
	ERC	Single	20	606	606	2,97
			115	2670	2670	13,09
		int32	29	757	757	3,71
			49	1171	1171	5,74
		in16	29	894	894	4,38
			49	1392	1392	6,82
		int16 SIMD	29	773	773	3,79
			49	1160	1160	5,69
		q15	20	1596	1596	7,82
			115	3456	3456	16,94
		q31	20	2458	2458	12,05
			115	5598	5598	27,44

Anexo N°2: "Presupuestos y precios de lista para el análisis económico"

Presupuesto del bloque de adquisición de EEG.



g.tec medical engineering GmbH
 Siemingsstrasse 14
 AT-4521 Schiedberg
 Tel: +43-7251/22240
 Fax: +43-7251/22240-39

FIUNER

**3100 Oro Verde, Entre Ríos
ARGENTINIA**

OF-2016-02-93
Date: 08.02.2016
Customer no.: 10.583
Responsible P.: Lechner
Page: 1 / 2

Terms of payment: payable within 30 days, strictly net

Offer: G.MOBILAB+

Pos	Article no.	Description	Quantity QU	Price/pc.	Total price
Your inquiry of 08-02-2016 according to e-mail					
01	5603	g.MOBilab+ 8 channel EEG version 8 unipolar EEG channels; 4 x digital I/O + 4 x digital IN; battery supplied with 4 AA type batteries; wireless signal transmission; streaming onto removable storage card; removable storage card (3015A); electronic handbook; Bluetooth dongle (3016)	1 pcs	4.796,00	4.796,00
	001.	3015 mini SD-card, 2 GB		1 pcs	
	002.	3016 Bluetooth dongle for g.MOBilab+ or e-puck		1 pcs	
	003.	5603DEV g.MOBilab+ 8 channel EEG version device		1 pcs	
	004.	80002 AA battery pack		1 pcs	
02	1114	g.GAMMAbundle for g.MOBilab+ 8 channel EEG version consisting of g.GAMMAcap3SET; 9x g.LADYbird; 2x g.LADYbirdGND; 2x g.GAMMAearclip Ag/AgCl; 5x g.GAMMAgel; 1x g.GAMMAbox for 16 channels; 1x Active Electrode Driver Box Connector for g.MOBilab+ EEG; 3x g.GAMMAAsyringe (1027_3+9x1033+2x1034+2x1039+5x1021+1016a+1019b+3x1067)	1 pcs	2.871,00	2.871,00
	001.	1016A g.GAMMAbox for 16 channels, DC		1 pcs	
	001.01.	1016ADEV g.GAMMAbox for 16 channels, DC device		1 pcs	
	001.02.	80001 9-Volt battery block		1 pcs	
	001.03.	1016IFU g.GAMMAAsys Instructions for use		1 pcs	
	002.	1019B Active Electrode Driver Box Connector for g.MOBilab+ EEG		1 pcs	
	003.	1021 g.GAMMAgel		5 pcs	
	004.	1027_3 g.GAMMAcap3SET, 5mm		1 pcs	
	004.01.	1023L_3 g.GAMMAcap3, size L, 5mm		1 pcs	
	004.02.	1023M_3 g.GAMMAcap3, size M, 5mm		1 pcs	
	004.03.	1023S_3 g.GAMMAcap3, size S, 5mm		1 pcs	
	004.04.	1028 g.GAMMAcapBELT		1 pcs	
	005.	1033 g.LADYbird		9 pcs	
	006.	1034 g.LADYbirdGND		2 pcs	
	007.	1039 g.GAMMAearclip Ag/AgCl		2 pcs	
	008.	1067 g.GAMMAAsyringe		3 pcs	

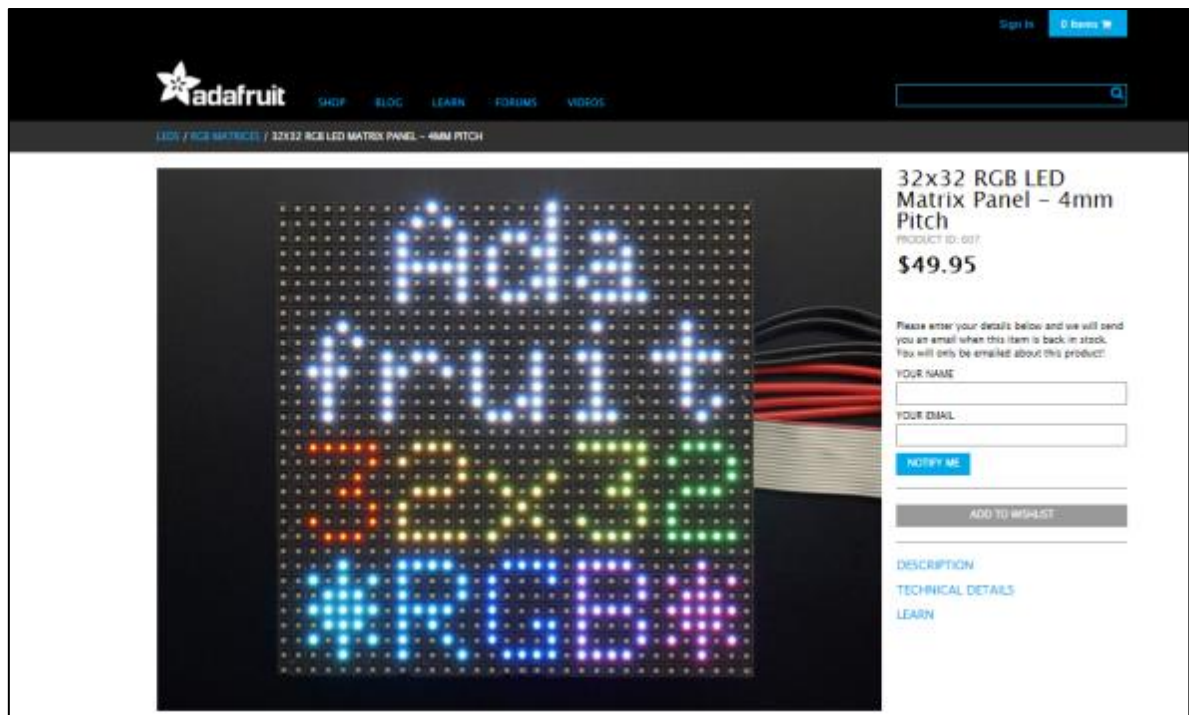
Shipping costs are not included

delivery of goods is normally processed via UPS if no special arrangements have been requested by the customer; shipping costs are added in conclusion to the invoice; approximate costs are available on request

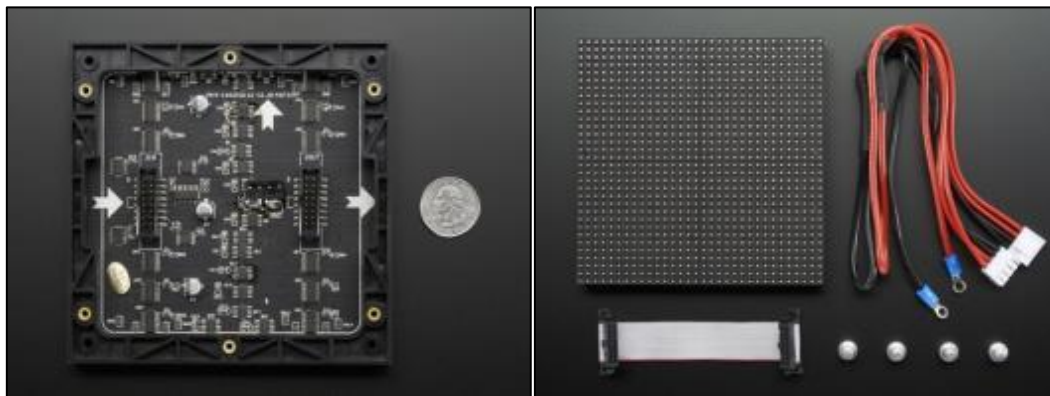
g.tec medical engineering GmbH, Siemingsstrasse 14, 4521 Schiedberg, Austria, Europe, VAT No.: ATU58083139
 Tel.: +43 7251 22240, Fax: +43 7251 22240-39, office@gtec.at, www.gtec.at, Commercial register no.: 249844v, Place of venue: Steyr
 Bank details: SPARKASSE NEUHOFEN BANK A-4501 Neuhofen, Bank no.: 20326, Account no.: 0000-021196
 BIC/SWIFT: SPNKAT21, IBAN: AT492032600000021196

Pos	Article no.	Description	Quantity QU	Price/pc.	Total price
Delivery Terms <i>The goods will be delivered as soon as possible normally within 8 to 14 weeks. Delivery time for customized items can last up to 18 weeks.</i>					
Subtotal:				EUR	7.667,00
Total amount:				EUR	7.667,00
<p>Shipment with UPS Special Warranty: 60 months (= 5 years) for g.USBamp, g.BSamp, g.MOBilab+ and g.HIamp amplifiers! All other items except consumables have a 1 year warranty.</p> <p>Deposit of 30% necessary at order Delivery period: 8-14 weeks after deposit if not otherwise stated.</p> <p>In case of order send the offer back with your company stamp, signature, and your VAT No. for tax free European Union transfers.</p> <p>Prices are excl. tax/duties. The offer is valid for 4 weeks. This price is only valid in your territory.</p> <p>When placing an order, the customer assigns g.tec to organize transport and insurance of the ordered goods on behalf of the customer. Arising costs of transport and insurance are charged to the customer separately. Please indicate your BIC/SWIFT and IBAN for bank transfers.</p> <p>Binding prices are based on costs in Euro obtaining at the time of this quotation. In case the costs or exchange rates have increased by the time of delivery, g.tec shall have the right to adjust prices accordingly.</p> <p>Please note that the Instructions for Use of our products are only available in English language. With your order you confirm that you accept this circumstance.</p> <p>All deliveries effected by g.tec shall be subject to retention of title. Title to the delivered goods shall remain with g.tec until the purchase price - including any and all additional fees - has been fully paid. When placing an order, you agree to our general terms and conditions which may be viewed at http://www.gtec.at/content/download/2180/13147/file/AGB_gtec.pdf</p>					
<p>g.tec medical engineering GmbH, Siemingsstrasse 14, 4521 Schiedlberg, Austria, Europe, VAT No.: ATU58083139 Tel.: +43 7251 22240, Fax: +43 7251 22240-39, office@gtec.at, www.gtec.at, Commercial register no.: 249844v, Place of venue: Steyr Bank details: SPARKASSE NEUHOFEN BANK A-4501 Neuhofer, Bank no.: 20326, Account no.: 0000-021196 BIC/SWIFT: SPNKAT21, IBAN: AT492032600000021196</p>					

Precio de lista de la Matriz de 32x32 LEDs RGB



Vista Posterior y componentes kit de la matriz de leds



Precio de Lista de la fuente de alimentación para la matriz de LEDs

adafruit

SHOPBLOGLEARNFORUMSVIDEOS

POWER / WALL SUPPLIES / 5V / 5V 4A (4000MA) SWITCHING POWER SUPPLY - UL LISTED

5V 4A (4000mA) switching power supply - UL Listed

PRODUCT ID: 1456

\$14.95

IN STOCK

1

ADD TO CART

QTYDISCOUNT

1-9\$14.95

10-99\$13.48


100+\$11.95

ADD TO WISHLIST

DESCRIPTION

TECHNICAL DETAILS

LEARN



94

Anexo N°3: "Estipendio de Becas doctorales del CONICET según zona de trabajo"

PROVINCIA		LOCALIDAD	DOCTORALES AUMENTO JUNIO	POSTDOCTORALES AUMENTO JUNIO	DOCTORALES AUMENTO AGOSTO	POSTDOCTORALES AUMENTO AGOSTO	DOCTORALES AUMENTO DICIEMBRE	POSTDOCTORALES AUMENTO DICIEMBRE
Ciudad Autónoma de Buenos Aires		Bahía Blanca	\$ 10.596,95	\$ 11.063,39	\$ 11.499,59	\$ 11.972,94	\$ 12.074,57	\$ 14.883,03
		Chascomús	\$ 11.087,48	\$ 14.513,07	\$ 12.682,09	\$ 15.750,25	\$ 13.257,69	\$ 16.658,26
		La Plata	\$ 10.905,58	\$ 14.804,29	\$ 12.979,67	\$ 16.065,30	\$ 13.204,67	\$ 16.773,30
		Maz del Plata	\$ 10.596,95	\$ 13.062,39	\$ 11.499,58	\$ 14.175,03	\$ 12.074,57	\$ 14.883,03
		Tandil	\$ 10.596,95	\$ 13.062,39	\$ 11.499,58	\$ 14.175,03	\$ 12.074,57	\$ 14.883,03
Catamarca		San Fernando del Valle de Catamarca	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		Chaco	\$ 12.341,79	\$ 15.384,92	\$ 12.393,04	\$ 16.895,39	\$ 13.508,04	\$ 17.603,39
		Chubut	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		Puerto Madryn	\$ 13.808,53	\$ 17.417,14	\$ 15.049,84	\$ 18.900,71	\$ 15.024,84	\$ 19.608,71
		Córdoba	\$ 10.596,95	\$ 13.062,39	\$ 11.499,58	\$ 14.175,03	\$ 12.074,57	\$ 14.883,03
Corrientes		Corrientes Capital	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		Rio Cuarto	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		Paraná	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		Diamante	\$ 15.384,92	\$ 18.900,71	\$ 13.393,04	\$ 16.895,39	\$ 12.547,94	\$ 17.403,39
		Entre Ríos	\$ 12.843,79	\$ 15.384,92	\$ 12.682,09	\$ 15.750,25	\$ 13.257,69	\$ 16.658,26
Formosa		Formosa Capital	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		Jujuy	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		La Pampa	\$ 11.087,48	\$ 14.513,07	\$ 12.682,09	\$ 15.750,25	\$ 13.257,69	\$ 16.658,26
		Santa Rosa	\$ 13.050,43	\$ 17.126,82	\$ 14.813,16	\$ 18.585,66	\$ 15.308,16	\$ 19.393,67
		La Rioja	\$ 13.050,43	\$ 17.126,82	\$ 14.813,16	\$ 18.585,66	\$ 15.308,16	\$ 19.393,67
Mendoza		Mendoza Capital	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		Misiones	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		Neuquén	\$ 13.432,32	\$ 16.856,51	\$ 14.576,47	\$ 18.270,62	\$ 15.351,47	\$ 18.978,62
		San Carlos de Bariloche	\$ 13.432,32	\$ 16.856,51	\$ 14.576,47	\$ 18.270,62	\$ 15.351,47	\$ 18.978,62
		Rio Negro	\$ 13.050,43	\$ 17.126,82	\$ 14.813,16	\$ 18.585,66	\$ 15.308,16	\$ 19.393,67
Salta		Salta Capital	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		San Juan	\$ 12.996,11	\$ 16.256,87	\$ 14.103,09	\$ 17.440,53	\$ 14.678,09	\$ 18.348,53
		San Luis	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		San Martín	\$ 14.934,74	\$ 17.997,77	\$ 15.523,20	\$ 19.540,80	\$ 16.008,20	\$ 20.238,80
		Santa Cruz	\$ 14.934,74	\$ 17.997,77	\$ 15.523,20	\$ 19.540,80	\$ 16.008,20	\$ 20.238,80
Santa Fe		Puerto Deseado/Puerto San Julián	\$ 14.934,74	\$ 17.997,77	\$ 15.523,20	\$ 19.540,80	\$ 16.008,20	\$ 20.238,80
		Rosario	\$ 10.596,95	\$ 13.062,39	\$ 11.499,58	\$ 14.175,03	\$ 12.074,57	\$ 14.883,03
		Santa Fe Capital	\$ 11.033,16	\$ 13.643,02	\$ 11.972,94	\$ 14.805,12	\$ 12.074,57	\$ 15.513,12
		Santiago del Estero	\$ 12.905,58	\$ 14.804,29	\$ 12.919,68	\$ 16.065,30	\$ 13.204,67	\$ 16.773,30
		Tierra del Fuego	\$ 15.831,48	\$ 20.029,99	\$ 17.179,99	\$ 21.736,12	\$ 17.954,96	\$ 22.444,12
Tucumán		Uzuquén/Rio Grande	\$ 15.831,48	\$ 20.029,99	\$ 17.179,99	\$ 21.736,12	\$ 17.954,96	\$ 22.444,12
		San Miguel de Tucumán	\$ 12.341,79	\$ 15.384,92	\$ 13.393,04	\$ 16.895,39	\$ 13.508,04	\$ 17.603,39