Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

# Trabajo de Fin de Grado

## Análisis de eficiencia energética en procesadores ARM

*Analysis of energy efficiency in ARM processors*

Ángel Julián Bolaño Campos

La Laguna, 14 de julio de 2023

D. **Iván Castilla Rodríguez**, con N.I.F. 78.565.451-G profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-X profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

**C E R T I F I C A (N)**

Que la presente memoria titulada:

"*Análisis de eficiencia energética en procesadores ARM*"

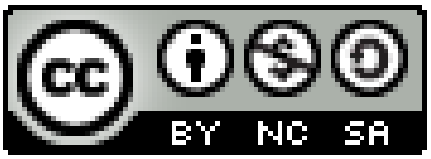
ha sido realizada bajo su dirección por D. **Ángel Julián Bolaño Campos**, con N.I.F. 79.065.477-X.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

# Agradecimientos

Agradezco la ayuda inconmensurable que me ha dado mi familia y la paciencia de mi pareja. Agradezco también a todos los profesores de la Universidad de La Laguna que ayudan a plantar la semilla de la curiosidad académica.

# Licencia



**Resumen**

*En este trabajo de fin de grado se investiga el consumo energético del procesador M1 de Apple y se realiza una comparación con un procesador INTEL para obtener una visión más general. Este tipo de comparaciones entre diferentes arquitecturas (RISC y CISC) no son comunes debido a las características distintivas de cada procesador. Sin embargo, dada la poca frecuencia en el uso de procesadores con arquitectura RISC en equipos portátiles, se llevará a cabo esta comparación en el marco de este trabajo.*

*Para garantizar la precisión de la recopilación de datos, se decidió evitar el uso de métodos convencionales de medición de energía que podrían introducir errores en los datos recopilados. Como resultado, se realizó un desarrollo significativo tanto en el hardware, diseñando y construyendo un dispositivo para realizar mediciones en equipos con arquitecturas diferentes, como en el software, al implementar el programa en dos sistemas operativos distintos.*

*Durante el desarrollo del estudio, se encontraron diversos obstáculos y se tomaron decisiones importantes, las cuales serán comentadas en el trabajo.*

*Finalmente, se presentarán los resultados obtenidos de las mediciones realizadas en cada uno de los equipos, ofreciendo un análisis comparativo del consumo energético entre el procesador M1 de Apple y el procesador INTEL. Estos resultados brindarán una visión más clara de las diferencias en eficiencia energética entre las dos arquitecturas evaluadas y contribuirán a una comprensión objetiva del avance que representa, en determinadas circunstancias, la utilización de procesadores RISC en equipos portátiles.*

**Palabras clave:** M1, MAC, INTEL, Rendimiento, Eficiencia, RISK, CISK, Arduino, ACS712, FZ0430

**Abstract**

*In this final thesis, the researcher investigate the energy consumption of Apple's M1 processor and compare it with an Intel processor to provide a comprehensive assessment of the results. The study involves conducting comparisons between different architectures, specifically RISC and CISC, which is not common due to the distinct characteristics of each processor. However, given the uncommon use of RISC processors in portable devices, the comparison is conducted within the scope of this project.*

*To ensure the accuracy of data collection, the researcher decide to avoid conventional energy measurement methods that may introduce errors. As a result, significant developments are made in both hardware and software. A device is designed and constructed specifically for measurements on different architecture systems, while the program is implemented on two different operating systems.*

*Throughout the project, the researchers encounter various obstacles and make important decisions, thoroughly discussed in the final report.*

*Finally, the study presents the measurement results of each processor, providing a comparative analysis of the energy consumption between Apple's M1 processor and the Intel processor. These results offer a clearer understanding of the differences in energy efficiency between the two evaluated architectures and contribute to an objective assessment of the advancements that the utilization of RISC processors can bring to portable devices in certain circumstances.*

**Keywords:** M1, MAC, INTEL, Performance, Efficiency, RISK, CISK, Arduino, ACS712, FZ0430

# Índice general

# Índice de Figuras

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. La evolución de ARM

Durante la última década, los procesadores ARM (Avanced RISK Machine) han experimentado un notable crecimiento en diversos ámbitos de la informática. En la actualidad, al considerar la adquisición de un ordenador, se priorizan características como la velocidad, potencia y precio, relegando durante mucho tiempo la consideración del consumo energético a un segundo o tercer plano. No obstante, a medida que hemos alcanzado límites físicos en términos de capacidad de procesamiento y densidad de transistores, la eficiencia energética se ha vuelto más relevante que nunca.

A partir de 2008, grandes empresas como Google y Amazon, al comenzar a ofrecer servicios en la nube, se encontraron con la necesidad de reducir costes, es decir, el consumo energético de sus servidores. De forma gradual, se introdujeron servidores con arquitectura ARM para tareas específicas, como el reconocimiento y procesamiento de imágenes, debido a su destacada eficiencia y rendimiento en dichas aplicaciones. De forma simultanea, los dispositivos móviles han experimentado un incremento significativo en su capacidad de procesamiento, y dado que funcionan con baterías, es imprescindible que el consumo de energía sea eficientes.

En 2020, Apple presentó su primer ordenador portátil de uso doméstico equipado con un procesador de arquitectura ARM, el MacBook Air con procesador M1. No fue el primer portátil con procesador ARM en el mercado ya que otras marcas habían realizado intentos previos de comercializar este tipo de equipos sin lograr los resultados esperados. El procesador M1 de Apple presume de rendimiento y eficiencia energética, llegando al punto de comparar sus resultados con los procesadores de arquitectura CISC de Intel o AMD.

## 1.2. Comparaciones entre RISC y CISC ¿reales?

Se pueden encontrar numerosas comparativas en internet entre el procesador RISC M1 de Apple y diferentes procesadores CISC de Intel y AMD. La mayoría de estas comparativas utilizan los mismos testbench para evaluar el consumo energético. Esto plantea interrogantes acerca de la fiabilidad de los testbench para reflejar

el consumo real. ¿Es posible que los procesadores proporcionen información no verídica por motivos de marketing?

Es importante tener en cuenta que la información sobre el consumo se obtiene directamente del propio procesador y esto significa que, sin importar el testbench utilizado para medir el consumo, los datos provienen de la misma fuente. Por lo tanto, para conocer de manera objetiva el consumo del procesador, es necesario contar con un método de medición externo que no dependa únicamente de los datos proporcionados por el procesador.

La mayoría de los datos disponibles en internet utilizan dispositivos que se conectan al transformador del ordenador para medir el consumo, pero este tipo de medición (Figura 1.1) puede llevar a obtener datos imprecisos. Estos dispositivos realizan una conversión de corriente alterna para obtener el consumo, pero no es posible conocer la frecuencia de muestreo ni tampoco es posible exportar los datos para su procesamiento. Para la investigación que se pretende realizar, es fundamental contar con una recopilación precisa del consumo.

El objetivo de este Trabajo de Fin de Grado es abordar el consumo energético de los procesadores de manera equitativa, tanto para el ordenador con el procesador M1 como para el ordenador con el procesador de Intel, utilizando herramientas de medición externas.


Figura 1.1: Medidor

## 1.3. Metodología

La metodología utilizada para el desarrollo del trabajo fue el siguiente:

1. Investigación y estado actual de las comparaciones de los procesadores.
2. Identificación y evaluación de diferentes enfoques para la captura de dato.
3. Consideración de distintas formas para obtener datos relevantes y confiables.

4. Diseños de dispositivo adaptados a los requisitos de captura de datos.
5. Búsqueda, recopilación y solicitud de los componentes electrónicos.
6. Montaje de dispositivo en una base.
7. Soldadura de conexiones.
8. Integración de los componentes electrónicos.
9. Verificación y calibración de datos obtenidos para equipo con procesador Intel,
10. Implementación de correcciones
11. Desarrollo de software con los requerimientos necesarios.
12. Verificación y calibración de datos obtenidos para equipo con procesador M1,
13. Implementación de correcciones.
14. Recopilación de pruebas.
15. Desarrollo software para comparación de datos y generación de gráficos.
16. Conclusiones

# Capítulo 2

# Materiales

## 2.1. Factores clave en el desarrollo del TFG

Antes de iniciar el desarrollo del Trabajo de Fin de Grado, se establecieron varios factores que debían tenerse en consideración para garantizar la veracidad de los resultados finales.

- **Pérdidas en los transformadores:** Se reconoció la presencia de pérdidas en los transformadores y se deberá evitar distorsiones en las mediciones de consumo energético.
- **Control del número de muestras:** Se consideró fundamental tener control sobre el número de muestras tomadas por segundo para obtener datos precisos.
- **Almacenamiento de datos:** Se identificó la necesidad de almacenar los datos obtenidos para su posterior análisis, permitiendo investigaciones más detalladas y comparativas en el futuro.
- **Homogeneidad en la toma de muestras:** Se reconoció la importancia de mantener la homogeneidad en la toma de muestras, lo que implicaba seguir un enfoque consistente y estandarizado para obtener mediciones comparables entre los diferentes procesadores evaluados.

Durante el desarrollo de este trabajo los desafíos más relevantes estuvieron directamente relacionados con estas consideraciones. Se procuró implementar soluciones ingeniosas y se tomaron todas las medidas posibles para garantizar la calidad y la fiabilidad de los resultados obtenidos.

## 2.2. Materiales utilizados

### 2.2.1. Equipos

Los equipos utilizados para este proyecto son:

- **MacBook Pro**
  - **Procesador:** M1
  - **RAM:** 16 GB
  - **Núcleos / Hilos:** 8 / 8
  - **Frecuencia núcleos de rendimiento(4):** 3.2 GHz
  - **Frecuencia núcleos de eficiencia(4):** 2.0 GHz
  - **Cache N1**: 2048 kB
  - **Cache N2**: 16 MB
  - **Sistema Operativo:** macOS Centura 13
- **ASUS ZenBook**
  - **Procesador:** Intel i7-10510U
  - **RAM:** 16 GB
  - **Núcleos / Hilos:** 4 / 8
  - **Frecuencia Max/Min:** 4.90 / 1.80 GHz
  - **Caché:** 8 MB Intel Smart Cache
  - **Sistema Operativo:** Windows 11

### 2.2.2. Alimentación

Con el fin de obtener mediciones precisas del consumo energético del procesador, se examina el escenario ideal donde se eliminan todos los componentes del ordenador que puedan interferir en dichas mediciones, como la pantalla, el wifi y el sistema operativo. Sin embargo, alcanzar este escenario resulta imposible.

Para poder acercar el trabajo a ese escenario ideal, se ha implementado una estrategia: excluir el transformador como variable. Con esta exclusión se busca incrementar la exactitud de las mediciones y disminuir posibles pérdidas. Para poder llevar a cabo esta estrategia, las mediciones se realizan entre el transformador y el ordenador.

### 2.2.3. Fuente de Laboratorio

Las fuentes de laboratorio son instrumentos que brindan la capacidad de ajustar el voltaje e intensidad según necesidades, lo que resulta especialmente útil para obtener mediciones precisas del consumo de corriente de los dispositivos conectados. Sin embargo, a pesar de ser una opción fiable, las fuentes disponibles carecen de salidas, como puertos seriales o USB que permitan transferir los datos de manera directa al ordenador para su análisis posterior, por lo que la utilización de una fuente de laboratorio quedó fuera de las opciones.

### 2.2.4. Arduino

Después de explorar otras alternativas, que abarcaron desde la búsqueda de fuentes de alimentación con capacidad de salida de datos hasta el diseño de un circuito integrado personalizado, se opto por una solución más simple y efectiva: utilizar una placa y sensores Arduino. Ésta decisión facilitó la obtención de las mediciones de intensidad y el voltaje de manera sencilla. Además, esta metodología permitió controlar la frecuencia de muestreo y almacenar los datos en un archivo para su posterior análisis.

Inicialmente, se consideró utilizar un único sensor para medir la intensidad. Sin embargo, se decidió modificar esta estrategia debido a que la alimentación del ordenador con el procesador M1 se basa en el protocolo Power Delivery a través del conector USB-Type C. Esto implica posibles variaciones en el voltaje durante las pruebas, por lo que el voltaje sería un dato relevante para obtener conclusiones más sólidas y precisas en el análisis de rendimiento. (Figura 2.1)


Figura 2.1: Arduino UNO

**Sensor de corriente ACS712-30A**

El módulo ACS712-30A (Figura 2.2) es un componente diseñado para medir la corriente utilizando un sensor de efecto Hall, capaz de detectar corrientes de hasta 30 Amperios. Para realizar la conexión, es necesario cortar el cable por el cual se desea medir la intensidad y colocar el módulo en serie.

El módulo cuenta con tres conexiones: una para la alimentación de 5V, otra para la conexión a tierra y una salida que debe ser conectada a uno de los pines de lectura analógica de la placa Arduino UNO. El sensor emite una señal de 2.5 Voltios cuando no hay corriente (0 Amperios) y a partir de este punto, la señal aumenta de manera proporcional a la intensidad que fluye a través del sensor. La placa Arduino mapea esta señal en un rango de 0 a 1023, y mediante una conversión adecuada, podemos obtener la lectura de corriente correspondiente.

En una gráfica la relación de voltaje con la corriente es una línea recta donde la pendiente es la sensibilidad.

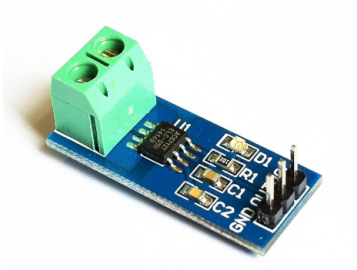


Figura 2.2: Sensor ACS712-30A

**Sensor de voltaje FZ0430**

El módulo FZ0430 (Figura 2.3)es un dispositivo utilizado para medir el voltaje, el cual incorpora un divisor de tensión compuesto por dos resistencias: una de 7.5K y otra de 30K. Este divisor de tensión permite convertir un voltaje de entrada en un rango de 0 a 25V a un voltaje de salida en el rango de 0 a 5V.

Al igual que el módulo ACS712 de medición de corriente, el FZ0430 también cuenta con tres conexiones: una para la alimentación positiva, otra para la conexión a tierra y una salida que debe ser conectada a uno de los pines de lectura analógica de la placa Arduino UNO.

El sensor emite un voltaje de salida que varía de 0V a 5V, el cual es interpretado por la placa Arduino. Esto permite realizar mediciones precisas del voltaje de entrada dentro del rango establecido y utilizar dicha información en los cálculos correspondientes.

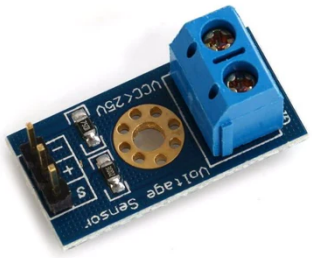



Figura 2.3: Sensor FZ0430

## 2.3. Herramientas

Las herramientas utilizadas para el montaje del dispositivo fueron:

- USB 3.2 Macho/Hembra (AISENS)
- Multímero
- Soldador
- Estaño
- Acido de soldadura
- Pinzas y Soportes
- Láminas de plástico
- Bridas

# Capítulo 3

# Diseño

Para el desarrollo del diseño se siguió el siguiente diagrama (3.1).

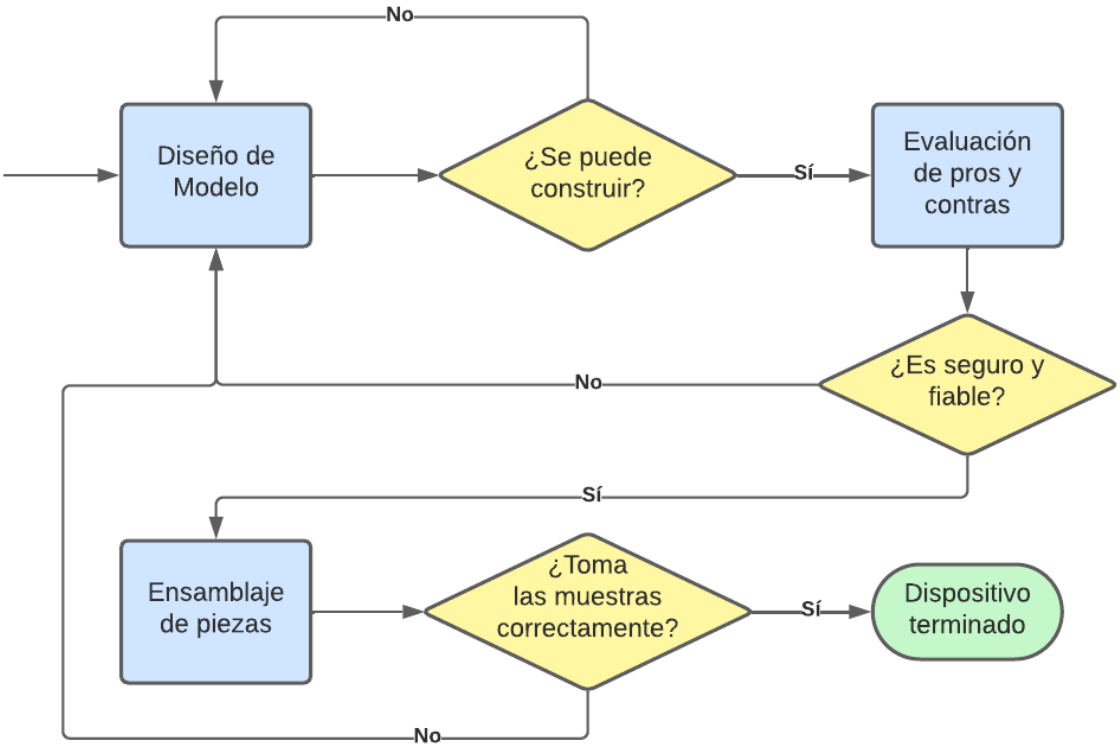



Figura 3.1: Boceto con conexiones estandarizadas

## 3.1. Primer modelo

Una vez que se establecieron los componentes necesarios para las mediciones, se procedió al diseño del dispositivo. Con el objetivo de lograr un dispositivo estandarizado y compatible con diversos equipos, se consideró la incorporación de conectores USB en la sección encargada de medir la corriente que atraviesa el procesador M1, así como conectores hembra para el ordenador con procesador Intel (Figura 3.2). Sin embargo, esta opción fue descartada debido a la dificultad

de obtener los conectores necesarios.

En dicha figura se puede observar que el sensor de voltaje no está presente, ya que en ese momento aún no se había considerado su necesidad en el diseño del dispositivo.

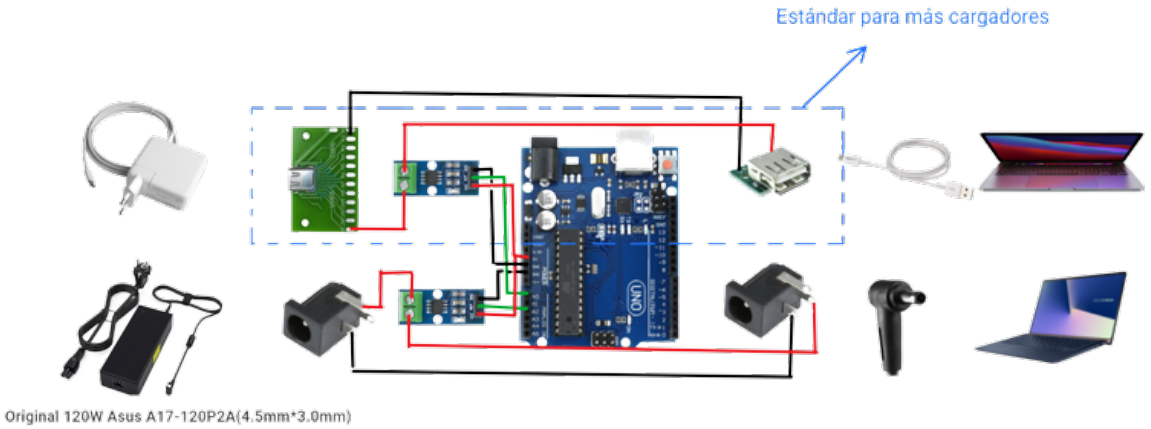



Figura 3.2: Boceto con conexiones estandarizadas

## 3.2. Segundo modelo

En el segundo diseño (Figura 3.3), se tomó la decisión de simplificar las conexiones por los problemas que se comentaron anteriormente. En su lugar, se optó por realizar un pequeño corte en los cables relevantes, permitiendo así una conexión directa entre el dispositivo de medición y los cables del sistema que se deseaban monitorear. Esta solución presentaba varias ventajas, ya que reducía la complejidad de las conexiones y eliminaba la necesidad de adquirir y utilizar conectores adicionales. También cabe destacar que al evitar la utilización de conectores USB se reducían las posibles interferencias o pérdidas de señal durante el proceso de medición. Este enfoque simplificado y directo en la conexión de los cables para la obtención de las mediciones permitía agilizar el proceso y maximizar la eficacia del dispositivo de medición.

### 3.2.1. Conector USB-Type C y MagSafe-3

Cuando se inició el diseño del proyecto, se consideró inicialmente el uso de un conector USB-Type C para el ordenador MacBook. Sin embargo, se pasó por alto la diversidad de conectores entre los modelos de MacBook, como el MacBook Air y el MacBook Pro. El modelo que se utilizaría para las mediciones, el MacBook Pro, utiliza el conector MagSafe 3 que es exclusivo de Apple. Aunque no se encontró documentación oficial sobre este cable, se pudo averiguar que además de la alimentación, el cable incorpora un protocolo de negociación de energía, posiblemente similar al PowerDelivery del USB-Type C, y también transmite información sobre el modelo, familia y serie del equipo al que se conecta. Debido a la incertidumbre y el riesgo potencial de dañar el equipo al manipular un cable del cual

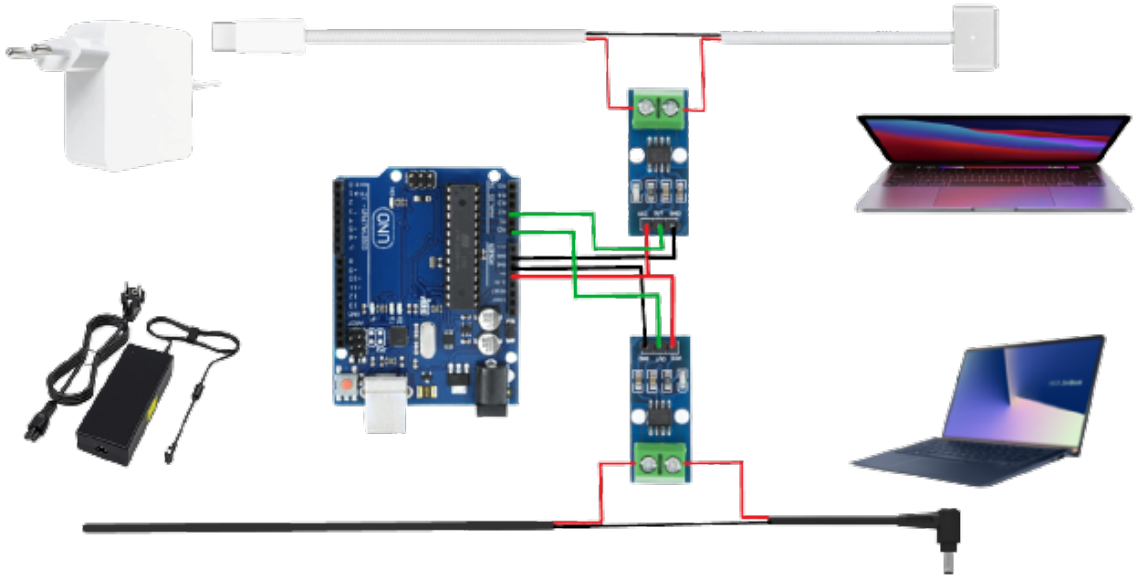


Figura 3.3: Boceto con conexiones fijas

no se tiene documentación clara, se tomó la decisión de modificar el diseño una vez más, optando por utilizar un alargador USB-Type C 3.2 que cuenta con documentación abierta y especificaciones claras.

### 3.2.2. Cargador Universal

Con el objetivo de preservar el cargador original del equipo, se consideró la opción de utilizar un cargador universal desde el inicio. Sin embargo, surgieron dificultades en este aspecto. Por un lado, no fue posible encontrar clavijas DC del tamaño necesario, como se muestra en la Figura 3.2. Por otro lado, obtener un cargador que cumpliera con las mismas especificaciones que el cargador original resultó más complicado de lo que se esperaba.

## 3.3. Tercer modelo

En el último diseño implementado (Figura 3.4, se logró recuperar la posibilidad de utilizar un enfoque estándar al emplear un cable USB-Type C. Sin embargo, conseguir este cable específico no fue una tarea sencilla debido a su relativa rareza en el mercado actual. Se requería un cable USB 3.2 que soportara el protocolo de Power Delivery para suministrar hasta 20V y 5A, cumpliendo así con los requisitos de alimentación. Un extremo del cable se conecta al cargador original del MacBook, mientras que el otro extremo se enlaza con el cable MagSafe3, preservando la integridad del cable original del portátil. La conexión con el ordenador equipado con el procesador de Intel se mantiene sin cambios.

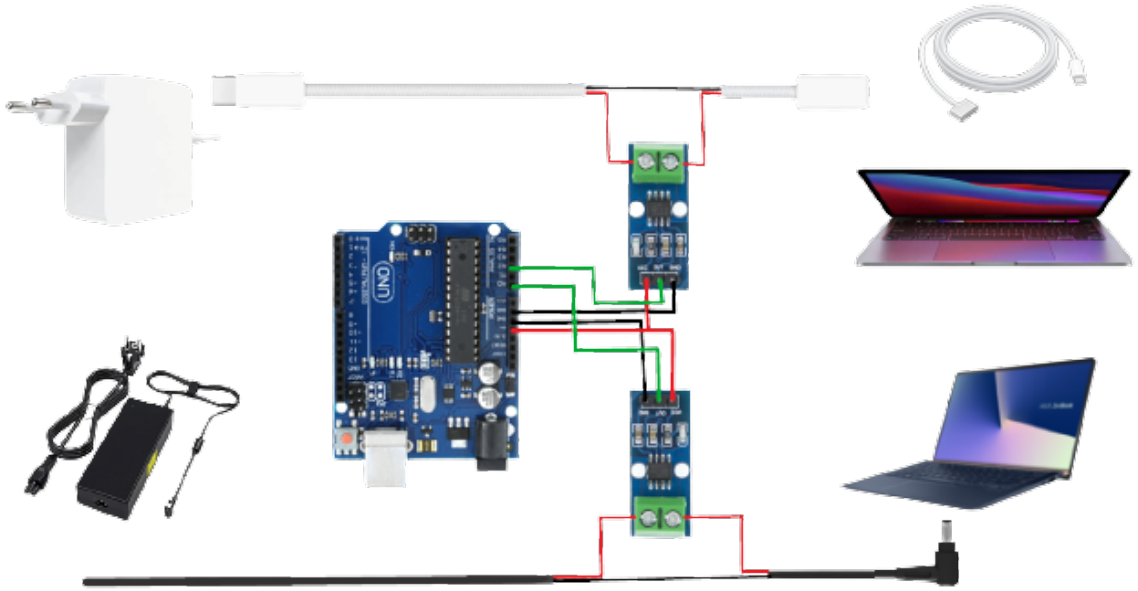


Figura 3.4: Boceto con UBC-Type C y conexiones fijas

# Capítulo 4

# Integración Hardware

## 4.1. Preparación del hardware

Esta sección abordará el ensamblaje de los distintos componentes y los desafíos surgidos durante este proceso. Se examinará detalladamente la fase de montaje, destacando los problemas encontrados y las estrategias adoptadas para resolverlos de manera efectiva. Este análisis permitirá comprender las dificultades técnicas enfrentadas y cómo se abordaron, brindando una visión completa del proceso de ensamblaje y las soluciones desarrolladas.

### 4.1.1. Cables del cargador universal

La elección de abordar esta sección en primer lugar se basa en su menor complejidad técnica. Según se muestra en la Figura 4.1, el transformador del cargador universal cuenta con tres cables: positivo, negativo y un tercer cable blanco específico que regula el voltaje y la tensión en base al tipo de clavija que se quiera utilizar. El cable rojo será utilizado para medir la intensidad eléctrica, mientras que, a través de un punto de soldadura en el cable negro permitirá la medición del voltaje. Con el propósito de lograr una configuración y manipulación sencillas de esta tarea, se han empleado terminales. Se puede apreciar en la Figura 4.4 las conexiones a ambos sensores.

### 4.1.2. Los sensores

Tanto el sensor de intensidad como el sensor de voltaje cuentan con tres conectores: positivo, negativo y señal. Para conectar estos sensores a la placa Arduino se emplearon cables de red. En la Figura 4.2 se aprecia que estos cables fueron soldados a los sensores y en cada extremo se añadió un pin para facilitar su conexión posterior a la placa. Esta configuración permitió establecer una conexión robusta y confiable entre los sensores y la placa Arduino.

Con el fin de facilitar la manipulación de la placa se decidió fijar los sensores y cables a una placa de madera. Si bien esta elección se basó en la necesidad de tener una base sólida, el uso de madera no era la solución ideal debido a posibles riesgos de derivaciones o cortocircuitos. Por lo tanto, como se puede apreciar en la parte derecha de la Figura 4.3, se introdujeron láminas de plástico entre los componen-

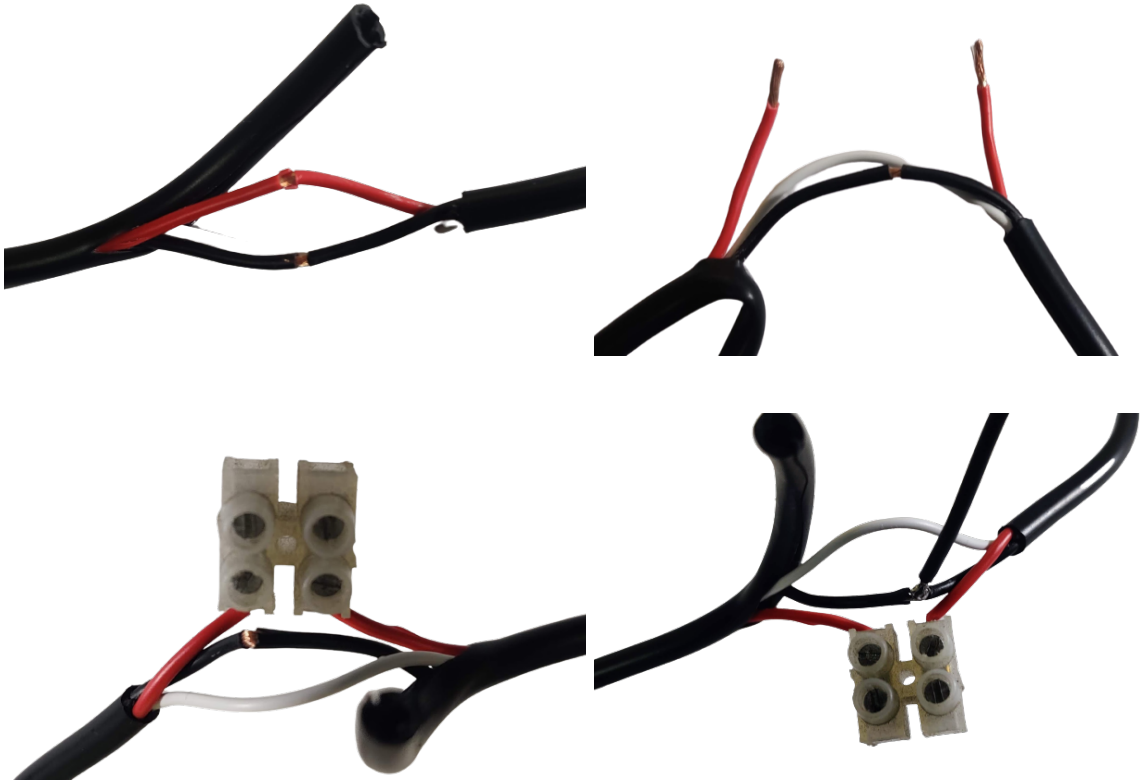


Figura 4.1: Cortado cable de cargador universal

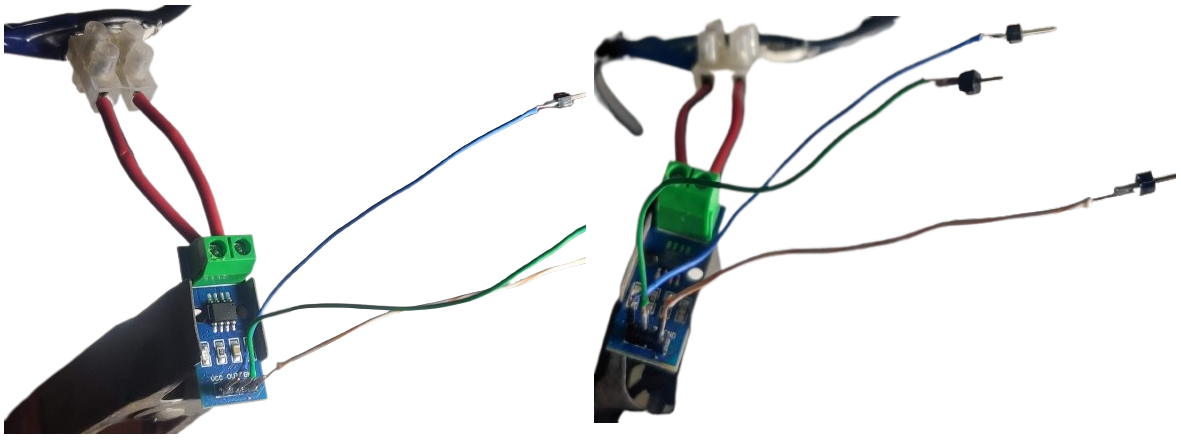


Figura 4.2: Soldaduras de cables

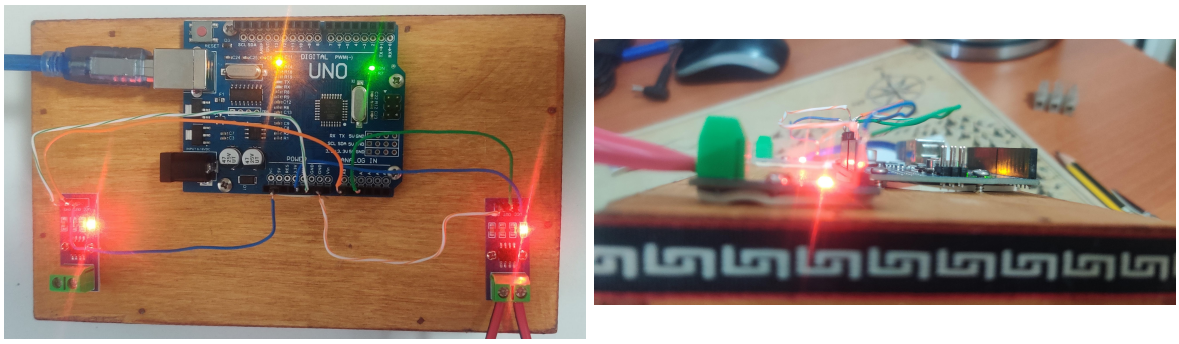



Figura 4.3: Base para placa y sensores

tes eléctricos y la base de madera. Esta medida se implementó con el objetivo de prevenir posibles interferencias eléctricas y garantizar un entorno seguro para el funcionamiento de los componentes. Las láminas de plástico proporcionan un aislamiento adecuado, minimizando los riesgos.

### 4.1.3. Conexiones y verificación

En la Figura 4.4 se muestra la inclusión de los sensores de voltaje en el diseño. Sin embargo, debido a la incorporación estos sensores, la placa Arduino se quedó sin suficientes conexiones de 5V y tierra. Para solucionar esta limitación, se agregó una pequeña placa protoboard para distribuir la energía necesaria a los sensores.

Una vez realizadas las conexiones, se procedió a introducir un amperímetro con el fin de verificar si los valores medidos por los sensores coincidían. Esta etapa de verificación permitió asegurar la precisión y la fiabilidad de los datos obtenidos por los sensores.

### 4.1.4. Cable USB-Type C

En el contexto de las conexiones del cable USB-Type C, es importante tener en cuenta que la versión requerida para el USB es la 3.2. Dicha versión es compatible con el protocolo Power Delivery y este protocolo permite la negociación dinámica de voltaje y corriente para lograr una carga eficiente y rápida de los dispositivos en los que se utiliza.

Para poder acceder a los cables se tuvo que tener mucho cuidado ya que son cables muy finos y todo ello viene con una protección exterior. Una vez que se removió la protección y apantallamiento del cable, se pudo observar que predominaban dos cables: uno rojo y otro negro (Figura 4.5). Al realizar una prueba utilizando un voltímetro, se comprobó que no había voltaje entre estos dos cables, pero sí se detectó voltaje en el apantallamiento externo. Erróneamente se dedujo que este último sería el cable de retorno. No obstante, se explicará más adelante la corrección de esta suposición equivocada

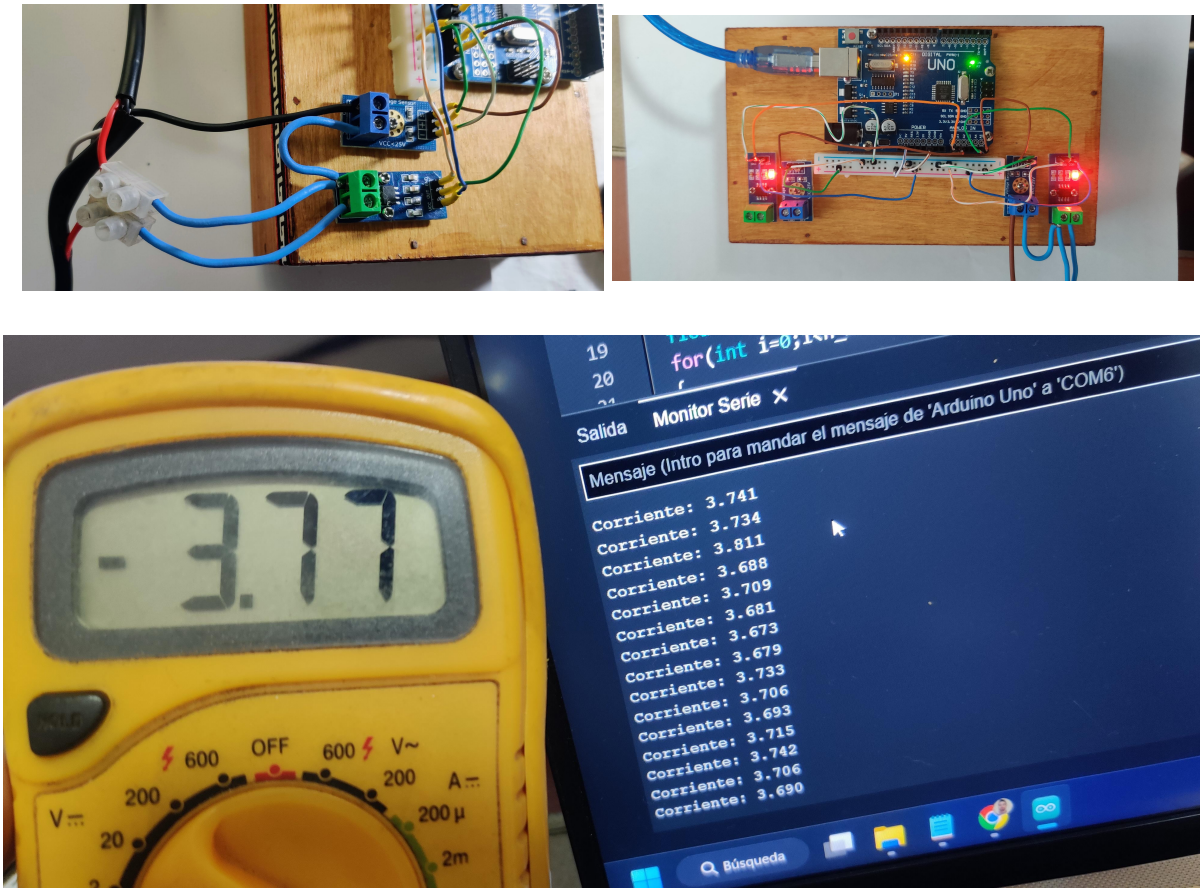



Figura 4.4: Conexiones y verificación

Figura 4.5: Cable USB-Type C sin apantallamiento

### 4.1.5. Conexiones USB-Type C

Durante la integración del cable USB en la estructura montada, se realizaron dos orificios laterales estratégicos para permitir la salida de los extremos del cable (Figura 4.6) y garantizar la protección adecuada de la zona expuesta. Para evitar el movimiento de los cables y prevenir posibles cortocircuitos, se aplicó silicona en los laterales y se crearon pequeños orificios para sujetarlos de forma segura.

Figura 4.6: Conexiones de cable USB inferior

Para sacar los cables necesarios se hicieron otros tres orificios (Figura 4.7) que se conectan a los sensores. Dos para medir la corriente y uno para el voltaje.

Finalmente podemos observar en la Figura 4.8 la placa terminada con ambos tipos de conexiones

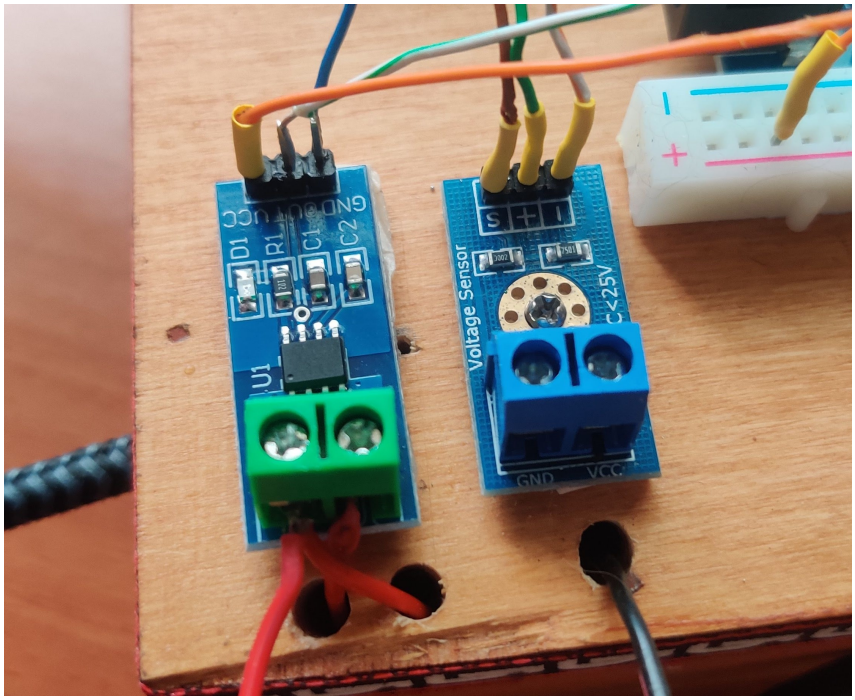



Figura 4.7: Conexiones de cable USB superior

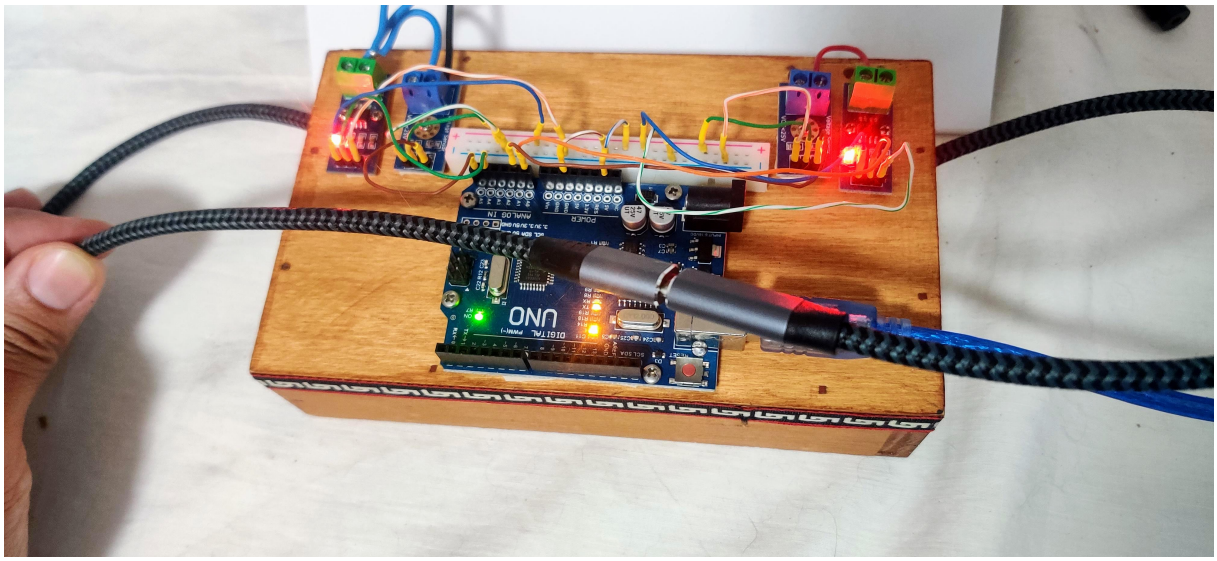



Figura 4.8: Placa terminada con ambos tipos de conexiones

# Capítulo 5

# Desarrollo Software

## 5.1. Preparación del Software

En este apartado, se abordarán las decisiones relacionadas con el desarrollo del software utilizado para la captura y análisis de las muestras obtenidas con la placa Arduino.(Véase A)

Se debe recalcar que no se cuenta con una disponibilidad absoluta del equipo con el procesador M1 por lo que todo se desarrollará primero en el entorno Windows y luego se verificará su compatibilidad en un entorno de iOS.

Se discutirán las decisiones tomadas para el procesamiento y visualización de las mediciones, así como las consideraciones de usabilidad y compatibilidad que se tuvieron en cuenta para su ejecución en distintos sistemas operativos.

### 5.1.1. Configuración de Arduino

Para la configuración y programación en Arduino se llevaron a cabo una serie de pruebas con la finalidad obtener el muestreo que realizaba con las funciones de lectura analógica y la capacidad aritmética de la placa.

En una primera aproximación de lectura, se observó que la placa Arduino proporcionaba, aproximadamente, 8300 muestras por segundo. Si bien esto representa un nivel de precisión satisfactorio, se había acordado que obtener alrededor de 200 muestras por segundo sería suficiente para este estudio. Por lo tanto, se decidió añadir una pausa de 3 ms entre cada muestra, lo que permitió obtener una media aproximada de 234 muestras por segundo.

Posteriormente, se llevó a cabo la conversión de los valores obtenidos por el sensor con el fin de obtener la intensidad correspondiente en una escala que se pueda entender. Dado que el sensor de intensidad ACS712, alimentado con 5 V, almacena los datos en un registro de 10 bits donde el valor 512 (equivalente a 2,5 V) corresponde a 0 A, se realizaron las operaciones aritméticas pertinentes (ver Apéndice A-A1, líneas 17 y 18) para convertir estos valores en unidad de amperios.

Tras verificar la precisión en la lectura de los valores, se procedió a integrar la captura de datos provenientes del sensor de voltaje. Este sensor tiene una capacidad máxima de 25 V y, al igual que el sensor ACS712, transmite los valores en un registro de 10 bits. En este registro, el valor 0 representa 0 V, mientras que el valor

1023 corresponde a 25 V. (Apéndice A-A2)

Sin embargo, la inclusión de operaciones aritméticas en la placa Arduino ocasionó una drástica reducción en el número de muestras por segundo, pasando de alrededor de 8300 a aproximadamente 80. Estas operaciones aritméticas con números de punto flotante representaron una carga significativa para el microcontrolador de la placa Arduino (ATmega328P). Se intentó abordar este problema utilizando preescalares para aumentar la velocidad del reloj y reestructurando el código para optimizarlo, pero los resultados fueron insatisfactorios.

Finalmente, la placa Arduino simplemente envía los datos a través del puerto serie para ser leídos desde el programa principal. (Apéndice A-A3)

### 5.1.2. Estructura de código en Python

La necesidad de poder replicar las mismas condiciones de ejecución en distintos sistemas operativos con procesadores M1 e Intel era de vital importancia. Para lograrlo, se optó por desarrollar un programa en Python que recibiera a través de la comunicación serial los datos capturados por los sensores.

A continuación, se explicará el flujo del programa (Figura 5.1) detallando las decisiones tomadas.

Con la idea de minimizar el uso de la memoria del sistema y garantizar un entorno de pruebas lo más óptimo posible, se decidió almacenar los datos en archivos en lugar de hacerlo en la memoria principal. Esta elección se fundamenta en el hecho de que al realizar las pruebas en el mismo equipo encargado de almacenar los datos podemos incurrir en una limitación indeseada de los recursos disponibles para la ejecución del propio test y estaríamos privando al equipo de una cantidad importante de memoria.

Para implementar esta estrategia, se procedió a crear y abrir múltiples archivos antes del inicio de la prueba y cerrarlos al finalizar dicho test. El uso de archivos diferenciados permitió delimitar y registrar los diversos estados del equipo durante el desarrollo de las pruebas. Para obtener una referencia del estado inicial del equipo, se toma una muestra antes de dar inicio a la prueba y, después de la ejecución del testbench, se obtiene una muestra de cómo esta el equipo al finalizar la prueba.

Mediante esta metodología de almacenamiento en archivos, se logra un manejo eficiente de los recursos de memoria y se garantiza la integridad de los datos generados durante las pruebas.

Además de los tres archivos mencionados anteriormente, se crean tres archivos adicionales:

- Un archivo de información para almacenar los detalles del entorno de prueba como los tiempos de cada toma de medidas y el número de muestras obtenidas en cada prueba.
- Un archivo que es una combinación de los archivos de prueba para tener una visión global de toda la prueba.

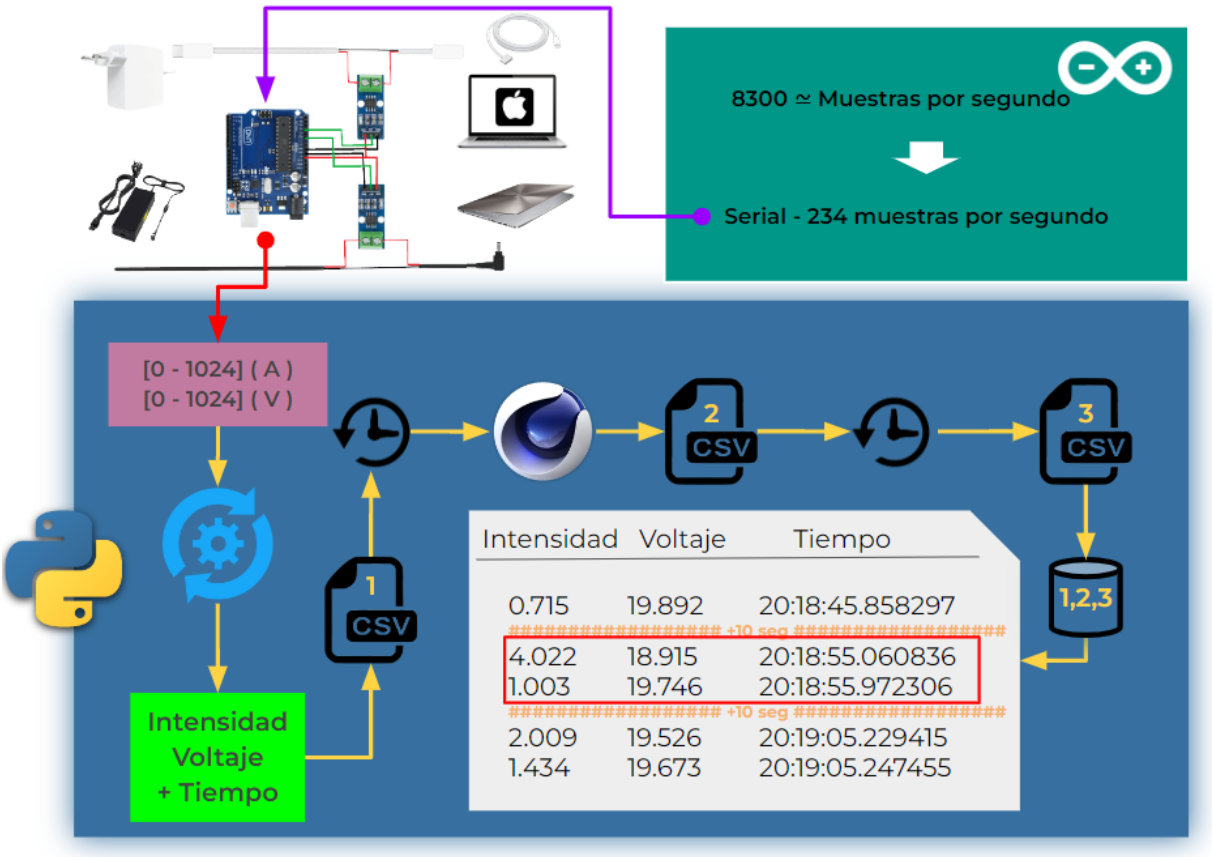



Figura 5.1: Flujo del programa

- Un archivo que contiene un filtrado de las pruebas globales.

Una vez finalizada la toma de datos, se cierran los archivos y se procede a su lectura. Se optó por este enfoque para evitar el procesamiento de datos durante el desarrollo de la prueba. Finalmente, con los datos del archivo global y su filtrado, se generan gráficas en formato PNG para cada archivo. (Figura 5.2)

Por último cabe destacar que el programa ofrece la posibilidad de introducir distintas configuraciones a través de la línea de comandos. Entre las configuraciones disponibles se encuentran poder seleccionar el sistema operativo, el puerto serial y su velocidad, los tiempos de ejecución, el tipo de ejecución, el tipo de filtrado y la generación de gráficos. En la Figura 5.3 se muestra un ejemplo de la ayuda desplegada. Para más información véase B.

### 5.1.3. Verificación del entorno en Windows

Este apartado se enfoca en la verificación del funcionamiento del hardware y software desarrollados en el entorno del sistema operativo Windows. El objetivo principal es garantizar que todas las condiciones necesarias se cumplan antes de proceder con la recolección de datos.

Para lograr esta verificación, se llevaron a cabo una serie de pasos y pruebas que se describen a continuación:

1. **Verificación de los sensores:** Se comprobó que los sensores estuvieran co-

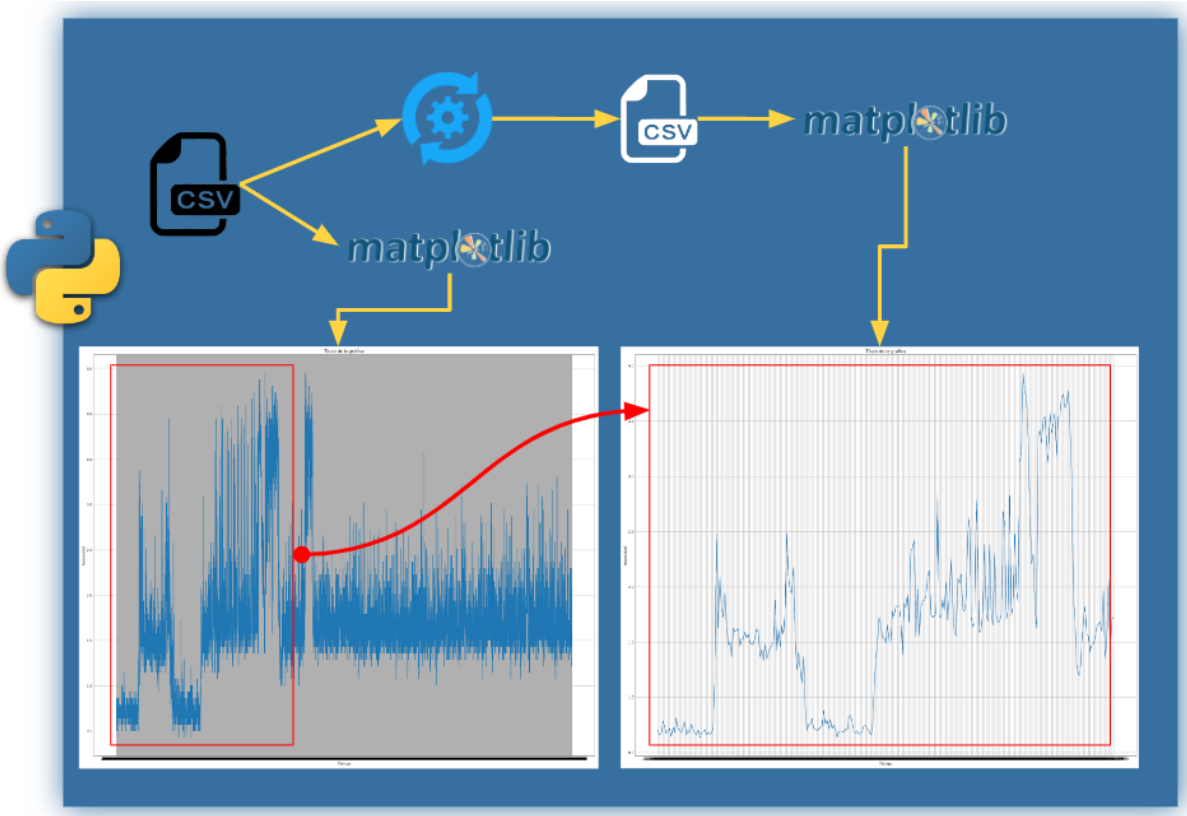



Figura 5.2: Conversión a gráficas

rrectamente conectados y funcionando adecuadamente. Se verificó que los datos capturados por los sensores fueran coherentes y consistentes.

2. **Comunicación serial:** Se realizó una prueba de comunicación serial entre los sensores y el programa en Python. Se verificó que los datos se recibieran correctamente.

3. **Calibración de los sensores:** Se realizaron ajustes necesarios en caso de desviaciones o errores en las lecturas.

4. **Ejecución del programa:** Durante esta etapa, se verificó que el entorno se configure correctamente de acuerdo a los parámetros suministrados a través de linea de comando. Esto involucró varios aspectos clave, entre ellos:

    - **Control del testbench:** Se verificó la capacidad de cerrar el testbench en un tiempo especificado o al finalizar la prueba funcionase correctamente.
    - **Núcleos utilizados:** Se aseguró que el testbench pueda realizar operaciones utilizando un solo núcleo o aprovechando todos los núcleos disponibles.
    - **Control de tiempos de mediciones:** Se garantizó que el control de los tiempos de las mediciones antes y después de la prueba se establezcan correctamente.

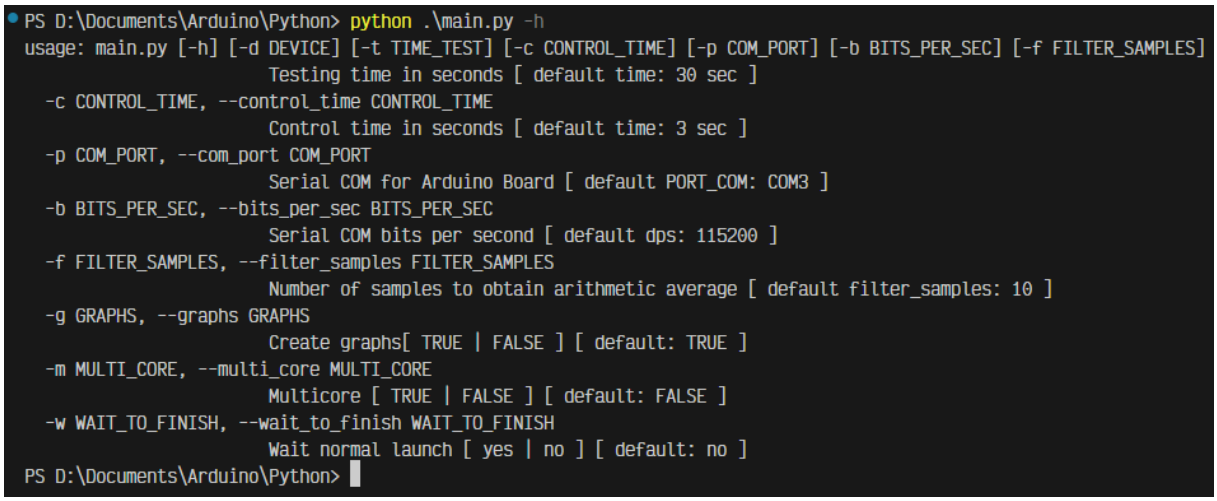



Figura 5.3: Menú de ayuda

- **Configuración del puerto serial y velocidad de comunicación:** Se verificó que la configuración del puerto serial y la velocidad de comunicación, se establezcan en el programa correctamente.
- **Configuración del número de muestras para el filtrado:** Se verificó que la configuración del número de muestras utilizado en el proceso de filtrado se pueda configurar fácilmente.
- **Generación de gráficos:** Se comprobó que las gráficas se crean en el formato correcto, tamaño correcto y ubicación.

5. Prueba de funcionamiento general: Se realizó una prueba integral en el entorno de **Windows** del funcionamiento general del entorno.

La verificación del funcionamiento del entorno se llevó a cabo de manera minuciosa y rigurosa, asegurando que todas las condiciones necesarias estuvieran cumplidas antes de iniciar la toma de datos.

## 5.2. Verificación del entorno para iOS

Como se mencionó previamente, debido a la falta de disponibilidad de un ordenador con iOS, se llevaron a cabo las verificaciones del hardware utilizando un dispositivo móvil para probar el cable USB Type C 3.2. Se comprobó que los sensores estuvieran recibiendo el voltaje y amperaje adecuados utilizando los valores medidos por un voltímetro como referencia. La preparación del entorno software se llevó a cabo de manera remota, permitiendo adelantar la comprobación del programa en iOS antes de acudir con el dispositivo hardware.

### 5.2.1. Software

Debido a que la verificación del programa en el entorno iOS tuvo que realizarse de manera remota, es importante mencionar que algunos aspectos, como la comprobación de la conexión serial, no pudieron ser probados en esta etapa. No

obstante, se consideró fundamental llevar a cabo la preparación previa del equipo con el fin de optimizar el tiempo durante las pruebas.

A continuación, se detallan las acciones mas importantes llevadas a cabo en la preparación del entorno software:

- **Preparación del entorno software:** Se procedió a instalar en el entorno iOS el lenguaje Python, así como todas las librerías necesarias para el funcionamiento del programa desarrollado.
- **Control del testbench:** Se verificó la capacidad de lanzar y cerrar el testbench utilizando la librería subprocess(Popen). Fueron necesarios algunos ajustes en la invocación de la aplicación, pero se logró obtener un funcionamiento adecuado.
- **Configuración serial:** La configuración de la comunicación serial en iOS es similar a la de Linux, utilizando un archivo en la ruta /dev/tty. Se implementó una función que garantizó la correcta configuración de las variables encargadas de establecer la conexión serial con Arduino.
- **Estructura de datos y carpetas:** Se verificó que los archivos y la estructura de datos se crearan correctamente durante la ejecución del programa, asegurando así una organización adecuada.

Una vez preparado el entorno software en iOS se procedió a acudir con el dispositivo hardware para realizar una comprobación mas exhaustiva.

# Capítulo 6

# Incidencias durante la verificación

## 6.1. Conexión al MAC

Antes de conectar la placa Arduino al equipo MAC, se llevó a cabo una verificación del cable USB utilizado para asegurarse de su correcto funcionamiento y de que el protocolo Power Delivery no se viera afectado por posibles cortes. Durante la comprobación, se obtuvieron resultados exitosos, ya que al conectar el cable USB al MAC, se registró un voltaje promedio de 20V utilizando un voltímetro externo.

Una vez confirmado que el cable suministraba la energía necesaria, se procedió a conectar la placa Arduino al MAC. Sin embargo, se encontró un inconveniente: la falta de puertos USB-A en el equipo MAC por lo que para realizar la conexión se tuvo que utilizar un adaptador. Al conectar la placa Arduino al MAC la placa Arduino se apagó y el MAC dejó de cargar. Ante esta situación, se desconectó la placa Arduino y se revisaron cuidadosamente las conexiones en los diferentes pines para descartar cualquier error de conexión.

Luego de comprobar las conexiones se volvió a probar con idéntico resultado. Así que se llevó a cabo una prueba conectando la placa Arduino a un ordenador personal. Resultó interesante observar que, en este caso, la placa Arduino no experimentó apagones, y el ordenador MAC continuó cargando. Este evento condujo a considerar la posibilidad de que el adaptador utilizado previamente presentara algún problema o incompatibilidad, por lo cual se decidió realizar una nueva prueba utilizando otro adaptador disponible.

Sin embargo, al conectar la placa Arduino por tercera vez al equipo MAC, se produjo una situación inesperada: la placa Arduino dejó de funcionar por completo. Este contratiempo indujo a pensar que el problema no fueron los adaptadores.

Después de consultar con los profesores, se determinó de manera clara que existía un problema: había indicios de un cortocircuito en alguna parte del hardware diseñado.

### 6.1.1. Derivación a tierra

Después de investigar el incidente, se determinó que posiblemente se estaban tomando los cables erróneos para la corriente y el voltaje. De la investigación resultante con los cables se debe destacar que el interior de los cables USB Type-C 3.2

puede variar entre fabricantes ya que no existe un estándar de colores establecido para estos cables. Por ejemplo, el estándar internacional establece que los colores rojo y negro (Figura 3.5) son para suministrar energía, no obstante se verificó que no existía una diferencia de potencial entre ellos, pero sí respecto al recubrimiento del cable. De manera equivocada, se asumió que esos eran los cables adecuados para realizar las mediciones.

Al no poder encontrar información sobre la codificación de colores en la página del fabricante del USB (AISENS), se decidió buscar ejemplos que se asemejaran a los cables USB en cuestión. En la Figura 6.1, se indica que un cable rojo es utilizado para la alimentación, mientras que hay otros dos cables más pequeños y estañados que se utilizan para el retorno. Al realizar pruebas con un multímetro se confirmó que esos cables más pequeños, estañados y sin recubrimiento, correspondían al retorno. Así de este incidente se deduce que lo que inicialmente se creía que era el retorno, en realidad resultó ser la conexión a tierra y esto provocó una derivación que terminó por quemar la placa Arduino.

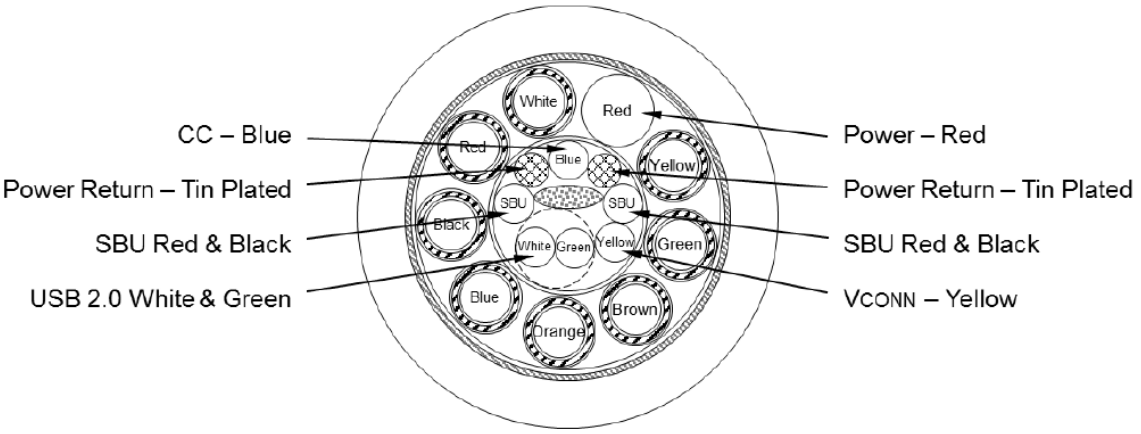



Figura 6.1: Corte transversal de cable USB-Type C

## 6.2. Cables desconectados

Una vez solucionado el problema, se procedió a reconectar la placa Arduino al MAC con resultados favorables. Tras verificar que la ejecución del programa era correcta, se procedió a tomar muestras. Los resultados fueron desconcertantes, según se muestra en la gráfica de la Figura 6.2. Se observó que el procesador consumía menos energía a medida que trabajaba más, llegando incluso a presentar un consumo muy por debajo de lo esperado.

Al realizar una revisión de los cables, se constató que el cable rojo utilizado para realizar las mediciones se encontraba cortado, posiblemente debido a los diversos ajustes y movimientos a los que fue sometido. Sin embargo, este error ayudó a esclarecer un aspecto importante. A pesar de que el cable rojo estaba cortado, el ordenador continuaba recibiendo carga. Esto se debía a que los cables rojo y negro tienen la misma función y características, por lo que la energía seguía circulando a través del cable negro.

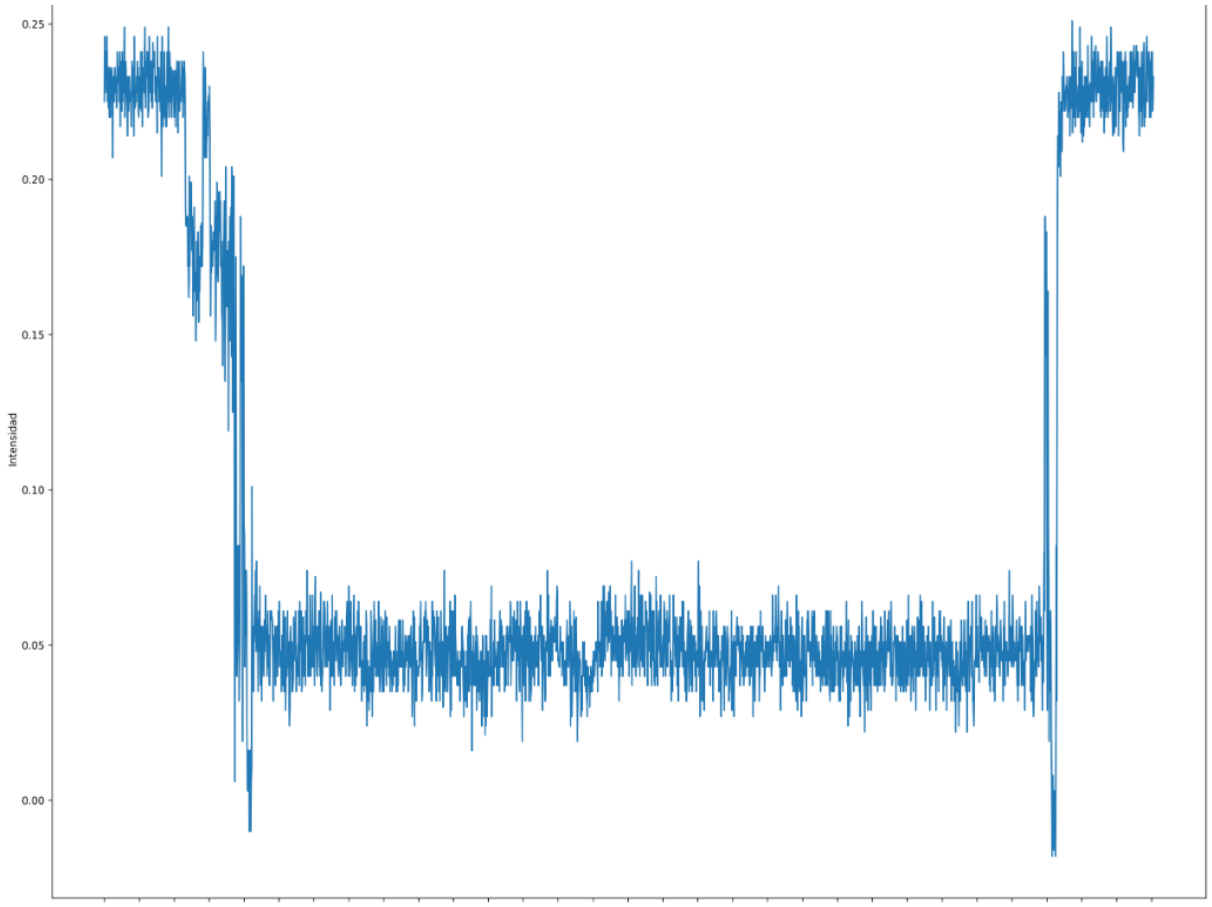



Figura 6.2: Gráfica con datos confusos

Con esta nueva información, se tomó la decisión de cortar el cable negro, permitiendo que toda la energía fluyera exclusivamente a través del cable rojo. Esto permitiría obtener resultados fiables.

# Capítulo 7

# Resultados de las pruebas

## 7.1. Eficiencia energética en procesadores ARM

Durante este trabajo de fin de grado, se desarrolló un dispositivo hardware específico, junto con su parte software correspondiente, para obtener datos sobre el consumo de energía de equipos con procesadores M1 e Intel. Esto evitó la dependencia de dispositivos externos con frecuencias de muestreo desconocidas y la información proporcionada por los propios procesadores durante las pruebas. La eliminación de incertidumbres asociadas a otros métodos de medición garantizó la obtención de datos más objetivos y precisos.

## 7.2. Obtención de los datos

El dispositivo hardware desarrollado ha registrado variables como el voltaje, amperaje y tiempo en que fue tomada la muestra. Como se mencionó en capítulos anteriores, la frecuencia de muestreo promedio es de 230 muestras por segundo, obteniendo un total de aproximadamente 40 mil muestras durante cada prueba que se realizo. Cabe destacar que, para obtener los datos de consumo en vatios, se ha utilizado un programa adicional (véase Apéndice 3). Este otro programa ha permitido un análisis posterior a la toma de datos de forma rápida.

Las variantes en cada prueba han sido:

1. **Dispositivo utilizado (MacBookPro M1 / AsusZenbook Intel)**
2. **Tipo de ejecución (Mono-core / Multi-core)**
3. **Tiempo de la prueba**

## 7.3. Comparativa de gráficas

Las siguientes gráficas presentarán datos de voltaje, amperaje y consumo en vatios obtenidos en distintas pruebas realizadas en un mismo dispositivo con las mismas configuraciones. Es importante tener en cuenta que cada prueba genera tres archivos, que corresponden al consumo del procesador antes, durante y después de la prueba. En las gráficas siguientes se mostrará el conjunto completo, con las zonas delimitadas para cada intervalo.

### 7.3.1. Procesador M1

Las siguientes gráficas representan distintas pruebas en las que se comparan el voltaje (Figura 7.3), amperaje (Figura 7.2) y vatios (Figura 7.1) bajo las mismas condiciones y con la siguiente configuración:

- **Tipo de ejecución:** Multi-core
- **Tiempo de la prueba:** 10 s de control al inicio y fin más 90 s de prueba

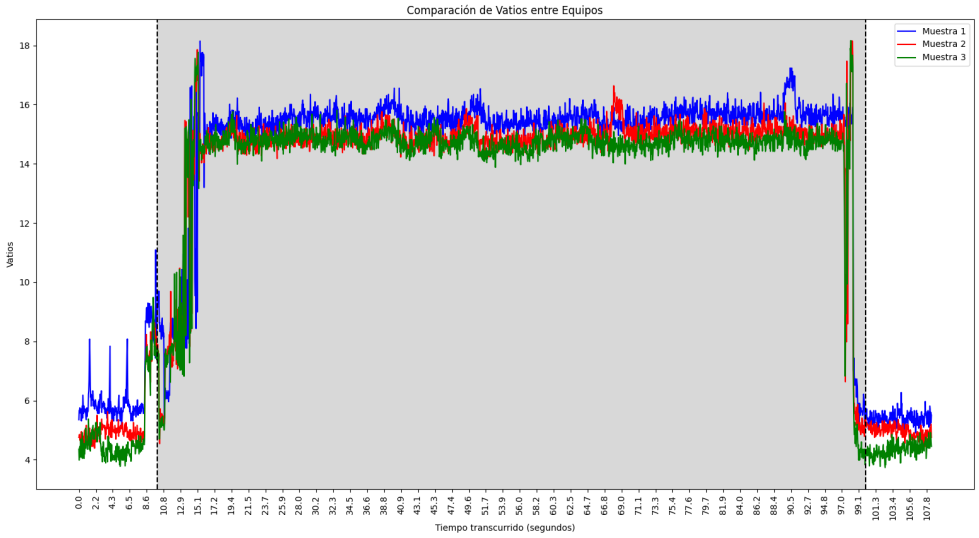



Figura 7.1: Gráfica de Vatios Mac

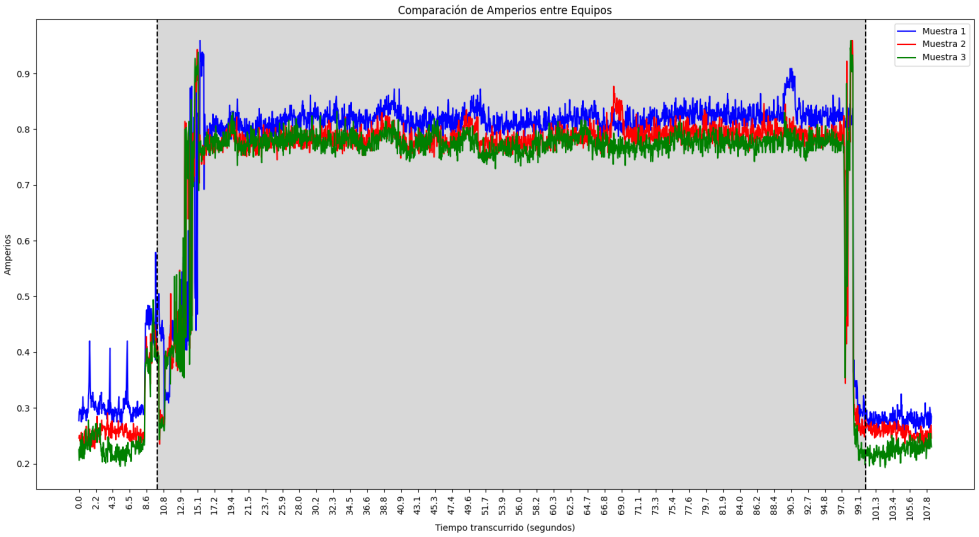



Figura 7.2: Gráfica de Amperaje Mac

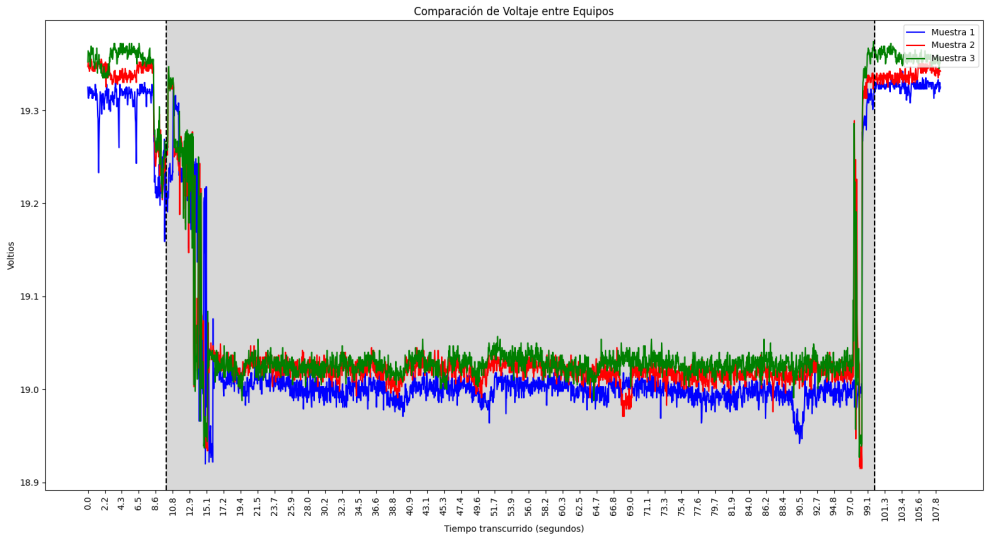



Figura 7.3: Gráfica de Voltaje Mac

### 7.3.2. Procesador Intel

De igual forma que la sección anterior, las siguientes gráficas se representan distintas pruebas comparando voltaje (Figura 7.6, amperaje (Figura 7.5 ) y vatios (Figura 7.4) en las mismas condiciones y con las siguientes configuraciones:

- **Tipo de ejecución:** Multi-core
- **Tiempo de la prueba:** 10 s de control al inicio y fin más 90 s de prueba

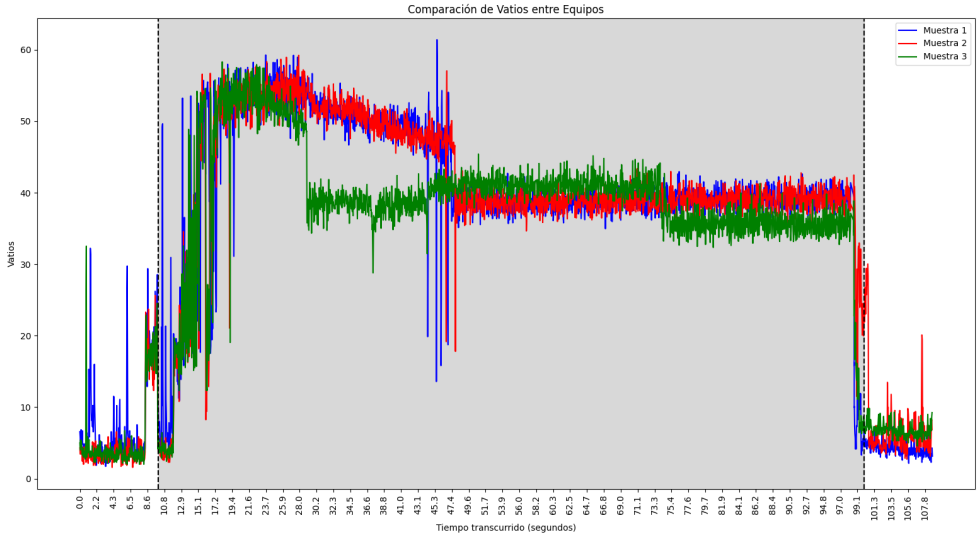



Figura 7.4: Gráfica de Vatios Intel

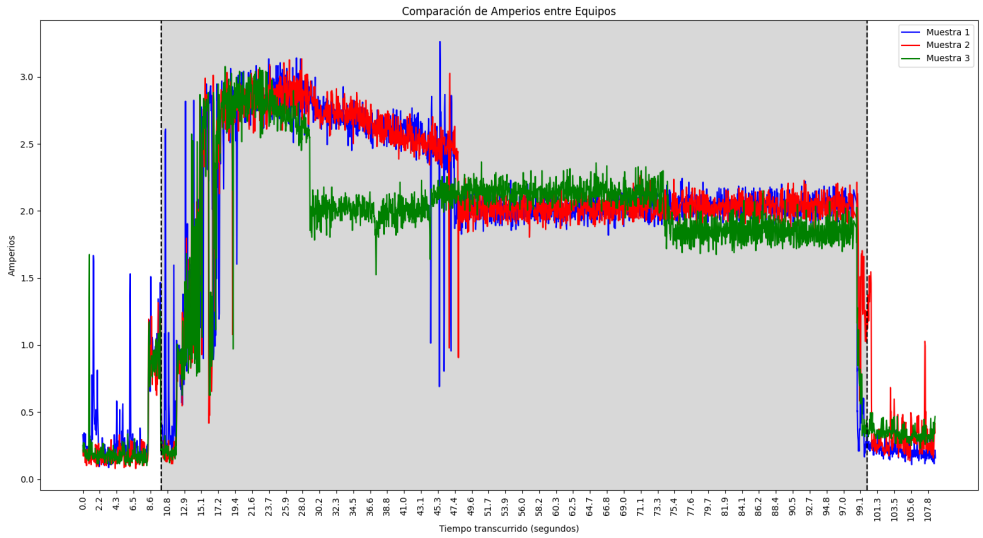



Figura 7.5: Gráfica de Amperaje Intel

## 7.4. Observaciones

Las gráficas revelan diferencias significativas entre el procesador M1 y el procesador Intel en términos de estabilidad. Ambos procesadores presentan un comportamiento similar en cada prueba, aunque se observa que el procesador M1 es notablemente más estable y eficiente en comparación con el procesador Intel.

En el caso del procesador M1, se puede apreciar que su comportamiento es prácticamente idéntico en cada iteración de las pruebas. Cuando el requerimiento energético aumenta el voltaje disminuye en proporción. Por el contrario, el procesador Intel muestra una mayor inestabilidad, además de que la tercera muestra (representada en color verde) difiere ligeramente en su comportamiento con respecto a las muestras 1 y 2, aunque sigue un patrón general similar. En todos los casos, el comportamiento de consumo sigue un patrón en el que, durante el primer cuarto del tiempo de prueba obtiene valores ligeramente más elevados, seguido de una disminución gradual en forma de escalones. No obstante, dado que el análisis detallado del consumo de energía de los procesadores Intel no es el objetivo principal de este Trabajo de Fin de Grado, el resto de los análisis se realizan sin la distinción de las diferentes áreas que generan los datos.

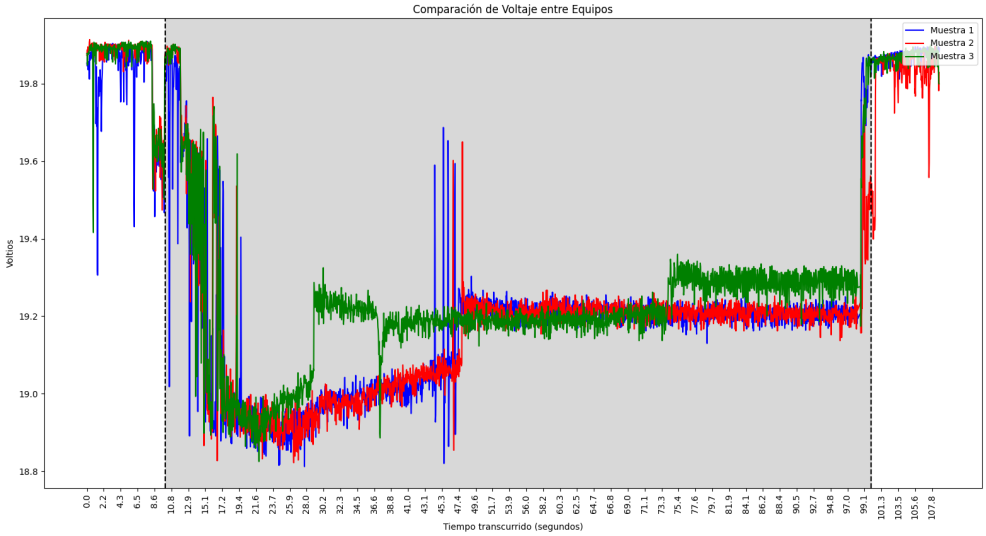



Figura 7.6: Gráfica de Voltaje Intel

## 7.5. Comparación gráfica de resultados

Con el fin de resaltar la notable diferencia en el consumo de energía entre los dos equipos, se presenta la gráfica 7.7 que ilustra los vatios consumidos por cada uno de ellos.

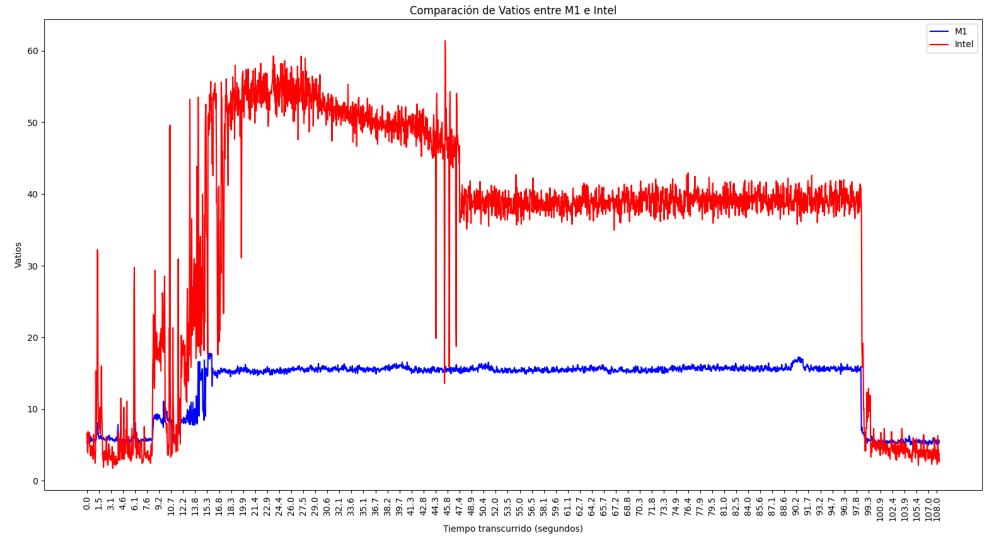



Figura 7.7: Gráfica comparativa de vatios

## 7.6. Análisis de datos

Dado que para las distintas pruebas los resultados son muy similares y no es necesaria la medida en diferentes escenarios, se muestran los datos en relación a una prueba para cada equipo. La muestra con la que se representan ambos cálculos es la **muestra 1**, no obstante en el apéndice C se encuentran las tablas de las demás muestras. Se a utilizado ésta configuración de multi-core por ser la más relevante e interesante de las pruebas.

En la tabla 7.1 se representan los datos del procesador M1 y la muestra 1 del mismo. En la tabla 7.2 se representan los datos de la muestra 1 del procesador Intel.

Tabla 7.1: Tabla de Medias M1 (Muestra 1)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.35s) | Voltaje | (19.313, 19.315) | 19.314 V |
| | Amperaje | (0.301, 0.303) | 0.302 A |
| | Vatios | (5.797, 5.857) | 5.827 W |
| Testbench (90s) | Voltaje | (19.014, 19.016) | 19.015 V |
| | Amperaje | (0.792, 0.794) | 0.793 A |
| | Vatios | (15.05, 15.099) | 15.075 W |
| Final (10s) | Voltaje | (19.321, 19.323) | 19.322 V |
| | Amperaje | (0.285, 0.288) | 0.287 A |
| | Vatios | (5.505, 5.571) | 5.538 W |
| Global (108.35s) | Voltaje | (19.065, 19.068) | 19.066 V |
| | Amperaje | (0.706, 0.711) | 0.708 A |
| | Vatios | (13.435, 13.524) | 13.479 W |

Los datos recopilados en las tablas confirman la estabilidad y eficiencia en el consumo del equipo con el procesador M1 en comparación con el equipo que emplea el procesador Intel. Los resultados evidencian una clara ventaja en términos de estabilidad y eficiencia energética del procesador M1, respaldando su rendimiento en comparación con el procesador Intel.

Tabla 7.2: Tabla de Medias Intel (Muestra 1)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.33s) | Voltaje | (19.851, 19.859) | 19.855 V |
| | Amperaje | (0.256, 0.281) | 0.268 A |
| | Vatios | (5.068, 5.52) | 5.294 W |
| Testbench (90s) | Voltaje | (19.161, 19.165) | 19.163 V |
| | Amperaje | (2.168, 2.183) | 2.175 A |
| | Vatios | (41.445, 41.713) | 41.579 W |
| Final (10s) | Voltaje | (19.847, 19.854) | 19.851 V |
| | Amperaje | (0.265, 0.284) | 0.274 A |
| | Vatios | (5.241, 5.606) | 5.424 W |
| Global (108.33s) | Voltaje | (19.278, 19.285) | 19.282 V |
| | Amperaje | (1.837, 1.858) | 1.847 A |
| | Vatios | (35.152, 35.534) | 35.343 W |

# Capítulo 8

## Conclusiones y líneas futuras

Durante el desarrollo de este trabajo de fin de grado, se abarcó un amplio espectro de ramas de la informática. La parte más difícil fue la implementación y desarrollo del dispositivo de hardware. Las diferentes conexiones de los equipos, así como la disponibilidad de materiales y la documentación limitada, como el caso del MagSafe de Apple, ralentizaron el progreso del proyecto. A pesar de ello, los resultados obtenidos de las pruebas realizadas con el procesador M1 fueron muy relevantes. Al compararlos con las muestras obtenidas del procesador Intel, se confirmó la eficiencia del rendimiento del procesador M1.

A medida que avanzaba el proyecto, surgieron nuevas preguntas que podrían servir como base para futuras líneas de investigación. Por ejemplo, se observó que el procesador Intel presentaba un patrón de consumo escalonado descendente, lo cual no era esperado y esto podría ser relevante en una investigación. Las pruebas realizadas tuvieron una duración predefinida, sin poder medir el tiempo que tardaba cada procesador en completar la tarea. Esto se debió a que la API del programa utilizado para el testbench, Cinebench, no es igual entre los sistemas operativos Windows e iOS. Como resultado, no fue posible capturar la señal de finalización de la tarea en iOS. En futuros experimentos, sería interesante asegurarse de que la API del testbench funcione correctamente en ambos sistemas operativos, considerar la realización de las pruebas desde un equipo externo y añadir distintos tipos de testbench.

# Capítulo 9

## Summary and Conclusions

The development of this thesis, a wide range of branches within computer science was covered. The most challenging part was the implementation and development of the hardware device. The different equipment connections, the availability of materials and limited documentation, such as Apple's MagSafe, slowed down the progress of the project. Nevertheless, the results obtained from the tests conducted with the M1 processor were highly relevant. When comparing them with the samples obtained from the Intel processor, the efficiency of the M1 processor's performance was confirmed.

New questions arose that could serve as a base for future lines of research. It was observed that the Intel processor exhibited a descending staggered consumption pattern, which was unexpected and could be relevant. The tests had a predefined duration, without the ability to measure the time it took for each processor to complete the task. This limitation was due to the difference in the Cinebench testbench program's API between Windows and iOS operating systems. As a result, it wasn't possible to capture the task completion signal in iOS. In future experiments, it would be interesting to ensure that the testbench API functions properly on both operating systems, consider conducting the tests from an external device, and incorporate different types of testbench.

# Capítulo 10

## Presupuesto

A continuación se muestra en la tabla 10.1un desglose aproximado del costo total del desarrollo e implementación del dispositivo hardware y software:

| Ítem | Cantidad | Precio | Total (€) |
|---|---|---|---|
| Placa Arduino | 2 | 2.65 € | 5.30 € |
| Sensor CS712-30A | 2 | 5.15 € | 10.30 € |
| Sensor FZ0430 | 1 | 4.80 € | 4.80 € |
| Cargador Universal | 1 | 40.00 € | 40.00 € |
| Alargador USB TypeC 3.2 | 1 | 7.80 € | 7.80 € |
| Mano de obra | 60h | 30€/h | 1800 € |
| **Total** | | | 1868.20 € |

Tabla 10.1: Tabla de presupuesto

# Apéndice A

# Arduino

## A.1. Código Arduino V1

```
/*********************************************************************
* Fichero .ino
*********************************************************************
* AUTOR: Ángel Julián Bolaño Campos
* FECHA: 21/1/2023
* DESCRIPCION: Codigo de placa arduino para lectura de valores
*********************************************************************/

int current_sensor = 5;
float sensibility = 0.066; // Sensibility  for sensor of 30A

void setup() {
    Serial.begin(115200);
}

void loop() {
    float voltage_readed = analogRead(current_sensor) * (5.0 / 1023.0);
// AnalogRead
    float current = (voltage_readed - 2.5) / sensibility; // Function for obtaining
    Serial.print("Current: ");
    Serial.println(current,3);
    delay(30);
}
```

Listado A.1: Obtención de lecturas del sensor ACS712-30A

## A.2. Código Arduino V2

```
/*********************************************************************
* Fichero .ino
*********************************************************************
```

```
4 * AUTOR: Angel Julian Bolaño Campos
5 * FECHA: 21/1/2023
6 * DESCRIPCION: Codigo de placa arduino para lectura de valores
7 ********************************************************************/
8
9 int current_sensor = 5;
10 int voltage_sensor = 4;
11
12 float sensibility = 0.066; // Sensibility  for sensor of 30A
13
14 void setup() {
15     Serial.begin(115200);
16 }
17
18 void loop() {
19     float voltage = (float) 25 * analogRead(voltaje_sensor)/1023;  // AnalogRead Volt
20     float voltage_readed = analogRead(current_sensor) * (5.0 / 1023.0);
   // AnalogRead Current
21     float current = (voltage_readed - 2.5) / sensibility; // Function for obtaining
22     Serial.print("Current: ");
23     Serial.println(current,3);
24     Serial.print("Voltage: ");
25     Serial.println(voltage,2);
26     delay(30);
27 }
```

Listado A.2: Obtención de lecturas de ambos sensores

## A.3. Código Arduino V3

```
1 /*********************************************************************
2 * Fichero .ino
3 *********************************************************************
4 * AUTOR: Angel Julian Bolaño Campos
5 * FECHA: 21/1/2023
6 * DESCRIPCION: Codigo de placa arduino para lectura de valores
7 ********************************************************************/
8
9 int sInt = 5;
10 int sVol = 4;
11
12 void setup() {
13   Serial.begin(115200);
14 }
15
16 void loop() {
```

```
17   delay(30);
18   Serial.println(analogRead(sInt));
19   Serial.println(analogRead(sVol));
20 }
```

Listado A.3: Obtención de lecturas de ambos sensores simplificada

# Apéndice B

# Python

## B.1. Código Python V4

```
/*************************************************************************
* Fichero main.py
*************************************************************************
* AUTOR: Ángel Julián Bolaño Campos
* FECHA: 21/1/2023
* DESCRIPCION: Codigo de placa arduino para lectura de valores
*************************************************************************/

import serial
import argparse
import os
import re
import subprocess
import time
from time import ctime
import datetime
import psutil
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


def makeGraphic(df,path):

    plt.figure(figsize=(20, 16))
    plt.plot(df['Tiempo'], df['Intensidad'])
    plt.xlabel('Tiempo')
    plt.ylabel('Intensidad')
    plt.title('Consumo energético')

    x_values = df['Tiempo']
    step = len(x_values) // 30
```

```
33      ticks = x_values[::step]
34      plt.xticks(ticks, rotation='vertical')
35      whitout_ext = path.split('.')[0]
36      png_name = f'{whitout_ext}.png'
37      plt.savefig(png_name, dpi=300, bbox_inches='tight')
38
39
40  def launchBench():
41      if (multi_core == 'FALSE'):
42              arg = 'g_CinebenchCpu1Test=true'
43      else:
44              arg = 'g_CinebenchCpuXTest=true'
45      if (device == 'INTEL'):
46          cinebench_path = r'D:/Downloads/CinebenchR23/Cinebench.exe'
47          bench = subprocess.Popen([cinebench_path, arg],  stderr=subprocess.PIPE)
48      else:
49          bench = subprocess.Popen(['open', '/Applications/Cinebench.app',
    '--args', arg])
50
51      return bench
52
53
54
55  def closeBench():
56      if (device == 'INTEL'):
57          program = 'Cinebench.exe'
58      else:
59          program = "Cinebench"
60
61      for proc in psutil.process_iter(['name']):
62          if proc.info['name'] == program:
63              # Force close program
64              proc.kill()
65
66
67  def applyFilter(df, nfilter):
68
69      avg_volage, avg_current , value = 0, 0, 0
70      for i in range( len(df["Intensidad"]) ):
71
72          if ( value < nfilter ):
73              avg_current += df["Intensidad"][i]
74              avg_volage += df["Voltaje"][i]
75              value += 1
76          else:
77              writeInFile( file_filter,"{:.3f}".format(avg_current/nfilter), "{:.3f}".f
78              avg_volage, avg_current , value = 0, 0, 0
```

```
79
80
81 def writeInFile( file_temp ,current, voltage, time_stamp):
82     file_temp.write(str(current) + '\t')
83     file_temp.write(str(voltage) + '\t')
84     file_temp.write(str(time_stamp) + '\n')
85
86
87
88 def getTime( opt='D' ):
89     date_time = datetime.datetime.now()
90     if (opt == 'T'):
91         return date_time.strftime('%H:%M:%S.%f')
92     else:
93         return date_time.strftime('%d-%H_%M')
94
95 def closeFiles():
96     try:
97         file_start.close()
98         file_testbench.close()
99         file_end.close()
100    except Exception as e:
101        print(f"Error closing files: {e} \u2717")
102        raise
103
104
105
106 def getMetrics( file_temp, time_of_test ):
107    moment = f"Metrics in: {time_of_test} sec"
108    saveInFile(moment)
109    print(moment)
110    elapsed_time, samples, samples_ps = 0, 0, 0
111    init_time = time.time()
112    while ( elapsed_time <= time_of_test ):
113
114        arduinoSerial.flush() # clean serial
115        arduinoSerial.flush() # clean serial
116        # Read VOLTAGE CURRENT SENSOR
117        vol_current = arduinoSerial.readline().decode('iso-8859-1').rstrip()
118        # Read VOLTAGE VOLTAGE SENSOR
119        vol_voltage = arduinoSerial.readline().decode('iso-8859-1').rstrip()
120        # C
121        current_sensor = int(re.sub('[^0-9]', '', vol_current)) * (5 / 1023)
122        adjust_intensity = "{:.3f}".format( 0.180 + ( (current_sensor - 2.5) / sensit
   )
123
124        voltage_sensor = "{:.3f}".format( int(re.sub('[^0-9]', '', vol_voltage)) * (2
```

```
125
126         writeInFile(file_temp, adjust_intensity, voltage_sensor, getTime('T'))
127
128         samples += 1
129         final_time = time.time()
130         elapsed_time = final_time - init_time
131
132     samples_ps = samples/elapsed_time
133
134     stamp = (
135         f"\n\tElapsed time: {elapsed_time:.0f} sec\n"
136         f"\tSamples obtained per second: {samples_ps:.0f}\n"
137         f"\tSamples obtained: {samples:.0f}\n"
138     )
139     print(stamp)
140     saveInFile(stamp)
141
142 def getMetricsWait( file_temp, bench ):
143
144     moment = f"Metrics waiting"
145     saveInFile(moment)
146     print(moment)
147     elapsed_time, samples, samples_ps = 0, 0, 0
148     init_time = time.time()
149     while ( bench.poll() is None ):
150
151         arduinoSerial.flush() # clean serial
152         arduinoSerial.flush() # clean serial
153         # Read VOLTAGE CURRENT SENSOR
154         vol_current = arduinoSerial.readline().decode('iso-8859-1').rstrip()
155         # Read VOLTAGE VOLTAGE SENSOR
156         vol_voltage = arduinoSerial.readline().decode('iso-8859-1').rstrip()
157         # C
158         current_sensor = int(re.sub('[^0-9]', '', vol_current)) * (5 / 1023)
159         adjust_intensity = "{:.3f}".format( (current_sensor - 2.5) / sensitivity
    )
160
161         voltage_sensor = "{:.3f}".format( int(re.sub('[^0-9]', '', vol_voltage)) * (2
162
163         writeInFile(file_temp, adjust_intensity, voltage_sensor, getTime('T'))
164
165         samples += 1
166         final_time = time.time()
167         elapsed_time = final_time - init_time
168
169     samples_ps = samples/elapsed_time
170
```

```
171     stamp = (
172         f"\n\tElapsed time: {elapsed_time:.0f} sec\n"
173         f"\tSamples obtained per second: {samples_ps:.0f}\n"
174         f"\tSamples obtained: {samples:.0f}\n"
175     )
176     print(stamp)
177     saveInFile(stamp)
178 
179 
180 def selectPortMAC():
181     response = subprocess.run(['ls', '/dev/tty.*'], capture_output=True, text=True)
182     print(response.stdout)
183     print("Output:", response.returncode)
184 
185     print('Example: /dev/tty.usbserial-XXXXXX')
186     com_port = input()
187 
188 def saveInFile( input ):
189     file_info.write( f"\n{'-'* 80}\n")
190     file_info.write(input + '\t')
191 
192 
193 
194 
195 # Default values
196 time_test = 30
197 control_time = 10
198 com_port = 'COM4'
199 bps = 115200
200 device = 'INTEL'
201 filter_samples = 10
202 graphs = 'TRUE'
203 multi_core = 'FALSE'
204 wait_to_finish = 'no'
205 # Getting arguments
206 parser = argparse.ArgumentParser()
207 parser.add_argument("-d", "--device",          help="Select device [ INTEL | M1 ] [
208 parser.add_argument("-t", "--time_test",       help="Testing time in seconds [ defa
209 parser.add_argument("-c", "--control_time",    help="Control time in seconds [ defa
210 parser.add_argument("-p", "--com_port",        help="Serial COM for Arduino Board [
211 parser.add_argument("-b", "--bits_per_sec",    help="Serial COM bits per second [ d
212 parser.add_argument("-f", "--filter_samples",  help="Number of samples to obtain ari
213 parser.add_argument("-g", "--graphs",          help="Create graphs[ TRUE | FALSE ]
214 parser.add_argument("-m", "--multi_core",      help="Multicore [ TRUE | FALSE ] [ d
215 parser.add_argument("-w", "--wait_to_finish",  help="Wait normal launch [ yes | no
216 
217 
```

```
218
219 args = parser.parse_args()
220 # Global variables
221 device          = args.device if args.device else device
222 time_test       = args.time_test if args.time_test else time_test
223 control_time    = args.control_time if args.control_time else control_time
224 com_port        = args.com_port if args.com_port else com_port
225 bps             = args.bits_per_sec if args.bits_per_sec else bps
226 filter_samples  = args.filter_samples if args.filter_samples else filter_samples
227 graphs          = args.graphs if args.graphs else graphs
228 multi_core      = args.multi_core if args.multi_core else multi_core
229 wait_to_finish  = args.wait_to_finish if args.wait_to_finish else wait_to_finish
230
231 try:
232     if (multi_core == "FALSE"):
233         nameGlobal = getTime()
234     else:
235         nameGlobal = f"{getTime()}_m"
236
237     if not os.path.exists(device): os.mkdir(device)
238
239     #Paths and create folder
240     if ( device == 'INTEL'):
241         os.mkdir(f"{device}\\{nameGlobal}")
242         path = f"{os.getcwd()}\{device}\{nameGlobal}\{nameGlobal}"
243     else:
244         os.mkdir(f"{device}/{nameGlobal}")
245         path = f"{os.getcwd()}/{device}/{nameGlobal}/{nameGlobal}"
246
247
248     path_testbench  = f"{path}_testbench.csv"
249     path_start      = f"{path}_start.csv"
250     path_end        = f"{path}_end.csv"
251     path_global     = f"{path}_global.csv"
252     path_filter     = f"{path}_filter.csv"
253     path_info       = f"{path}_info.txt"
254
255     info = (
256         f"\tParameters used:\n"
257         f"\t{'-'* 16}\n"
258         f"\tDevice: {device}\n"
259         f"\tMulti-Core: {multi_core}\n"
260         f"\tWait until end testbench: {wait_to_finish}\n"
261         f"\tTime testing(sec): {time_test}\n"
262         f"\tTime control(wait in start/end): {control_time}\n"
263         f"\tPort COM Arduino Board: {com_port}\n"
264         f"\tBits per second of COM: {bps}\n"
```

```
265         f"\tNumber of samples to filter: {filter_samples}\n"
266         f"\tCreate graphs: {graphs}\n"
267     )
268     print(info)
269     print("Openning files.")
270     #Open files
271     try:
272         file_testbench  = open(path_testbench, 'w')
273         file_start      = open(path_start, 'w')
274         file_end        = open(path_end, 'w')
275         file_filter     = open(path_filter, 'w')
276         file_info       = open(path_info, 'w')
277         print("\tFiles opened. \u2713")
278     except Exception as e:
279         print(f"Error opening files: {e} \u2717")
280         raise
281
282
283
284     saveInFile(info)
285
286     #Setting varialbes
287     columns=['Intensidad', 'Voltaje', 'Tiempo']
288
289     try:
290         arduinoSerial       = serial.Serial(com_port , bps)
291     except Exception as e:
292         print(f"Error opening serial connection with Arduino: {e} \u2717")
293         raise
294
295     sensitivity         = 0.068 if device == 'INTEL' else 0.165
296     adjust_intensity    = 0.0
297     voltage_sensor      = 0.0
298
299
300     try:
301         getMetrics(file_start, int(control_time))
302     except Exception as e:
303         print(f"Error during initial sampling: {e} \u2717")
304         raise
305
306
307     try:
308         cine = launchBench()
309         print('POL= ', cine.poll() )
310     except Exception as e:
311         print(f"Error during launch testbench: {e} \u2717")
```

```
312         raise
313
314
315     try:
316         if (wait_to_finish == 'no'):
317             getMetrics(file_testbench, int(time_test))
318         else:
319             getMetricsWait(file_testbench, cine)
320     except Exception as e:
321         print(f"Error during test sampling: {e} \u2717")
322         raise
323
324
325     try:
326         closeBench()
327     except Exception as e:
328         print(f"Error closing testbench: {e} \u2717")
329         raise
330
331
332     try:
333         getMetrics(file_end, int(control_time))
334     except Exception as e:
335         print(f"Error during final sampling: {e} \u2717")
336         raise
337
338
339     closeFiles()
340
341     #Open panda
342     print("Read files CSV in panda")
343     try:
344         df_start        = pd.read_csv(path_start,  sep='\t', header=None, names=colu
345         df_testbench    = pd.read_csv(path_testbench,  sep='\t', header=None, names=
346         df_end          = pd.read_csv(path_end,  sep='\t', header=None, names=columns
347         print("\tFiles read successfully. \u2713")
348     except Exception as e:
349         print(f"Error reading CSV files: {e} \u2717")
350         raise
351
352
353     #Concatenated in global file
354     print("Joining files")
355     try:
356         global_test = pd.concat([ df_start, df_testbench, df_end], ignore_index=True
357         global_test.to_csv(path_global, index=False, sep='\t', float_format='%.3f', 
358         print("\tFiles concatenated. \u2713")
```

```
359     except Exception as e:
360         print(f"File concatenation error: {e}  \u2717")
361         raise
362 
363 
364     try:
365         global_testbench = pd.read_csv(path_global,  sep='\t', header=None, names=col
366     except Exception as e:
367         print(f"Error reading CSV files: {e} \u2717")
368         raise
369 
370 
371     print("Applying filter")
372     try:
373         applyFilter(global_test, filter_samples)
374         file_filter.close()
375         df_filter = pd.read_csv(path_filter,  sep='\t', header=None, names=columns)
376         print("\tFilter applied. \u2713")
377     except Exception as e:
378         print(f"File concatenation error: {e}  \u2717")
379         raise
380 
381 
382     if (graphs == 'TRUE'):
383         print("Creating graphs. This may take a few minutes...")
384         try:
385             makeGraphic(global_testbench,path_global)
386             print("\tGraph global completed. \u2713")
387             makeGraphic(df_filter,path_filter)
388             print("\tGraph with filter completed. \u2713")
389         except Exception as e:
390             print(f"Error making graphs: {e} \u2717")
391             raise
392 
393 
394     try:
395         arduinoSerial.close()
396     except Exception as e:
397         print(f"Error closing Arduino serial : {e} \u2717")
398         raise
399 
400     file_info.close()
401     file_filter.close()
402     print("\nProcess completed successfully!!!\n")
403 except:
404     if args.help is not None:
405         closeFiles()
```

```
406         file_info.close()
407         file_filter.close()
408         dir_rm = f"{device}/{nameGlobal}"
409         if (device == 'INTEL'):
410             subprocess.run(["powershell.exe", f'Remove-Item "{device}/{nameGlobal}"
411         else:
412             os.system(f"rm {device}/{nameGlobal}")
413 
414     print(f"Ejecution stoped \u2717")
```

Listado B.1: Programa para obtención de valores

## B.2. Código Python gráfica de comparación Intel vs Mac

```
1  /*************************************************************************
2  * Fichero main.py
3  *************************************************************************
4  * AUTOR: Ángel Julián Bolaño Campos
5  * FECHA: 12/07/2023
6  * DESCRIPCION: Código crear gráfica de comparación Intel vs Mac
7  *************************************************************************/
8 
9  import pandas as pd
10 import matplotlib.pyplot as plt
11 import scipy.stats as stats
12 
13 
14 # Fichero MAC
15 datos_equipo_1 = pd.read_csv('global_mac.csv', names=['Amperios', 'Voltage', 'Tiempo
16 
17 # Fichero INTEL
18 datos_equipo_2 = pd.read_csv('global_intel.csv', names=['Amperios', 'Voltage', 'Tiemp
19 
20 # Conversión a valores numéricos
21 datos_equipo_1['Amperios'] = pd.to_numeric(datos_equipo_1['Amperios'])
22 datos_equipo_1['Voltage'] = pd.to_numeric(datos_equipo_1['Voltage'])
23 datos_equipo_2['Amperios'] = pd.to_numeric(datos_equipo_2['Amperios'])
24 datos_equipo_2['Voltage'] = pd.to_numeric(datos_equipo_2['Voltage'])
25 
26 # Calculo de vatios
27 datos_equipo_1['Watts'] = datos_equipo_1['Amperios'] * datos_equipo_1['Voltage']
28 datos_equipo_2['Watts'] = datos_equipo_2['Amperios'] * datos_equipo_2['Voltage']
29 
30 
31 # filtro
```

```
datos_filtrados_1 = datos_equipo_1 #datos_equipo_1.iloc[::0]
datos_filtrados_2 = datos_equipo_2 #datos_equipo_2.iloc[::0]


datos_equipo_1['Tiempo'] = pd.to_datetime(datos_equipo_1['Tiempo'], format='%H:%M:%S.
datos_equipo_1['Tiempo'] = (datos_equipo_1['Tiempo'] - datos_equipo_1['Tiempo'].iloc

datos_equipo_2['Tiempo'] = pd.to_datetime(datos_equipo_2['Tiempo'], format='%H:%M:%S.
datos_equipo_2['Tiempo'] = (datos_equipo_2['Tiempo'] - datos_equipo_2['Tiempo'].iloc


fig, ax = plt.subplots()
x_values = datos_filtrados_1['Tiempo']
step = len(x_values) // 70
ticks = x_values[::step]

ax.plot(datos_filtrados_1['Tiempo'], datos_filtrados_1['Watts'], 'b-', label='M1')
ax.plot(datos_filtrados_1['Tiempo'], datos_filtrados_2['Watts'], 'r-', label='Intel')

ax.set_xlabel('Tiempo transcurrido (segundos)')
ax.set_ylabel('Vatios')
ax.set_title('Comparación de Vatios entre M1 e Intel')
plt.xticks(ticks, rotation='vertical')
plt.legend(loc='upper right')
plt.show()
```

Listado B.2: Programa para obtención de valores

## B.3. Código Python para comparación entre diferentes muestras

```
/*****************************************************************
* Fichero main.py
*****************************************************************
* AUTOR: Ángel Julián Bolaño Campos
* FECHA: 12/07/2023
* DESCRIPCION: Código para comparación entre diferentes muestras
*****************************************************************/

import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import numpy as np

# Fichero global_1
datos_equipo_1 = pd.read_csv('global_1.csv', names=['Amperios', 'Voltaje', 'Tiempo']
```

```
16
17 # Fichero global_2
18 datos_equipo_2 = pd.read_csv('global_2.csv', names=['Amperios', 'Voltaje', 'Tiempo']
19
20 # Fichero global_3
21 datos_equipo_3 = pd.read_csv('global_3.csv', names=['Amperios', 'Voltaje', 'Tiempo']
22
23
24 # Conversión a valores numéricos
25 datos_equipo_1['Amperios'] = pd.to_numeric(datos_equipo_1['Amperios'])
26 datos_equipo_1['Voltaje'] = pd.to_numeric(datos_equipo_1['Voltaje'])
27 datos_equipo_2['Amperios'] = pd.to_numeric(datos_equipo_2['Amperios'])
28 datos_equipo_2['Voltaje'] = pd.to_numeric(datos_equipo_2['Voltaje'])
29 datos_equipo_3['Amperios'] = pd.to_numeric(datos_equipo_3['Amperios'])
30 datos_equipo_3['Voltaje'] = pd.to_numeric(datos_equipo_3['Voltaje'])
31
32
33
34
35 # # Filtro
36 datos_filtrados_1 = datos_equipo_1 #datos_equipo_1.iloc[::50]
37 datos_filtrados_2 = datos_equipo_2 #datos_equipo_2.iloc[::50]
38 datos_filtrados_3 = datos_equipo_3 #datos_equipo_3.iloc[::50]
39
40 # Convertir el tiempo en segundos transcurridos desde el inicio
41 datos_equipo_1['Tiempo'] = pd.to_datetime(datos_equipo_1['Tiempo'], format='%H:%M:%S.
42 datos_equipo_1['Tiempo'] = (datos_equipo_1['Tiempo'] - datos_equipo_1['Tiempo'].iloc
43
44 datos_equipo_2['Tiempo'] = pd.to_datetime(datos_equipo_2['Tiempo'], format='%H:%M:%S.
45 datos_equipo_2['Tiempo'] = (datos_equipo_2['Tiempo'] - datos_equipo_2['Tiempo'].iloc
46
47 datos_equipo_3['Tiempo'] = pd.to_datetime(datos_equipo_3['Tiempo'], format='%H:%M:%S.
48 datos_equipo_3['Tiempo'] = (datos_equipo_3['Tiempo'] - datos_equipo_3['Tiempo'].iloc
49
50
51
52
53 fig, ax = plt.subplots()
54 x_values = datos_filtrados_1['Tiempo']
55 step = len(x_values) // 50
56 ticks = x_values[::step]
57
58 ax.plot(datos_filtrados_1['Tiempo'], datos_filtrados_1['Amperios'], 'b-', label='Mue
59 ax.plot(datos_filtrados_2['Tiempo'], datos_filtrados_2['Amperios'], 'r-', label='Mue
60 ax.plot(datos_filtrados_3['Tiempo'], datos_filtrados_3['Amperios'], 'g-', label='Mue
61
62 ax.axvspan(10, 100, alpha=0.3, color='gray')
```

```
63
64 # líneas verticales en los 10 y 100 segundos
65 ax.axvline(x=10, color='k', linestyle='--')
66 ax.axvline(x=98, color='k', linestyle='--')
67
68 ax.set_xlabel('Tiempo transcurrido (segundos)')
69 ax.set_ylabel('Amperios')
70 ax.set_title('Comparación de Amperios entre Equipos')
71 plt.xticks(ticks, rotation='vertical')
72 plt.legend(loc='upper right')
73 plt.show()
```

Listado B.3: Programa para obtención de valores

## B.4. Código Python para comparación entre diferentes muestras

```
1  /**************************************************************************
2  * Fichero main.py
3  **************************************************************************
4  * AUTOR: Ángel Julián Bolaño Campos
5  * FECHA: 12/07/2023
6  * DESCRIPCION: Calculo de valores
7  **************************************************************************/
8
9  dir = # {inicio de nombre fichero}
10 nombres_equipos = ['start', 'testbench', 'end', 'global']
11
12 datos_files = {}
13
14 for i, nombre_equipo in enumerate(nombres_equipos):
15     archivo = f'{dir}{nombre_equipo}.csv'
16     datos = pd.read_csv(archivo, names=['Amperios', 'Voltage', 'Tiempo'], sep='\s+')
17     datos['Amperios'] = pd.to_numeric(datos['Amperios'])
18     datos['Voltage'] = pd.to_numeric(datos['Voltage'])
19     datos['Watts'] = datos['Amperios'] * datos['Voltage']
20     datos['Tiempo'] = pd.to_datetime(datos['Tiempo'], format='%H:%M:%S.%f')
21     datos['Tiempo'] = (datos['Tiempo'] - datos['Tiempo'].iloc[0]).dt.total_seconds()
22     tiempo_total = datos['Tiempo'].iloc[-1]
23     datos_files[nombre_equipo] = {'datos': datos, 'tiempo_total': tiempo_total}
24
25
26 ic_equipos = {}
27 media_aritmetica = {}
28
29 for nombre_equipo, datos_equipo in datos_files.items():
```

```
30      ic_equipos[nombre_equipo] = {
31          'Voltaje': [round(ic, 3) for ic in stats.norm.interval(0.95, loc=np.mean(dato
32          'Amperaje': [round(ic, 3) for ic in stats.norm.interval(0.95, loc=np.mean(dat
33          'Vatios': [round(ic, 3) for ic in stats.norm.interval(0.95, loc=np.mean(datos
34      }
35      media_aritmetica[nombre_equipo] = {
36          'Voltaje': round(np.mean(datos_equipo['datos']['Voltage']), 3),
37          'Amperaje': round(np.mean(datos_equipo['datos']['Amperios']), 3),
38          'Vatios': round(np.mean(datos_equipo['datos']['Watts']), 3)
39      }
40      tiempo_total = datos_equipo['tiempo_total']
41      print(f"Tiempo total para {nombre_equipo}: {round(tiempo_total, 3)} segundos")
42
43  # intervalos de confianza y media aritmética
44
45  for nombre_equipo, ic_medidas in ic_equipos.items():
46      print(f"Intervalo de confianza (95%) para {nombre_equipo}:")
47      for medida, ic in ic_medidas.items():
48          print(f"{medida}: {ic}")
49      print()
50      print(f"Media aritmética para {nombre_equipo}:")
51      for medida, valor in media_aritmetica[nombre_equipo].items():
52          print(f"{medida}: {valor}")
53      print()
```

Listado B.4: Programa para obtención de valores

# Apéndice C

## Otras muestras

Tabla C.1: Tabla de Medias M1 (Muestra 2)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.35s) | Voltaje | (19.341, 19.342) | 19.342 V |
| | Amperaje | (0.252, 0.254) | 0.253 A |
| | Vatios | (4.883, 4.931) | 4.907 W |
| Testbench (90s) | Voltaje | (19.033, 19.035) | 19.034 V |
| | Amperaje | (0.761, 0.764) | 0.762 A |
| | Vatios | (14.486, 14.535) | 14.510 W |
| Final (10s) | Voltaje | (19.334, 19.336) | 19.335 V |
| | Amperaje | (0.260, 0.264) | 0.262 A |
| | Vatios | (5.039, 5.107) | 5.073 W |
| Global (108.35s) | Voltaje | (19.084, 19.087) | 19.085 V |
| | Amperaje | (0.674, 0.679) | 0.677 A |
| | Vatios | (12.852, 12.942) | 12.897 W |

Tabla C.2: Tabla de Medias M1 (Muestra 3)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.35s) | Voltaje | (19.356, 19.357) | 19.357 V |
| | Amperaje | (0.227, 0.230) | 0.228 A |
| | Vatios | (4.403, 4.455) | 4.429 W |
| Testbench (90s) | Voltaje | (19.040, 19.041) | 19.041 V |
| | Amperaje | (0.750, 0.753) | 0.752 A |
| | Vatios | (14.292, 14.341) | 14.317 W |
| Final (10s) | Voltaje | (19.353, 19.356) | 19.355 V |
| | Amperaje | (0.227, 0.231) | 0.229 A |
| | Vatios | (4.397, 4.469) | 4.433 W |
| Global (108.35s) | Voltaje | (19.092, 19.095) | 19.094 V |
| | Amperaje | (0.660, 0.665) | 0.663 A |
| | Vatios | (12.593, 12.686) | 12.639 W |

Tabla C.3: Tabla de Medias Intel (Muestra 2)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.33s) | Voltaje | (19.851, 19.859) | 19.855 V |
| | Amperaje | (0.256, 0.280) | 0.268 A |
| | Vatios | (5.068, 5.520) | 5.294 W |
| Testbench (90s) | Voltaje | (19.161, 19.165) | 19.163 V |
| | Amperaje | (2.168, 2.183) | 2.175 A |
| | Vatios | (41.445, 41.713) | 41.579 W |
| Final (10s) | Voltaje | (19.847, 19.854) | 19.851 V |
| | Amperaje | (0.265, 0.284) | 0.274 A |
| | Vatios | (5.241, 5.606) | 5.424 W |
| Global (108.33s) | Voltaje | (19.278, 19.285) | 19.282 V |
| | Amperaje | (1.837, 1.858) | 1.847 A |
| | Vatios | (35.152, 35.534) | 35.343 W |

Tabla C.4: Tabla de Medias Intel (Muestra 3)

| Etapa - Duración(s) | Medida | IC 95 % | Media Aritmética |
|---|---|---|---|
| Inicio (8.33s) | Voltaje | (19.885, 19.889) | 19.887 V |
| | Amperaje | ([0.184, 0.198) | 0.191 A |
| | Vatios | (3.661, 3.934) | 3.798 W |
| Testbench (90s) | Voltaje | (19.225, 19.230) | 19.227 V |
| | Amperaje | ([2.010, 2.023) | 2.016 A |
| | Vatios | (38.555, 38.804) | 38.680 W |
| Final (10s) | Voltaje | (19.831, 19.840) | 19.835 V |
| | Amperaje | 0.418, 0.441) | 0.429 A |
| | Vatios | (8.256, 8.704) | 8.480 W |
| Global (108.33s) | Voltaje | (19.333, 19.339) | 19.336 V |
| | Amperaje | (1.716, 1.734) | 1.725 A |
| | Vatios | (32.953, 33.299) | 33.126 W |

# Bibliografía


[1] Karthik Iyer. Apple M1 series vs Intel Core i9-12900HK: Which laptop CPU is better?. Link.

[2] Kalpa D. Fernando. How apple took the “RISC” Link.

[3] Jon Hannibal Stokes. RISC vs. CISC: the Post-RISC Era Link.

[4] Datasheet. ACS712 Link.

[5] Sensor FZ0430 Link.

[6] Datasheet USB Type-C Link.