UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso para Alumnos con Práctica Profesional Supervisada

TÍTULO: Desarrollo e implementación de un medidor inteligente de energía eléctrica

AUTOR/A: Manzin, Matías Federico

DIRECTOR/A ACADÉMICO: Medina, Santiago

DIRECTOR/A PROFESIONAL: Navarria, Leonardo

CODIRECTOR/A ACADÉMICO: Libutti, Leandro

CARRERA: Licenciatura en Sistemas

RESUMEN

*En este trabajo se desarrolló un medidor de tensión, corriente y potencia activa (entre otros) en Corriente Alterna basándose en el circuito integrado **HLW8012**. Para ello, se realizó el prototipo del hardware del sistema, así como la implementación del firmware que envía los resultados alámbrica y/o inalámbricamente, en formatos JSON y/o MessagePack.*

Palabras Clave

Eficiencia Energética, Control de Consumo Energético, Servicios Web, Concurrencia, Programación Orientada a Objetos, Programación Dirigida por Eventos, Programación Orientada a Eventos, Integración de Proyectos, Sistemas Inteligentes, Internet de las Cosas (IoT), Monitorización en Tiempo Real, Microcontroladores.

Conclusiones

El diseño y la implementación del Medidor constituyó un proyecto que incluye desafíos diversos y constantes, a cambio de haber facilitado el proceso de medición. El desarrollo del Framework ha permitido minimizar la deuda técnica a través de un diseño modular y multiplataforma, que facilita las modificaciones al código y las migraciones hacia otros microcontroladores. Al medir el consumo de un algoritmo de multiplicación de matrices en distintos escenarios, se ha comprobado que los algoritmos paralelos pueden aumentar el consumo instantáneo (respecto a sus equivalentes secuenciales) pero también reducir el tiempo de ejecución, y en consecuencia, la cantidad de energía eléctrica total requerida.

Trabajos Realizados

Diseño y prototipado del hardware del sistema, incluyendo la fuente de alimentación y el circuito aislante basado en optoacopladores. Implementación del firmware del microcontrolador, basado en un framework de desarrollo propio. Calibración del medidor, comparando sus resultados con los de un multímetro de mayor precisión. Comparación de consumo de algoritmos de multiplicación de matrices.

Trabajos Futuros

Se propone a futuro un conjunto de posibles trabajos a desarrollar, con el objetivo de mejorar el sistema: Diseñar y fabricar el circuito impreso final, implementar un dashboard web que permita visualizar los resultados obtenidos en tiempo real y emitir alertas cuando los valores medidos no estén en un rango numérico determinado, usar modelos de regresión lineal para aumentar la exactitud, etcétera.

Fecha de la presentación: FEBRERO 2025

Facultad de Informática | UNIVERSIDAD NACIONAL DE LA PLATA

Índice General

1. Exposición de lo realizado en la PAE-PPS

El Instituto de Investigación en Informática LIDI [1] de la Facultad de Informática de la UNLP viene trabajando en el despliegue de un sistema inteligente encargado de recolectar información de las diferentes zonas de los edificios y tomar decisiones para generar un ahorro en el consumo energético [2] [3], a través de Unidades Locales de Procesamiento Inteligente (LIPUs - Local Intelligent Processing Unit), que permiten monitorear y controlar los ambientes [4] mediante un conjunto de sensores que detectan eventos y parámetros. Por ejemplo, la presencia de personas, la temperatura y humedad, entre otros. En la figura 1 se muestra el diagrama de bloques que conforman a una LIPU.

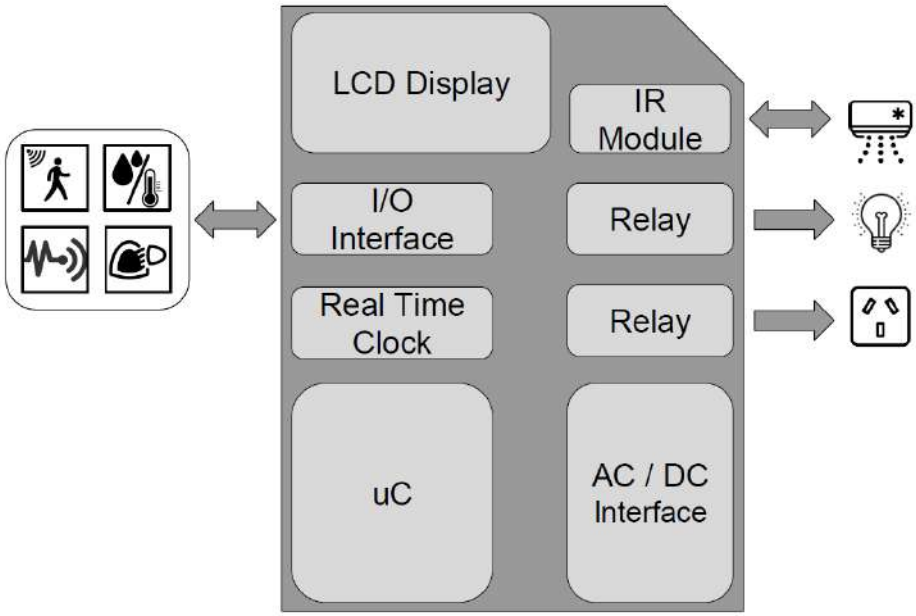

Fig. 1: Representación genérica de la estructura interna de los dispositivos que forman parte de las LIPUs.

Sin embargo, las LIPUs no cuentan con un dispositivo que permitiera medir el consumo eléctrico en cada zona. Por lo tanto, el objetivo de esta Práctica Profesional Supervisada fue desarrollar y probar un medidor de consumo energético en corriente alterna, capaz de recolectar muestras para ser analizadas, conectándose al sistema inteligente y/o enviándolas de manera

independiente a través de una conexión inalámbrica o alámbrica. De esta forma, permite integrarse fácilmente a otros sistemas o aplicaciones.

Para ello, se realizó una solución basada en Internet de las Cosas, incorporando un microcontrolador ESP8266 [5] junto a un módulo de medición basado en el circuito integrado HLW8012 [6]; conformando así un nodo sensor. Los nodos sensores se complementan con servicios como el almacenamiento de los datos y su visualización en paneles de control interactivos. Además, para maximizar la seguridad de los usuarios del medidor, se desarrolló un circuito basado en optoacopladores y elementos de protección adicionales. Se conectó entre el ESP8266 y el HLW8012, evitando que una falla en el segundo provoque daños en el primero y en la computadora que el usuario podría estar utilizando para acceder a los resultados de las mediciones. En la figura 2 se muestra el prototipo desarrollado del Medidor.

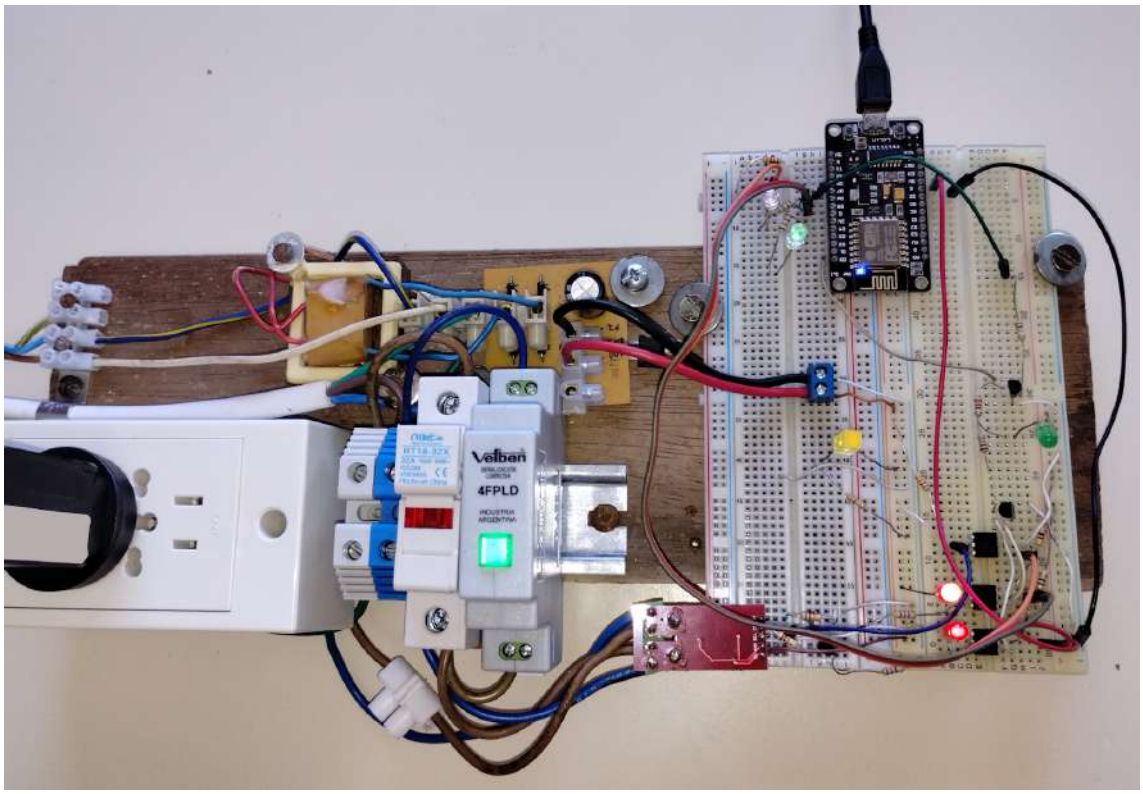

Fig. 2: Vista superior del prototipo realizado.

Por otra parte, su incorporación en el sistema inteligente permitirá analizar las métricas de energía en búsqueda de definir estrategias eficientes para el entorno controlado.

1.1. Introducción al proyecto

En el marco del Programa de Apoyo al Egreso con Práctica Profesional Supervisada (PAE-PPS), se llevó a cabo el desarrollo de un medidor inteligente avanzado de energía eléctrica (AMI). [7]

El sistema cumple con las características de un medidor inteligente modular [7]. Al medidor básico HLW8012 se le incorporaron diferentes módulos a través de sus pines de conexión. Cada uno de ellos cumple determinadas responsabilidades, indicadas a continuación:

- Módulo Medidor: Transmite los valores medidos a través de pulsos eléctricos no aislados de la red eléctrica.
- Módulo Aislante: Intermediario entre el Módulo Inteligente y el Módulo Medidor, que se encarga de transmitir señales eléctricas de uno al otro, sin acoplarlos eléctricamente entre sí.
- Módulo Inteligente: Ejecuta el *firmware* del sistema, recuperando los valores medidos a partir de los pulsos recibidos (de forma aislada, gracias al Módulo Aislante).
- Módulo de Potencia: alimenta de energía al Módulo Aislante y al Módulo Medidor y constituye el circuito eléctrico que permite alimentar al dispositivo del cual se quiere conocer el consumo energético.

La arquitectura modular permitió la división del desarrollo en fases, cada una centrada en un módulo específico, lo que resultó en un sistema escalable y eficiente.

2. Informe Técnico

2.1. Resumen

En los últimos años, el consumo energético en instituciones educativas ha crecido significativamente, lo que ha derivado en la necesidad de soluciones que permitan un uso eficiente de la energía. Este trabajo tiene como objetivo el diseño y desarrollo de un medidor de energía eléctrica para el monitoreo del consumo energético de cualquier dispositivo que se encuentre en un establecimiento educativo y la posible detección de anomalías.

El medidor permite obtener resultados en tiempo real, enviados de manera alámbrica y/o inalámbrica para lograr una integración con otros servicios y dispositivos existentes. Se han utilizado tecnologías modernas como un framework de desarrollo propio, garantizando una plataforma escalable, segura y eficiente.

2.2. Palabras claves

Eficiencia Energética, Control de Consumo Energético, Servicios Web, Concurrencia, Programación Orientada a Objetos, Programación Dirigida por Eventos, Programación Orientada a Eventos, Integración de Proyectos, Sistemas Inteligentes, Internet de las Cosas (IoT), Monitorización en Tiempo Real, Microcontroladores.

2.3. Marco Conceptual

2.3.1. Antecedentes

En los últimos años, en el Laboratorio de Investigación LIDI [1] de la Facultad de Informática de la UNLP, se trabaja en el despliegue de un sistema inteligente encargado de recolectar información de las diferentes zonas de los edificios y tomar decisiones para generar un ahorro en el consumo energético [2] [3]. Este sistema está constituido fundamentalmente por Unidades Locales de

Procesamiento Inteligente (LIPUs - Local Intelligent Processing Unit), que permiten monitorear y controlar los ambientes [4]. Las LIPUs se encuentran conectadas a través de una red inalámbrica, que permite el intercambio de información entre las distintas zonas controladas del edificio. Cada LIPU cuenta con un conjunto de sensores que detectan eventos y parámetros del ambiente, como la presencia de personas, temperatura y humedad. En la figura 3 se muestra el prototipo existente de un LIPU.

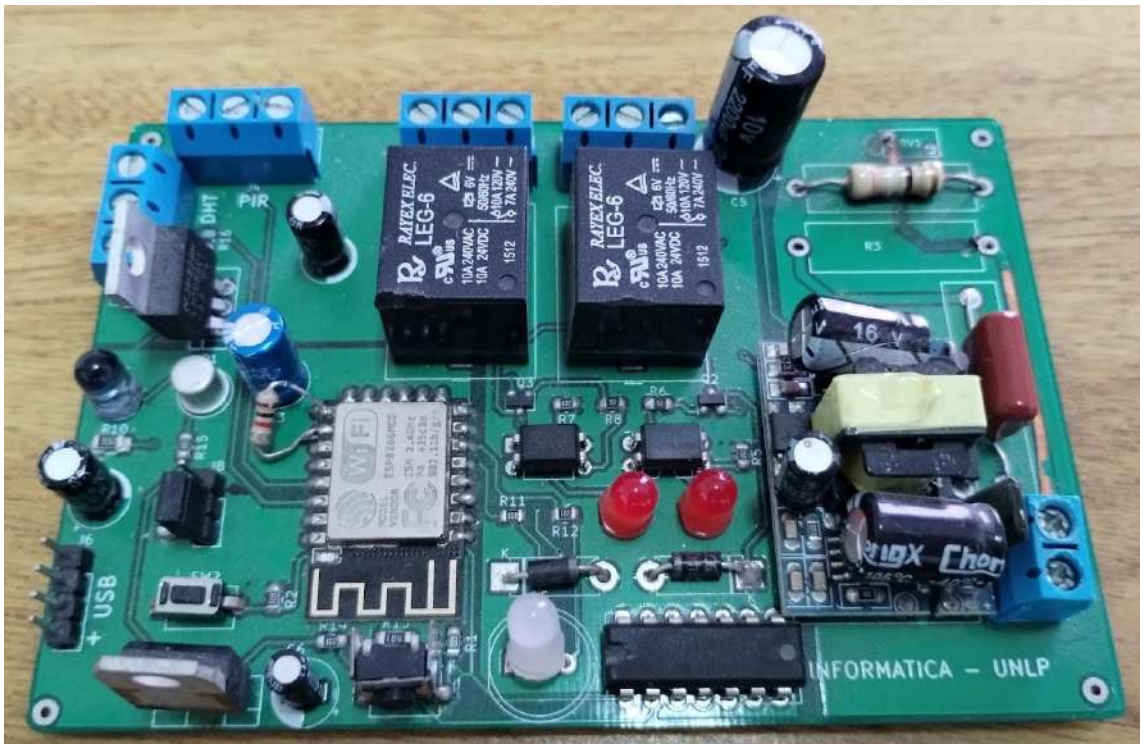

Fig. 3: Imagen de un nodo LIPU.

Sin embargo, los LIPUs no contaban con un dispositivo que permitiera medir el consumo eléctrico de un área controlada. Por lo tanto, el objetivo de esta Práctica Profesional Supervisada fue desarrollar y verificar el correcto funcionamiento de un medidor de consumo energético en corriente alterna. Entre las principales funcionalidades añadidas se encuentra la capacidad de recolectar muestras del ambiente controlado y de conectarse con el sistema inteligente para el envío de la información a través de una conexión inalámbrica

o alámbrica. Ésto permite integrar este desarrollo fácilmente a otros sistemas o aplicaciones.

2.3.2. Conceptos Eléctricos

El cálculo del consumo energético de los dispositivos electrónicos se realiza a partir del conocimiento de la potencia. A continuación se listan las diferentes variables físicas que debe calcular el medidor desarrollado:

- **Corriente (I):** Es la velocidad a la que un flujo de electrones pasa por un punto de un circuito eléctrico completo. Se mide en Amperes (A).
- **Tensión (U):** Es la presión de una fuente de energía de un circuito eléctrico que empuja los electrones cargados a través de un lazo conductor, generando una corriente eléctrica. Se mide en Volts (V).
- **Potencia Activa (P):** Es la potencia eléctrica real utilizada por un dispositivo para realizar un determinado trabajo. Se mide en Watts (W). También se la utiliza en los cálculos de las pérdidas en los conductores eléctricos.

El **valor eficaz** de una tensión o corriente alterna es el valor de una tensión o corriente continua que disipa la misma cantidad de potencia activa media en un resistor de valor no nulo.

$$U_{RMS} = \sqrt{\frac{1}{T} \cdot \int_{t_0}^{t_0+T} (u(t))^2 \cdot dt}, \text{ con } T > 0$$

Ecuación 1. Media cuadrática de la tensión.

$$I_{RMS} = \sqrt{\frac{1}{T} \cdot \int_{t_0}^{t_0+T} (i(t))^2 \cdot dt}, \text{con } T > 0$$

Ecuación 2. Media cuadrática de la corriente.

$$P_{RMS} = \sqrt{\frac{1}{T} \cdot \int_{t_0}^{t_0+T} (u(t) \cdot i(t)) \cdot dt}, \text{con } T > 0$$

Ecuación 3. Media cuadrática de la potencia.

Utilizando las ecuaciones 1, 2 y 3 se pueden obtener las diferentes variables necesarias para el cálculo del consumo energético. A continuación se explica cada una de ellas:

- **Potencia Aparente (S):** Es la potencia eléctrica total usada. Se mide en Volts-Amperes (VA). Se utiliza para realizar los cálculos de las instalaciones, como ser diámetros de conductores, transformadores, llaves térmicas, disyuntores, etc.

$$S_{RMS} = U_{RMS} \cdot I_{RMS} = \sqrt{P^2 + Q^2}$$

Ecuación 4. Potencia Aparente.

- **Potencia Reactiva Absoluta (Q_{abs}):** Es la potencia eléctrica usada por cargas inductivas y capacitivas para funcionar, pero no para realizar un trabajo real. Se mide en Volts-Amperes-reactivos (VAr). Está relacionada con el almacenamiento de la energía que se produce tanto en los inductores (en forma de campo magnético) y en los capacitores (en forma de campo eléctrico).

$$Q_{abs} = \sqrt{S^2 - P^2}$$

Ecuación 5. Potencia Reactiva Absoluta.

- **Energía Activa:** Es la potencia activa consumida, acumulada durante el período de una hora. Se mide en Watts-hora (Wh). Esta energía es la determinada por todos los medidores de energía[1], tanto de disco como electrónicos.

- **Factor de Potencia Absoluto:** Indica el aprovechamiento de la instalación, es decir, qué porcentaje de toda la energía que posee disponible la instalación eléctrica se convierte en trabajo útil.

$$fp_{abs} = \frac{P_{RMS}}{S_{RMS}}$$

Ecuación 6. Factor de Potencia Absoluto.

2.3.3. Monitoreo energético y su relevancia en la Universidad

El incremento en el consumo energético en los últimos años ha sido un tema de gran interés para las instituciones públicas y privadas. En particular, las universidades, que dependen de una gran cantidad de dispositivos electrónicos para sus actividades cotidianas, enfrentan grandes desafíos en términos de optimización del consumo de energía. Este uso intensivo de recursos tecnológicos no solo incrementa los costos operativos, sino que también tiene un impacto significativo en el medio ambiente debido a la generación de gases de efecto invernadero por parte de fuentes de energía no renovables.

[1] El costo de la energía está compuesto por un cargo fijo y un cargo variable, dichos cargos se pueden consultar en el organismo encargado de controlar las concesiones de empresas de energía. Para distribuidoras de energía nacionales, así como EDENOR y EDESUR, se consulta en la página del Ente Nacional Regulador de la Electricidad (ENRE) [8] y en el caso de distribuidoras regidas en el ámbito provincial, se deberá consultar en cada provincia. En el caso de la Provincia de Buenos Aires, se debería consultar al Organismo de Control de Energía Eléctrica de la Provincia de Buenos Aires (OCEBA) [9].

En Argentina, según la Secretaría de Energía de la Nación, el $56\ [\%]$ de la energía primaria consumida proviene de hidrocarburos[2], lo que subraya la urgencia de implementar soluciones para reducir la demanda energética, especialmente en instituciones públicas que operan con presupuestos limitados. El Decreto 140/2007 establece que las instituciones públicas deben reducir su consumo energético en al menos un $10\ [\%]$, lo cual evidencia la relevancia de abordar este problema desde un enfoque técnico y práctico.

La solución desarrollada en el marco del Programa de Apoyo al Egreso con Práctica Profesional Supervisa (PAE-PPS) permite monitorear el consumo energético de uno o más dispositivos, permitiendo la toma de decisiones en función de los valores analizados del ambiente controlado. Por ejemplo, si un aula de una Facultad se encuentra vacía pero presenta consumos elevados, puede deberse a que los dispositivos eléctricos de alto consumo en ese espacio están encendidos; por lo que el sistema tomaría la decisión de apagarlos de forma automática y segura.

2.3.4. Qué son los sistemas inteligentes de gestión energética

Un sistema inteligente de gestión energética se define como una solución capaz de monitorear, controlar y optimizar el uso de energía de manera autónoma o semiautónoma. Estos sistemas recopilan datos en tiempo real, analizan las condiciones del entorno y toman decisiones para regular el uso de energía, minimizando el consumo sin comprometer el confort o la funcionalidad de los espacios.

En el contexto de esta PAE-PPS, el sistema desarrollado previamente presenta Unidades Locales de Procesamiento Inteligente (LIPUs) para el monitoreo distribuido de los diferentes espacios mediante sensores que recogen datos

[2] La Compañía Administradora del Mercado Mayorista Eléctrico Sociedad Anónima (CAMMESA) tiene entre sus objetivos el despacho técnico del Sistema Argentino de Interconexión (SADI) de acuerdo a lo previsto por la Ley 24.065 y sus normas complementarias y reglamentarias. En el sitio web de esta compañía (https://cammesaweb.cammesa.com/operacion/#opdemrealyprev) se puede consultar en tiempo real la matriz energética de generación. Se entiende por energía no renovable aquellas que provienen de combustibles fósiles, como ser fuel-oil, diesel y gas. La Energía Hidráulica se la ha encuadrado como renovable (años atrás se la consideraba como NO renovable).

sobre variables como la temperatura, la presencia de personas y el estado de los dispositivos conectados (luces, aire acondicionado, proyectores, etc). Entonces, el medidor desarrollado se convierte en un sensor conectado alámbrica o inalámbricamente a un LIPU.

2.3.5. Internet de las Cosas (IoT), uno de los pilares.

El desarrollo de la Internet de las Cosas (IoT) ha facilitado la implementación de sistemas inteligentes en edificios al permitir la conexión de dispositivos distribuidos en diferentes ubicaciones, como sensores de temperatura, detectores de movimiento y actuadores. Estos dispositivos pueden comunicarse entre ellos y con servidores centrales, permitiendo un control dinámico del entorno en tiempo real.

Las LIPUs actúan como nodos distribuidos que forman parte de una red IoT más amplia, recopilando datos de sensores ubicados en diferentes espacios de la institución. La aplicación web desarrollada previamente permite visualizar y gestionar estos dispositivos, ofreciendo a los administradores la capacidad de:

- **Monitoreo en tiempo real:** La aplicación proporciona información continua sobre el estado de los dispositivos, lo que permite ajustar el consumo de energía de acuerdo con las condiciones presentes en el entorno.

- **Automatización:** El sistema puede actuar automáticamente sobre los dispositivos electrónicos (luces, ventiladores, aire acondicionado) para optimizar su uso sin intervención humana.

- **Escalabilidad:** La infraestructura IoT y la aplicación web están diseñadas para escalarse fácilmente, lo que permite agregar más dispositivos y ubicaciones sin realizar cambios importantes en el sistema.

2.4. Trabajo Realizado

En esta sección se va a realizar una explicación detallada de los módulos desarrollados, los cuales fueron descritos en los puntos anteriores.

2.4.1. Diseño del Sistema

Arquitectura del Hardware

Se optó por dividir el circuito electrónico en distintos módulos que se comunicarán a través de conectores, borneras y/o cables. De esta forma, se simplificó el diseño de cada subcircuito y de las modificaciones futuras, en las cuales sólamente sería necesario reemplazar el hardware utilizado.

A través del software KiCad [10], un editor de esquemáticos y PCBs de código abierto, se crearon los esquemáticos de los distintos módulos del Hardware con los niveles de abstracción que se muestran a continuación:

- **Alto nivel (máxima abstracción):** Diagrama de bloques que muestra cómo se interconectan los distintos módulos entre sí, ocultando los detalles de implementación de cada uno.
- **Medio nivel (opcional):** Esquemático que “instancia” a uno o más subcircuitos y los conecta a otros componentes.
- **Bajo nivel (mínima abstracción):** Esquemático que detalla cómo fue implementado un determinado módulo o subcircuito.

En el esquema de la figura 4 puede observarse el flujo de información entre los diferentes módulos del sistema. En primer lugar, una de las funciones del Módulo de Potencia es proporcionar energía al Módulo Medidor. Este módulo envía y recibe señales eléctricas del Módulo Aislante, el cual las “reenvía”, de forma segura y aislada, al Módulo Inteligente. Éste último las procesa y periódicamente envía los resultados de la medición a la computadora y/o al servidor externo.

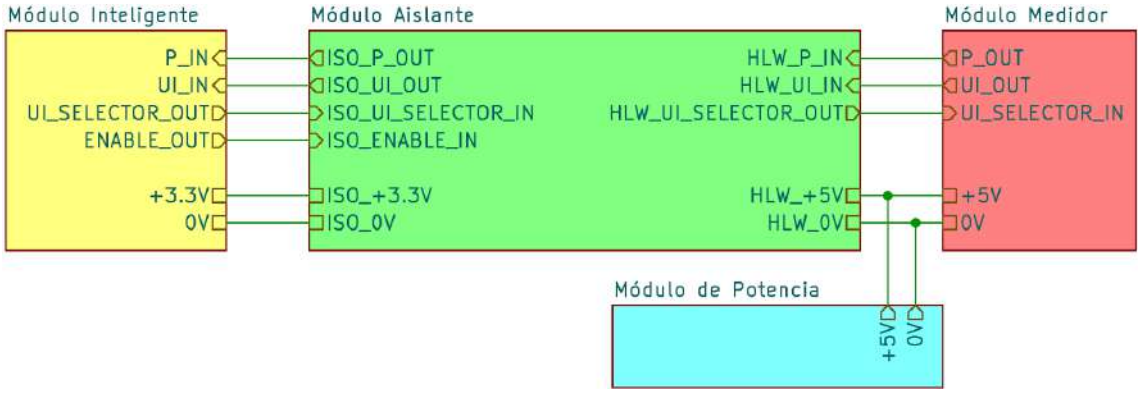

Fig. 4: Esquemático de alto nivel del sistema.

En la figura 5 se visualiza el prototipo de los distintos módulos desarrollados, donde cada uno se encuentra encerrado por un color de identificación. En rojo se observa el Módulo Medidor, en amarillo el Módulo Inteligente, en verde el Módulo Aislante, y en azul el de Potencia.

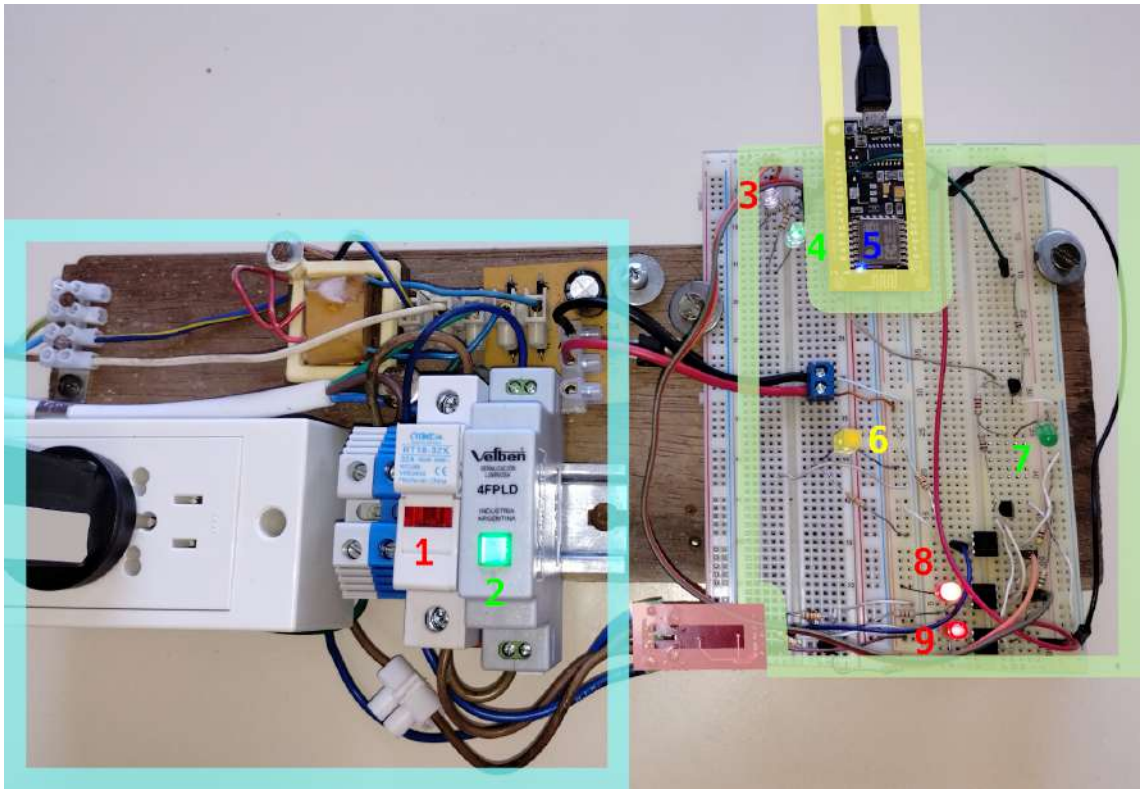

Fig. 5: Vista superior del prototipo desarrollado.

Por otra parte, el prototipo cuenta con diferentes indicadores lumínicos en sus módulos para el reconocimiento de diferentes situaciones. En la figura 5 también se observa la numeración de estos LEDs, y a continuación se describe el estado que representa cada uno:

1. El fusible alojado se quemó.
2. El tomacorriente está energizado.
3. El Módulo Inteligente está recibiendo pulsaciones aisladas de mediciones de potencia activa.
4. Ídem, respecto a mediciones de tensión o corriente.
5. Se estableció la conexión Wi-Fi a la red configurada.
6. El Módulo Medidor y el subcircuito izquierdo del Módulo Aislante están energizados.
7. El Módulo Inteligente solicita una medición de la tensión actual en lugar de la corriente.
8. El Módulo Medidor está emitiendo pulsaciones no aisladas de mediciones de potencia activa.
9. Ídem, respecto a mediciones de tensión o corriente.

A continuación, se explican las diferentes tecnologías utilizadas en los módulos desarrollados.

Comunicación Alámbrica

El Módulo Inteligente y la computadora del usuario se comunican a través del protocolo Universal Serial Bus, convertido a través de un circuito integrado CH340 a señales del protocolo Universal Asynchronous Receiver-Transmitter (UART) a una tasa de transferencia de $115200\ [baudio/seg]$. Esto permite que el usuario envíe comandos al módulo para su ejecución. Además, permite que éste envíe los resultados de las mediciones en el formato de codificación solicitado.

El Módulo Inteligente y el Módulo Medidor se comunican a través de pulsos eléctricos sin utilización de protocolos. Como intermediario se encuentra el Módulo Aislante que permite maximizar la seguridad del sistema, garantizando

la aislación eléctrica entre ambos subcircuitos, evitando así posibles cortocircuitos entre la computadora y la carga a medir, alimentadas por la misma red eléctrica.

Comunicación Inalámbrica

El Módulo Inteligente puede enviar los resultados de las mediciones a un servidor externo a través de una conexión Wi-Fi (IEEE 802.11) [11].

Además, se implementó la posibilidad de alojar un bot de Telegram en el Módulo Inteligente, permitiendo la ejecución de comandos a través de Internet. Esta funcionalidad puede ser activada o desactivada en tiempo de compilación, según las necesidades a satisfacer.

Firmware del Módulo Inteligente

Se optó por desarrollar todo el código necesario en el lenguaje C++ [12]. A continuación se describen los motivos por el cual se utiliza este lenguaje:

- Los algoritmos son compilados a instrucciones en lenguaje máquina. Esto permite que el tiempo de ejecución sea menor respecto a la implementación en un lenguaje interpretado.
- Permite alocar memoria estáticamente (en la pila de ejecución o variables globales), reduciendo la fragmentación de la *Heap*.
- Permite definir macros de preprocesador (con o sin parámetros), para escribir código una sola vez e “instanciarlo” múltiples veces.
- Permite usar expresiones de preprocesador como `#if`, `#ifdef` y `#else`, para determinar qué porciones de código incluir para su posterior compilación.
- Permite crear clases concretas y abstractas, incluyendo jerarquías de herencia simple y múltiple.

- Permite declarar y definir funciones globales, métodos abstractos y concretos, incluyendo la posibilidad de sobrecargarlos variando la cantidad y/o tipo de dato de los parámetros.

- Permite agregar parámetros de plantilla (tipos, punteros, referencias, números, entre otros) a las clases, funciones y métodos, las cuales pueden ser instanciadas o invocadas con distintos valores de este tipo de parámetros.

Además, para evitar la duplicación de código entre distintos proyectos que puedan compartir requerimientos, se creó un *framework* [13] formado por distintas interfaces, clases y macros que proveen funcionalidades abstractas, las cuales serán completadas por el desarrollador del proyecto mediante instanciaciones, subclasificaciones, composiciones, entre otros.

De esta forma se obtiene un diseño modular y multiplataforma, que facilita las modificaciones del código y las migraciones hacia distintos microcontroladores.

2.4.2. Implementación de Funcionalidades

2.4.2.1. Funcionamiento General

Los pasos de ejecución del sistema son:

1. Una tarea, periódicamente cada `TIEMPO_MEDICION_HLW8012` milisegundos, “dispara” el evento de obtener una nueva medición.
2. Cuando se obtienen los resultados, son almacenados en un `JsonObject`, y se genera otro evento que lo contiene.
3. Éste provoca una secuencia de acciones a ejecutar:
    a. Pausar la recepción de pulsos, para que éstos dejen de provocar interrupciones de CPU.
    b. Respecto a la emisión a través del puerto `Serial`:

- Si la función serializadora no es nula, la invoca para enviar el `JsonObject` del evento por el puerto `Serial` en el formato especificado.

c. Respecto a la emisión a través del cuerpo de una solicitud HTTP:

- Si la función serializadora no es nula, la invoca para guardar el `JsonObject` del evento en el búffer de la solicitud, en el formato especificado.
- Si la conexión Wi-Fi está establecida, la solicitud es enviada a la dirección y puerto configurados, con el cuerpo, encabezados y credenciales especificados.

d. Retomar la recepción de pulsos, permitiendo realizar una nueva medición.

2.4.2.2. Comandos Implementados

Comando `/settearBD clave:String tipoDato:String valor:JsonVariant`

Si el tipo de dato (convertido a `String`) del valor guardado en el `JsonVariant` es igual a `tipoDato`, convierte ese valor al tipo correspondiente y lo guarda (junto a la `clave`) en la base de datos. Mientras tanto, imprime mensajes de depuración a la `salida` correspondiente.

Comando `/verBD`

Imprime el estado actual de la base de datos a la `salida` correspondiente.

Comando `/guardarBD [reiniciarSistema:bool]`

Persiste el estado actual de la base de datos. Mientras tanto, imprime mensajes de depuración a la `salida` correspondiente. Luego, si `reiniciarSistema` fue especificado y es un booleano que vale `true`, reinicia la ejecución del firmware.

Comando `/calibrarHLW potenciaActiva:unsigned int tensión:unsigned int corriente:double`

Si el tipo de dato del valor guardado en los tres `JsonVariant` es el correcto, los convierte a esos tipos, llama a los métodos de calibración provistos por la librería `HLW8012.h` (`expectedActivePower()`, `expectedVoltage()` y `expectedCurrent()`), guarda en la base de datos a los multiplicadores de calibración calculados por la librería, y ejecuta el comando `/guardarBD`. Mientras tanto, imprime mensajes de depuración a la `salida` correspondiente.

2.4.2.3. Opciones de Compilación

Bandera `HABILITAR_TELEGRAM`

Si fue definida, declara e inicializa las variables globales que permiten alojar un bot de Telegram desde el Módulo Inteligente, incluyendo la ejecución de los comandos a través de mensajes, sólo si la ID del usuario que los envió está en el conjunto (persistido en la base de datos) de usuarios admitidos.

Bandera `USAR_TELEGRAM_ASINCRONICO`

Esta bandera sólo puede ser usada en el caso de que `HABILITAR_TELEGRAM` haya sido definida. Sino, es ignorada.

Si `USAR_TELEGRAM_ASINCRONICO` fue definida, se usará la librería `AsyncTelegram2.h` [14] para implementar la comunicación en bajo nivel. Sino, se usará `UniversalTelegramBot.h` [15].

Bandera `EMITIR_PULSOS`

Si fue definida, declara e inicializa las variables globales que permiten generar una señal periódica a través de un pin de salida del Módulo Inteligente, la cual sería conectada a los pines de entrada del mismo módulo correspondientes a las señales `CF` y `CF1`, simulando el comportamiento del Módulo Medidor.

Bandera `USAR_CLAVES_COMPACTAS`

Si fue definida, los objetos JSON o MessagePack emitidos usarán claves de menor longitud, para reducir la cantidad de bytes necesaria para transmitirlos. Sino, se usarán claves de mayor longitud, más legible para el usuario.

Bandera `USAR_TLS`

Si fue definida, la comunicación Wi-Fi se realizará a través de una instancia de la clase `WiFiClientSecure` (que le agrega una capa de encriptación TLS [16] al canal TCP [17]); sino, de la clase `WiFiClient`.

Bandera `USAR_INTERRUPCIONES_HLW8012`

Si fue definida, el procesamiento de los pulsos generados por el Módulo Medidor se realizará mediante interrupciones de CPU, de forma independiente a la ejecución del programa. Sino, el procesamiento será de forma bloqueante, en algún momento de la ejecución.

Bandera `USAR_BD_EEPROM`

Si fue definida, se usará la librería `Preferences.h` [18] para implementar la persistencia en bajo nivel. Sino, se usará `EEPROM.h` [19].

3. Pruebas del sistema

3.1. Error relativo del instrumento de medición

Se han realizado distintas pruebas en el Laboratorio de Ensayos y Mediciones Eléctricas (LEME) [20] de la Facultad de Ingeniería de la UNLP, comparando las mediciones emitidas por el Medidor desarrollado frente a las visualizadas en los multímetros ZERA TPZ 108/308 (corriente eficaz, potencia activa eficaz, y factor de potencia) [21] y FLUKE 289 (tensión eficaz) [22]. En las figuras 6 y 7 se pueden observar ambos multímetros; y en la 8, el esquemático que muestra la conexión realizada.

Fig. 6: Vista superior del multímetro ZERA TPZ 108/308.

Fig. 7: Vista frontal del multímetro FLUKE 289.

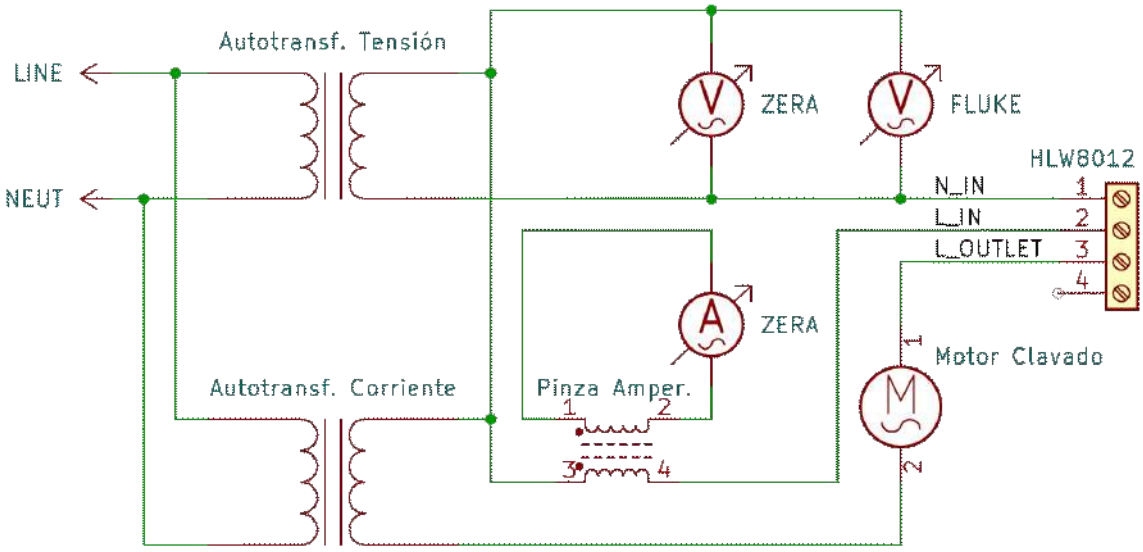

Fig. 8: Esquemático implementado para realizar las mediciones.

Todas las mediciones se realizaron usando el circuito de la figura 8, a temperatura ambiente de $23,7\ [ºCelsius]$, humedad del $46\ [\%]$ y presión de $1018\ [hectoPascal]$. Se pueden destacar los siguientes elementos dentro del circuito:

- **Autotransformador de tensión:** Permite entregar una tensión alterna en el rango de $[0; 250]\ [Volts]$, con la corriente de salida de hasta $22\ [Amperes]$.

- **Autotransformador de corriente:** Permite establecer la magnitud de la corriente a circular por el *shunt* resistivo del Módulo Medidor.

- **Motor de rotor anclado:** Permite establecer el ángulo de desfase de la corriente respecto a la tensión, y por lo tanto, el factor de potencia medido.

- **Pinza Amperimétrica:** Permite medir la magnitud de la corriente que circula por un cable, de forma aislada y sin necesidad de interrumpir el circuito. Se conectó al multímetro ZERA TPZ 108/308.

Medición de Tensión Eficaz

El primer conjunto de mediciones de tensión se realizó en vacío, variando la tensión del autotransformador en el subrango de $[50; 220]$ $[Volts]$, y usando las constantes de calibración que se definieron en una de las pruebas anteriores. En la tabla 1 se visualizan los valores observados.

#	Patrón [V]	Medidor [V]	Error
1	50,2	52	3,586%
2	101,9	104	2,061%
3	101,9	106	4,024%
4	151,5	155	2,310%
5	151,5	155	2,310%
6	200,7	206	2,641%
7	210,1	220	4,712%
8	221,9	227	2,298%

Tabla 1: Resultados de la medición de tensión en vacío, realizada antes de la recalibración.

De esta forma, el error promedio original (entre las 8 mediciones) es del $2,993$ $[\%]$.

Para realizar la calibración, se estableció la tensión a 220 $[Volts]$, la corriente a 0.25 $[Amperes]$, y el ángulo de desfase a 0 $[radianes]$. A partir de los valores medidos por los Patrones, se ejecutaron los comandos `/calibrarHLW 65 220 0.25` (para redefinir dichas constantes a nuevos valores) y `/guardarBD` (para persistirlas). En la figura 9 se muestra un resultado de tensión emitida por el medidor Patrón y el desarrollado.

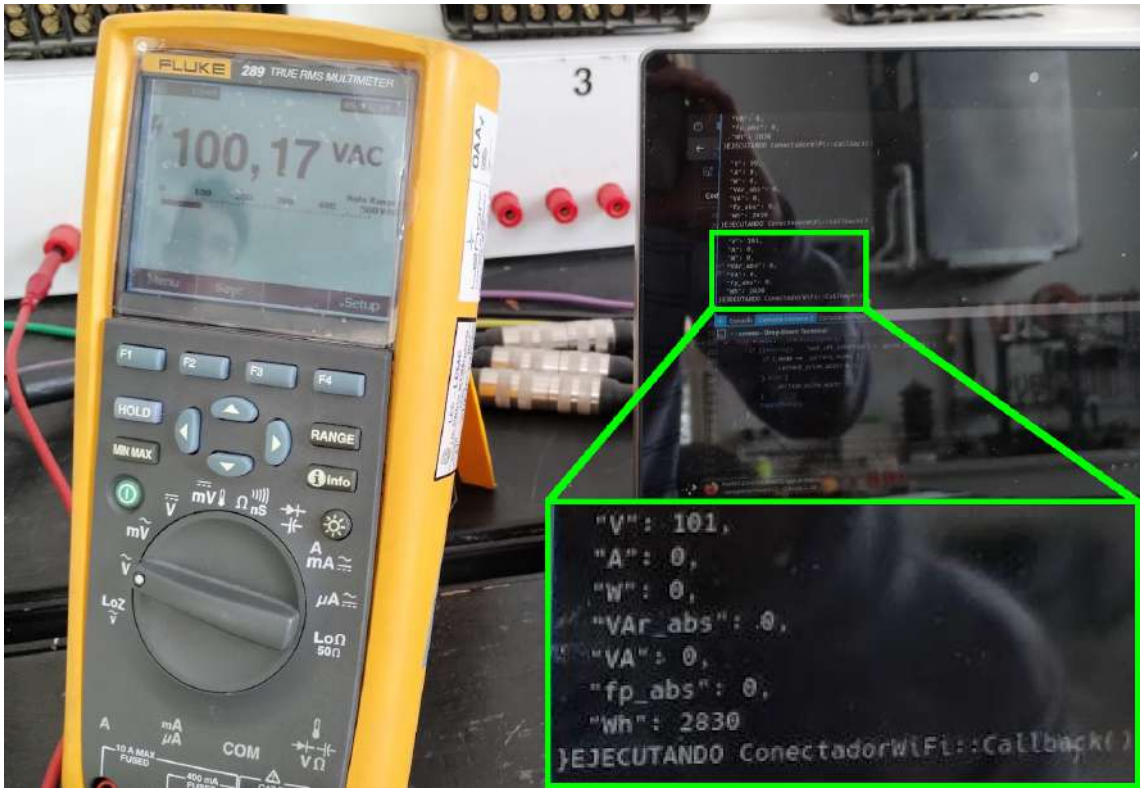

Fig. 9: Comparación de una tensión medida por el Patrón (izquierda) y Medidor (derecha, resultados en formato JSON indentado).

Al medir nuevamente la tensión, de la misma forma que la anterior, se obtuvieron los resultados visualizados en la tabla 2.

#	Patrón [V]	Medidor [V]	Error
1	50,3	50	-0,596%
2	100,2	99	-1,198%
3	110,1	109	-0,999%
4	150,3	150	-0,200%
5	200,5	198	-1,247%
6	200,7	205	2,143%
7	210,2	205	-2,474%
8	220,5	220	-0,227%

Tabla 2: Resultados de la medición de tensión en vacío, realizada después de la re-calibración.

Así, el nuevo error promedio (entre las 8 mediciones) es del $-\ 0,6\ [\%]$, observándose una mejora considerable.

Medición de Corriente Eficaz

Las mediciones de corriente se realizaron estableciendo el autotransformador de tensión a $220\ [Volts]$ y el ángulo de desfase a $0\ [radianes]$ (para únicamente medir potencia activa), variando la corriente del autotransformador en el subrango de $[0,25; 7]\ [Amperes]$. Los resultados obtenidos se visualizan en la tabla 3.

#	Patrón [A]	Medidor [A]	Error
1	0,24	0,24	0,000%
2	1,05	1,05	0,000%
3	1,53	1,52	-0,654%
4	2,05	2,03	-0,976%
5	2,55	2,53	-0,784%
6	3,01	3	-0,332%
7	5,03	4,94	-1,789%
8	6,06	6	-0,990%
9	7,38	7,1	-3,794%

Tabla 3: Resultados de la medición de corriente, realizada después de la re-calibración.

De esta forma, el error promedio (entre las 9 mediciones) es del $-\ 1,035\ [\%]$.

Medición de Potencia Activa Eficaz

El primer conjunto de mediciones de potencia activa se realizó estableciendo el autotransformador de tensión a $220\ [Volts]$ y el ángulo de desfase a $0,555\ [radianes]$ (para que el factor de potencia fuera $0,85$), variando la

corriente del autotransformador en el subrango de $[0,25; 7]$ $[Amperes]$. Los resultados obtenidos se visualizan en la tabla 4.

#	Tensión [V]	Corriente [A]	FP	Patrón [W]	Medidor [W]	Error
1	220,18	0,28	0,83	50,6	51	0,791%
2	220,18	0,62	0,83	114,3	117	2,362%
3	220,18	1,62	0,85	310,3	313	0,870%
4	220,18	1,71	0,85	318,6	323	1,381%
5	220,18	2,73	0,86	514,7	521	1,224%
6	220,18	4,03	0,86	758,2	773	1,952%
7	220,18	5,01	0,86	945,9	949	0,328%
8	220,18	6,07	0,86	1145	1164	1,659%
9	220,18	7,04	0,86	1332	1335	0,225%

Tabla 4: Resultados de la primera medición de potencia activa, realizada después de la re-calibración.

De esta forma, el error promedio (entre las 9 mediciones) es del $1,199$ $[\%]$.

El segundo conjunto de mediciones de potencia activa se realizó de forma similar al anterior, estableciendo el ángulo de desfase a $1,047$ $[radianes]$ (para que el factor de potencia fuera $0,5$). Se visualizan los valores obtenidos en la tabla 5.

#	Tensión [V]	Corriente [A]	FP	Patrón [W]	Medidor [W]	Error
1	220,1	0,25	0,53	28,87	29	0,450%
2	220,18	1	0,52	114,3	115	0,612%
3	220,18	5,05	0,48	530,9	533	0,396%

Tabla 5: Resultados de la segunda medición de potencia activa, realizada después de la recalibración.

De esta forma, el error promedio (entre las 3 mediciones) es del $0,486$ $[\%]$.

Medición de Factor de Potencia

El conjunto de mediciones de factor de potencia capacitivo se realizó estableciendo el autotransformador de tensión a $220\ [Volts]$ y el de corriente a $2\ [Amperes]$, variando el ángulo de desfase en el subrango de $[0; 1, 213]\ [radianes]$ (para que el factor de potencia se mantenga en el rango $[0, 35; 1]$ capacitivo). Los resultados obtenidos se visualizan en la tabla 6.

#	Tensión [V]	Corriente [A]	Patrón	Medidor	Error
1	220,35	1,9	1	1	0,000%
2	220,35	1,9	0,9	0,89	-1,111%
3	220,35	1,9	0,8	0,82	2,500%
4	220,35	1,9	0,7	0,7	0,000%
5	220,35	1,9	0,55	0,56	1,818%
6	220,35	1,9	0,45	0,46	2,222%
7	220,35	1,9	0,35	0,35	0,000%

Tabla 6: Resultados de la medición de factor de potencia capacitivo, realizada después de la re-calibración.

De esta forma, el error promedio (entre las 7 mediciones) es del $0, 776\ [\%]$.

El conjunto de mediciones de factor de potencia inductivo se realizó de forma similar al anterior, usando el subrango de ángulos de $[0, 555; 1, 266]\ [radianes]$ (para que el factor de potencia se mantenga en el rango $[0, 3; 0, 85]$ inductivo). Se visualizan los resultados obtenidos en la tabla 7.

#	Tensión [V]	Corriente [A]	Patrón	Medidor	Error
1	220,35	1,9	0,85	0,87	2,353%
2	220,35	1,9	0,5	0,51	2,000%
3	220,35	1,9	0,3	0,29	-3,333%

Tabla 7: Resultados de la medición de factor de potencia inductivo, realizada después de la recalibración.

De esta forma, el error promedio (entre las 3 mediciones) es del $0,340$ [%].

3.2. Medición de consumo energético

Las pruebas se realizaron conectando el cable de la fuente de alimentación de la computadora de escritorio (sobre la cual se ejecutaron los algoritmos) al tomacorrientes del Módulo de Potencia; y un cable USB desde el Módulo Inteligente a la computadora portátil (usada para recopilar y almacenar los resultados).

En la figura 10 se muestra el comando usado en la terminal Bash para guardar en archivos de texto, los resultados emitidos por el Módulo Inteligente:

```
screenlog -l "/dev/ttyUSB0" "115200"
```

Fig. 10: Comando de visualización y guardado.

Una vez obtenidos, para filtrarlos (manteniendo sólamente los objetos JSON) se usó el script mostrado en la figura 11.

```
#!/bin/bash

# Para cada archivo de texto crudo de resultados...
for selec in *.txt; do
    # Crear una tubería
    mkfifo "tuberia"

    # Si el archivo contiene "MULTIPLICADOR"...
    if grep -q "MULTIPLICADOR" "$selec"; then
        # Eliminar las primeras líneas hasta llegar a
        # la que contiene "MULTIPLICADOR" inclusive.
        sed -e "1,/MULTIPLICADOR/d" "$selec" \
        | sed -e "1,2d" > "tuberia" &
        # ^ Luego, eliminar las dos primeras líneas y
```

```
        # encolar el resultado en la tubería.
    else
        # Sino, encolar el contenido del archivo en la
        # tubería, sin procesamientos previos.
        cat "$selec" > "tuberia" &
    fi

    # Eliminar los objetos JSON de la tubería donde la
    # tensión medida haya sido 0 [V] y guardar el
    # resultado en un nuevo archivo.
    cat "tuberia" | jq \
    'del(. | select(.V == 0)) | objects' \
    > "Procesados/$selec.json"

    # Eliminar la tubería
    rm "tuberia"
done
```

Fig. 11: Script de filtrado de archivos de texto.

Además, para convertir los archivos JSON a CSV, se ejecutó el script de la figura 12.

```
#!/bin/bash

# Para cada archivo JSON de resultados...
for selec in "Procesados/"*.json; do
    # Convertir cada objeto en un array, convertirlo a CSV
    # y guardarlo en formato CSV en un nuevo archivo.
    cat "$selec" | jq --raw-output \
    '[.V, .A, .W, .VAr_abs, .VA, .fp_abs, .Wh] | @csv' \
    > "$selec.csv"
done
```

Fig. 12: Script de conversión de objetos JSON a tuplas en formato CSV.

Finalmente, se importaron los valores medidos a distintas hojas de cálculo, donde las columnas son los parámetros, y las filas son los valores de cada medición (una por segundo). En los resultados mostrados, se hará énfasis en la potencia activa, reactiva (absoluta) y aparente; además, todas estas mediciones se realizaron a la temperatura de $24\ [ºC]$.

Las especificaciones técnicas de la computadora, de la cual se registraron los consumos durante las pruebas, son:

- **CPU:** Intel Core i5-4460. Frecuencia básica de $3,20\ [GHz]$, frecuencia turbo de $3,40\ [GHz]$; 4 núcleos, 4 hilos, TDP de $84\ [W]$.
- **GPU:** Nvidia GeForce GTX 960. Frecuencia básica de $1,127\ [GHz]$, frecuencia turbo de $1,178\ [GHz]$; 1024 núcleos CUDA, consumo declarado de $120\ [W]$.
- **RAM:** 8GB DDR3.
- **Almacenamiento:** HDD 500GB de 3.5".
- **Fuente de Alimentación:** Sentey BXP65-0R 650W. Sin certificación de eficiencia.
- **Sistema Operativo:** Ubuntu 22.04. Arquitectura amd64.

3.2.1. Encendido de la computadora

Se registraron mediciones desde la pulsación del botón de encendido, hasta la visualización de la pantalla de bloqueo del sistema operativo instalado. Puede observarse en la figura 13 que el consumo fue variable en función del tiempo, lo cual es esperable, debido a que la secuencia de arranque está formada por distintas etapas que pueden requerir diferentes cantidades de energía eléctrica.

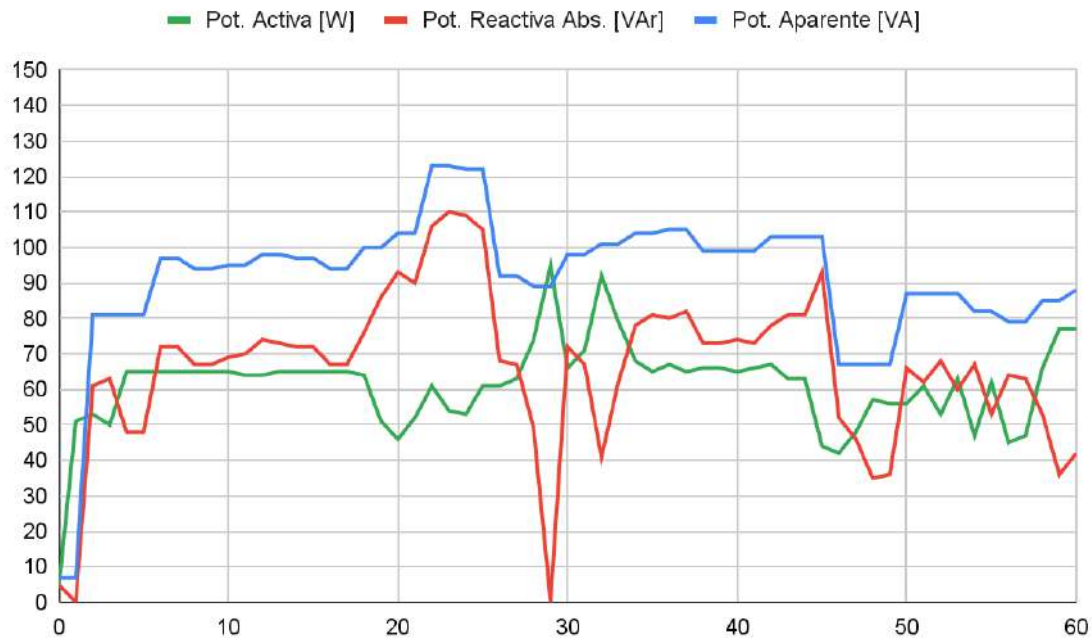

Fig. 13: Gráfico del consumo registrado durante el encendido de la computadora.

Luego de iniciar sesión, el consumo en reposo se estabilizó en $45\ [Watts]$ de potencia activa, $53\ [Volts - Amperes - reactivos]$ de reactiva absoluta, y $70\ [Volts - Amperes]$ de aparente.

3.2.2. Algoritmo de estudio

Se usó un algoritmo de multiplicación de dos matrices de $N \times N$ reales (variables de tipo `float`, punto flotante de 32 bits) capaz de ejecutarse en 1 o más núcleos de CPU; y además, otro compatible con las GPUs (Graphics Processor Unit, o Unidad de Procesamiento de Gráficos) de Nvidia, a través de la plataforma de desarrollo CUDA. Dicho algoritmo fue elegido ya que al ejecutarlo en una GPU se puede obtener mejoras en el tiempo de ejecución.

Algoritmo Secuencial

La multiplicación de dos matrices de 2048×2048 reales demoró $34,017\ [segundos]$, consumiendo una potencia activa total de

$2141\ [Watts - hora]$, reactiva absoluta total de $2228\ [Volts - Amperes - reactivos - hora]$, y aparente total de $3310\ [Volts - Amperes - hora]$. En la figura 14 se puede observar el gráfico correspondiente.

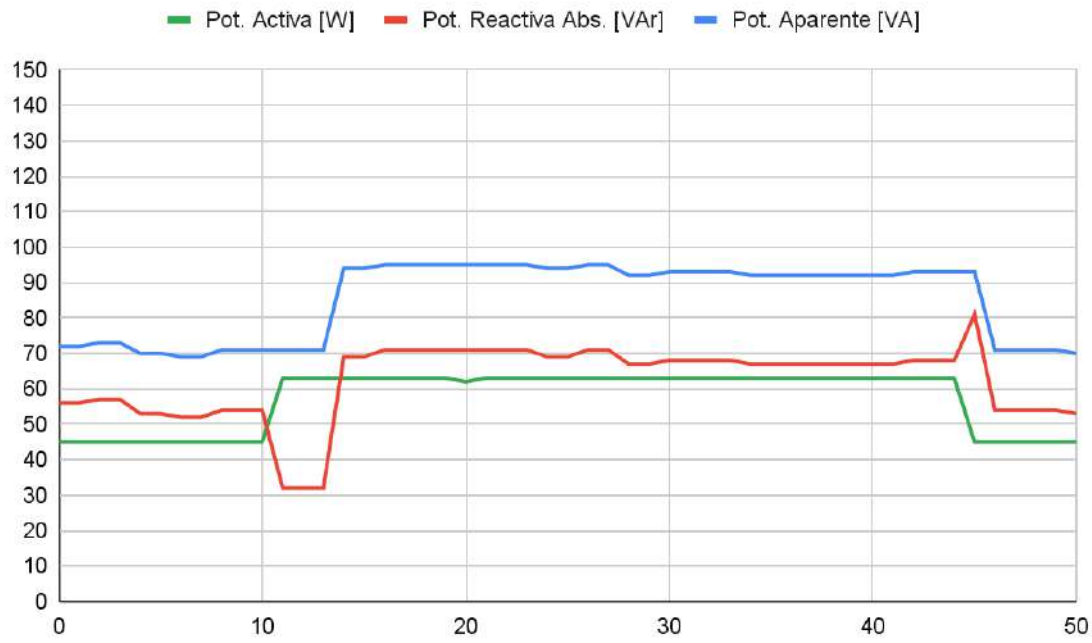

Fig. 14: Gráfico del consumo registrado durante la multiplicación de dos matrices de 2048×2048, usando 1 núcleo de CPU.

La multiplicación de dos matrices de 4096×4096 reales demoró $271,5\ [segundos]$, consumiendo una potencia activa total de $16947\ [Watts - hora]$, reactiva absoluta total de $18914\ [Volts - Amperes - reactivos - hora]$, y aparente total de $25487\ [Volts - Amperes - hora]$. En la figura 15 se puede observar el gráfico correspondiente.

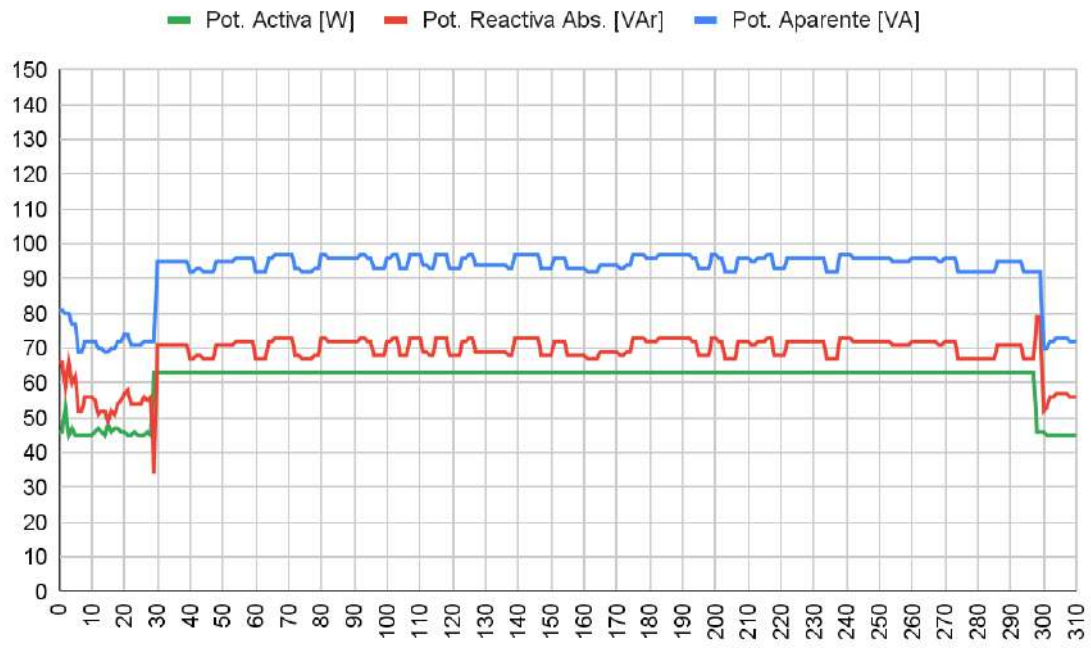

Fig. 15: Gráfico del consumo registrado durante la multiplicación de dos matrices de 4096×4096, usando 1 núcleo de CPU.

Algoritmo Paralelo

La multiplicación de dos matrices de 2048×2048 reales realizada usando 2 núcleos de CPU demoró $17,139$ $[segundos]$, consumiendo una potencia activa total de 1272 $[Watts - hora]$, reactiva absoluta total de 1116 $[Volts - Amperes - reactivos - hora]$, y aparente total de 1748 $[Volts - Amperes - hora]$. En la figura 16 se puede observar el gráfico correspondiente.

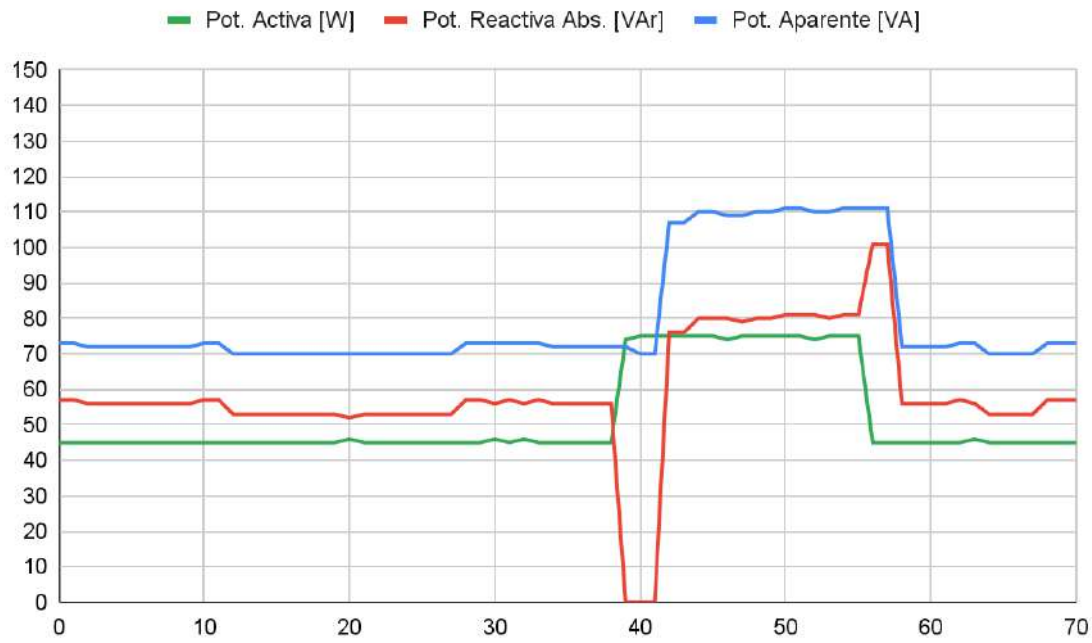

Fig. 16: Gráfico del consumo registrado durante la multiplicación de dos matrices de 2048×2048, usando 2 núcleos de CPU.

La multiplicación de dos matrices de 4096×4096 reales realizada usando 2 núcleos de CPU demoró $137,255\,[segundos]$, consumiendo una potencia activa total de $10238\,[Watts - hora]$, reactiva absoluta total de $11119\,[Volts - Amperes - reactivos - hora]$, y aparente total de $15238\,[Volts - Amperes - hora]$. En la figura 17 se puede observar el gráfico correspondiente.

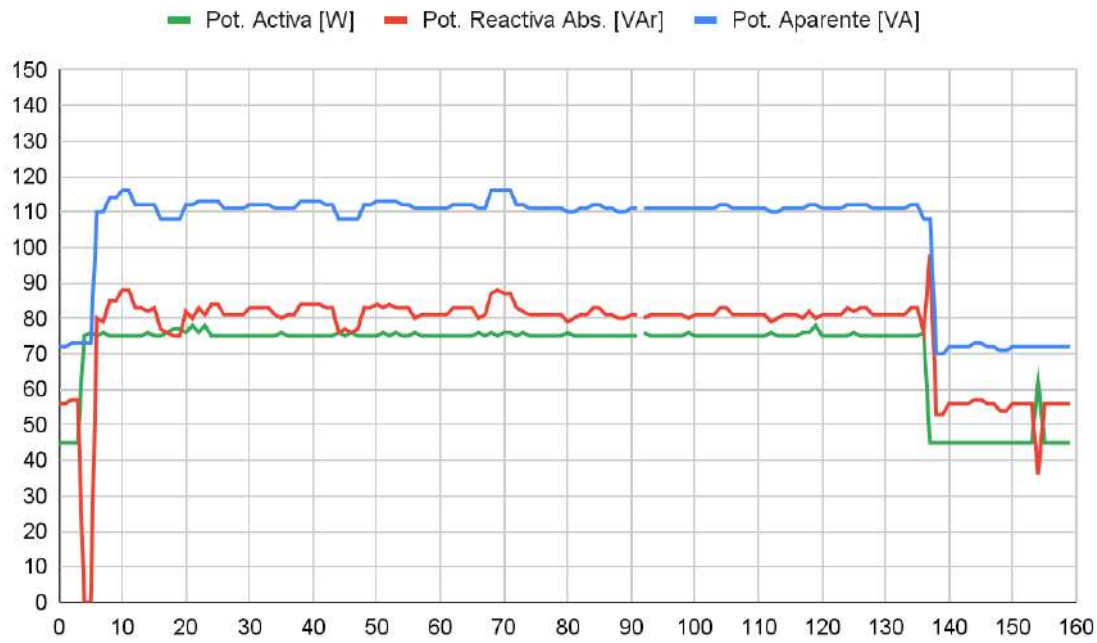

Fig. 17: Gráfico del consumo registrado durante la multiplicación de dos matrices de 4096×4096, usando 2 núcleos de CPU.

La multiplicación de dos matrices de 2048×2048 reales realizada usando 4 núcleos de CPU demoró $9,397$ $[segundos]$, consumiendo una potencia activa total de 865 $[Watts - hora]$, reactiva absoluta total de 787 $[Volts - Amperes - reactivos - hora]$, y aparente total de 1177 $[Volts - Amperes - hora]$. En la figura 18 se puede observar el gráfico correspondiente.

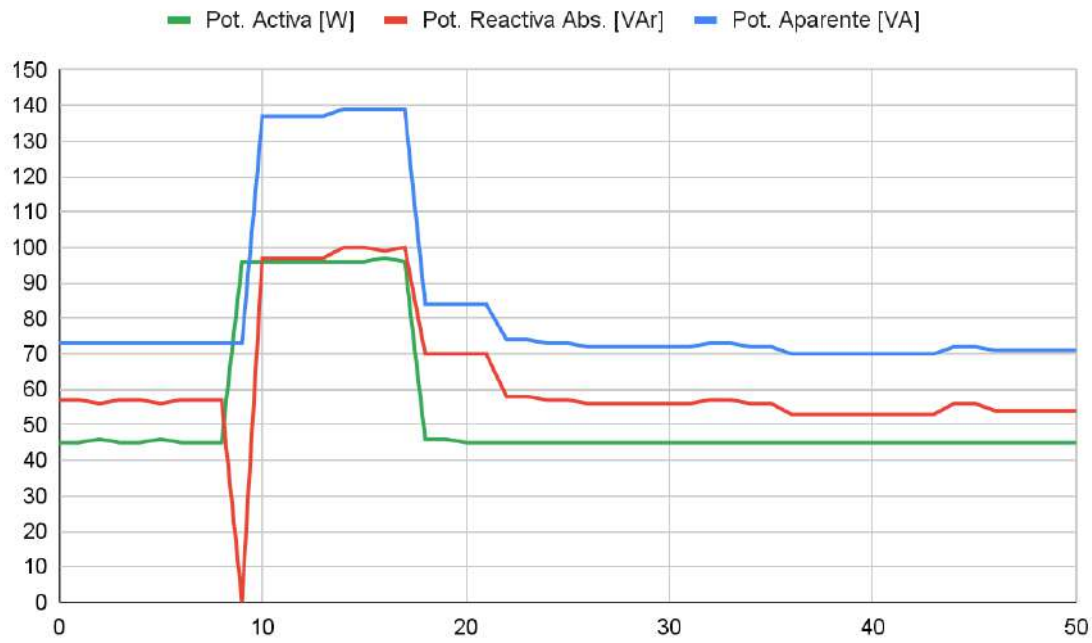

Fig. 18: Gráfico del consumo registrado durante la multiplicación de dos matrices de 2048×2048, usando 4 núcleos de CPU.

La multiplicación de dos matrices de 4096×4096 reales realizada usando 4 núcleos de CPU demoró $73,572$ $[segundos]$, consumiendo una potencia activa total de 7102 $[Watts - hora]$, reactiva absoluta total de 7082 $[Volts - Amperes - reactivos - hora]$, y aparente total de 10076 $[Volts - Amperes - hora]$. En la figura 19 se puede observar el gráfico correspondiente.

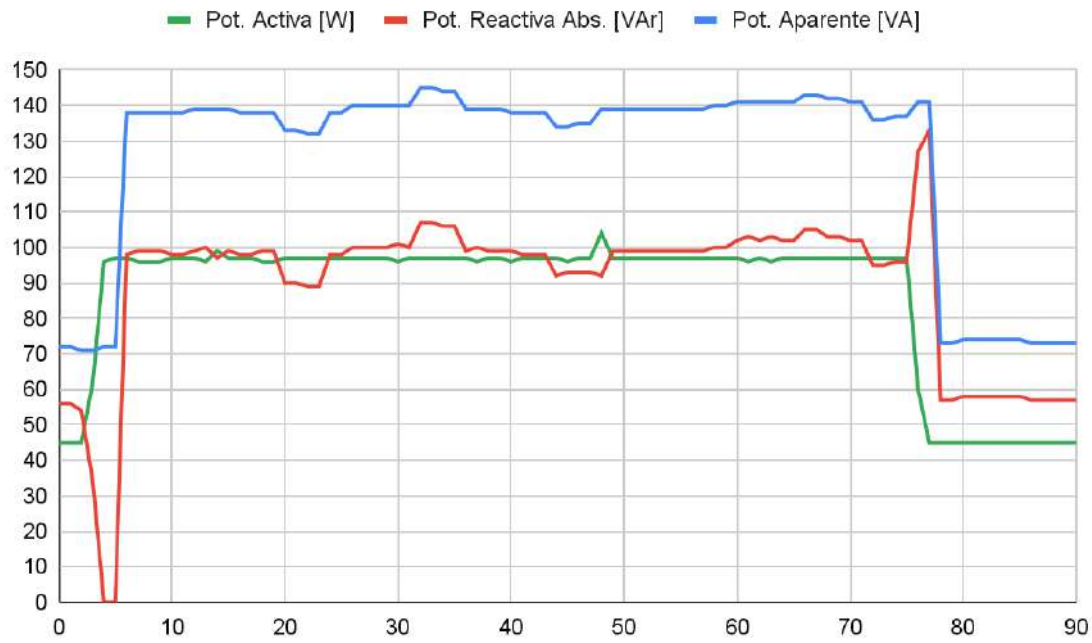

Fig. 19: Gráfico del consumo registrado durante la multiplicación de dos matrices de 4096×4096, usando 4 núcleos de CPU.

Algoritmo para GPUs

La multiplicación de dos matrices de 4096×4096 reales realizada usando 128 hilos por bloque de la GPU demoró $3,966\,[segundos]$, consumiendo una potencia activa total de $527\,[Watts - hora]$, reactiva absoluta total de $459\,[Volts - Amperes - reactivos - hora]$, y aparente total de $677\,[Volts - Amperes - hora]$. En la figura 20 se puede observar el gráfico correspondiente.

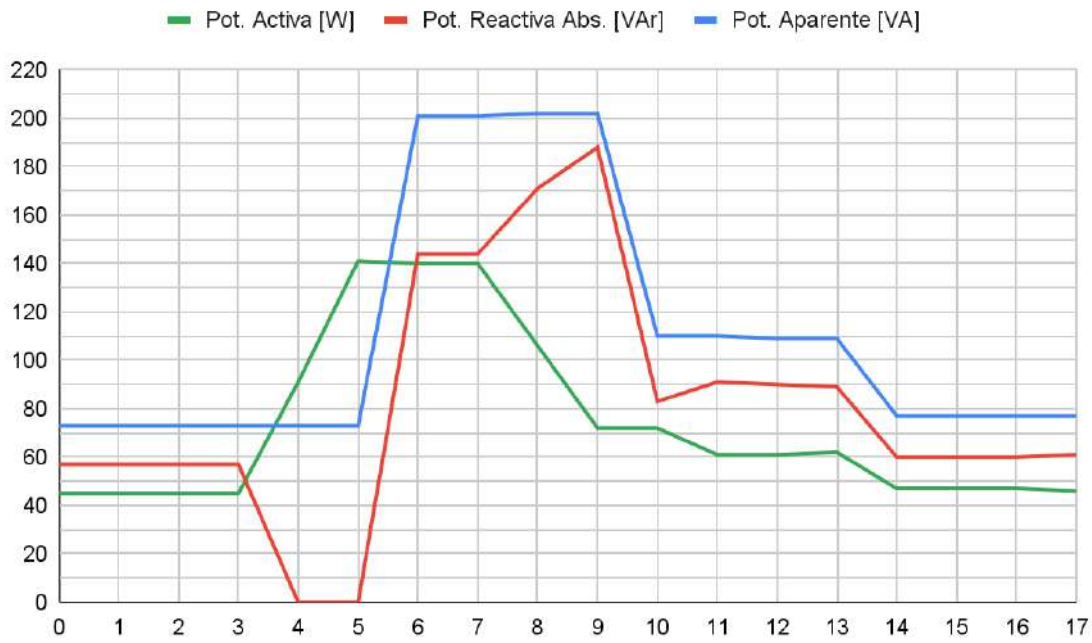

Fig. 20: Gráfico del consumo registrado durante la multiplicación de dos matrices de 4096×4096, usando 128 hilos por bloque de la GPU.

La multiplicación de dos matrices de 8192×8192 reales realizada usando 128 hilos por bloque de la GPU demoró $54,381\ [segundos]$, consumiendo una potencia activa total de $6819\ [Watts - hora]$, reactiva absoluta total de $7185\ [Volts - Amperes - reactivos - hora]$, y aparente total de $9964\ [Volts - Amperes - hora]$. En la figura 21 se puede observar el gráfico correspondiente.

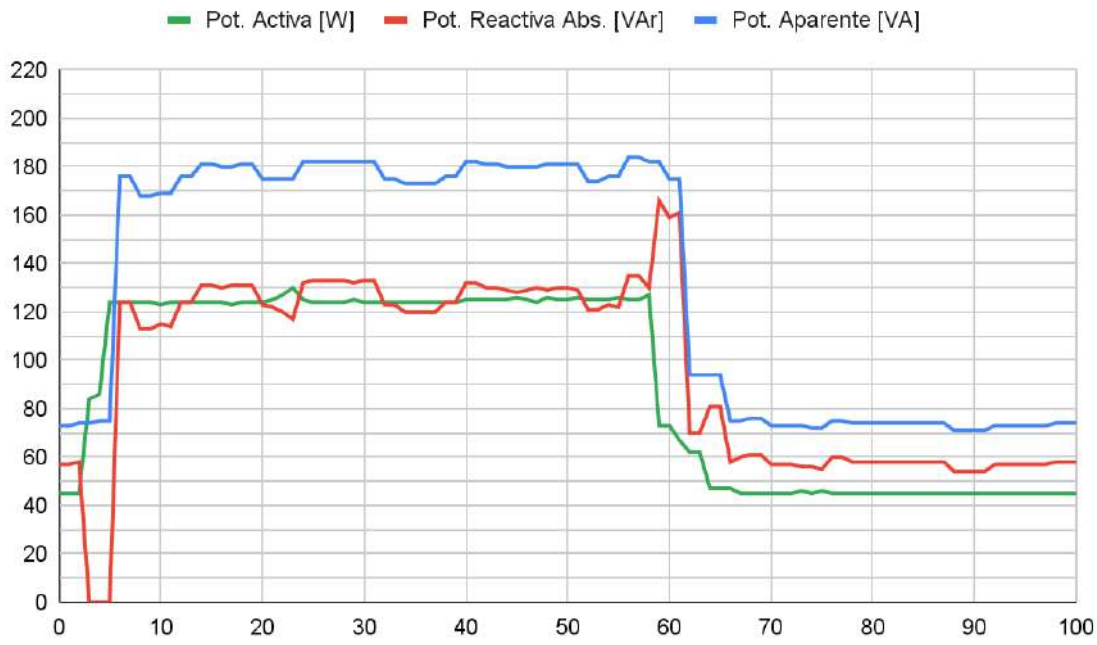

Fig. 21: Gráfico del consumo registrado durante la multiplicación de dos matrices de 8192×8192, usando 128 hilos por bloque de la GPU.

Resultados Generales

El resumen de los valores anteriores, para la multiplicación de dos matrices de 2048×2048 reales, puede encontrarse la tabla 8.

CantNúcleos	1	2	4
Tiempo [s]	34,017	17,139	9,397
P acumulada [Wh]	2.141,000	1.272,000	865,000
Q acumulada [VArh_abs]	2.228,000	1.116,000	787,000
S acumulada [VAh]	3.110,000	1.748,000	1.177,000
P promedio [W]	62,939	74,217	92,051
Q promedio [VAr_abs]	65,497	65,115	83,750
S promedio [VA]	91,425	101,990	125,253

Tabla 8: Resultados de las mediciones de tiempo y consumo, registrados durante la multiplicación de dos matrices de 2048×2048.

El resumen de los valores anteriores, para la multiplicación de dos matrices de 4096×4096 reales, puede encontrarse en la tabla 9.

CantNúcleos	1	2	4	CUDA (128 TPB)
Tiempo [s]	271,500	137,255	73,572	3,966
P acumulada [Wh]	16.947,000	10.238,000	7.102,000	527,000
Q acumulada [VArh_abs]	18.914,000	11.119,000	7.082,000	459,000
S acumulada [VAh]	25.487,000	15.238,000	10.076,000	677,000
P promedio [W]	62,420	74,591	96,531	132,879
Q promedio [VAr_abs]	69,665	81,010	96,259	115,734
S promedio [VA]	93,875	111,020	136,954	170,701

Tabla 9: Resultados de las mediciones de tiempo y consumo, registrados durante la multiplicación de dos matrices de 4096×4096.

Puede observarse en ambos casos, que al aumentar la cantidad de núcleos, aumentan los consumos promedios pero se reduce el tiempo de ejecución. En total, la energía requerida también se reduce.

4. Conclusiones y Trabajos Futuros

Se ha desarrollado el hardware y firmware de un medidor inteligente de energía eléctrica, que ha simplificado el proceso de relevamiento de datos, respecto al consumo de diferentes artefactos eléctricos (computadoras, servidores, electrodomésticos, entre otros) y/o subcircuitos de una instalación eléctrica existente.

Respecto al hardware, se destaca la importancia de construir prototipos asegurándose que todas las partes estén firmemente sujetas entre sí para maximizar la seguridad de uso, especialmente al utilizar tensiones eléctricas peligrosas para un ser humano. Además, el diseño actual ha brindado (o brindaría) los siguientes beneficios:

- La división del circuito en distintos módulos (interconectados mediante cables o conectores, formando “interfaces”) ha permitido simplificar el diseño de cada uno, abriendo la posibilidad a realizar futuras modificaciones a alguno de ellos sin afectar al resto, ahorrando de esta forma, tiempo y costos.

- Se facilita la producción en masa, ya que dado que un mismo módulo podría servir para distintos proyectos, cuando se planifique la producción de ese módulo, entonces se podría seleccionar la opción de fabricar cientos o miles de copias.

- En total, el costo por Medidor puede ser significativamente menor al de Medidores de Energía para tableros eléctricos comerciales, lo cual aumenta la viabilidad de instalar múltiples medidores por edificio, para realizar monitoreos exhaustivos del consumo en cada habitación.

Respecto al firmware, la decisión de crear un Framework ha permitido minimizar la deuda técnica a través de un diseño modular y multiplataforma, que facilita las modificaciones al código y las migraciones hacia otros microcontroladores. Además, el Subsistema de Comandos resultó vital para interactuar con el Módulo Inteligente, alámbrica (mediante una conexión USB) o inalámbricamente

(si se activara el bot de Telegram); y al combinarlo con el Subsistema de Persistencia, le permite al usuario modificar distintos valores de configuración sin la necesidad de recompilar el programa.

Por otro lado, se ha analizado el tiempo de ejecución y el consumo de un algoritmo de multiplicación de matrices en distintos escenarios. En función de los resultados obtenidos, se puede afirmar que los algoritmos paralelos pueden reducir el tiempo de ejecución (respecto a sus equivalentes secuenciales), y en consecuencia, también permiten reducir la cantidad de energía eléctrica total requerida. Por otra parte, se puede afirmar que la fuente de alimentación de la computadora[3] no cuenta con un circuito de corrección de factor de potencia, comprobándose que la energía aparente es mayor a la energía activa, reduciendo el aprovechamiento de la instalación eléctrica. Como ventaja de la fuente, se puede mencionar que provee una demanda de corriente muy baja (respecto a las fuentes de alimentación “lineales”).

Sin embargo, estas pruebas por sí mismas no fueron suficientes. Para garantizar que el Medidor registra valores certeros (dentro de un determinado margen de error, y sin tener en cuenta la distorsión armónica que se puede producir), ha resultado indispensable contrastarlos con los obtenidos por medidores de mayor precisión y exactitud. De esta forma, luego de calibrar el Medidor, se determinó que el el error relativo máximo (entre todos los parámetros medidos) está entre $-\ 1,035\ [\%]$ y $1,199\ [\%]$, resultando en un rango moderado pero aceptable.

Para mejorar el sistema, se propone:

- Diseñar y fabricar el circuito impreso final.
- Implementar un *dashboard* web que permita visualizar los resultados obtenidos en tiempo real.

[3] Se debe aclarar que las fuentes conmutadas poseen componentes alineales (diodos, tiristores, transistores, etc), es decir, que no se los puede caracterizar como elementos que siguen las leyes fundamentales de circuitos. El estudio de estos elementos es mucho más profundo del alcance de este trabajo, pero se puede afirmar que los elementos alineales son generadores de distorsión armónica y reducidores del factor de potencia.

- Emitir alertas cuando los valores medidos no estén en un rango numérico determinado.
- Agregar LEDs para indicar con mayor detalle el estado actual de la conexión Wi-Fi (por ejemplo, la intensidad de señal y si hay una transferencia de datos en curso).
- Reemplazar el ESP8266 del Módulo Inteligente por un ESP32, para que éste aloje al bot de Telegram y sea capaz de ejecutar los comandos recibidos.
- Re-calibrar el medidor anualmente y cumpliendo las normas IRAM, para mantener su precisión a largo plazo.
- Hacer que el Módulo Inteligente pueda conectarse a otros microcontroladores a través de un puerto UART secundario.
- Usar dos Módulos Medidores a la vez (con sus respectivos Aislantes), uno midiendo potencia activa y tensión, y el otro potencia activa y corriente.
- Aplicar modelos de Regresión Lineal para ajustar los resultados emitidos por el Medidor a los medidos por el Patrón.
- Mejorar el subcircuito de pausado de señales del Módulo Aislante, reemplazando una compuerta Y por dos *latches* tipo D (uno por pin de salida del Módulo Medidor).
- Agregar subcircuitos al Módulo de Potencia, encargados de las protecciones frente a sobretensiones y descargas atmosféricas.

Bibliografía

[1] Página oficial del Instituto de Investigación en Informática LIDI.

[2] Khajenasiri, I., Estebsari, A., Verhelst, M., & Gielen, G. (2017). A review on Internet of Things solutions for intelligent energy control in buildings for smart city applications. Energy Procedia, 111, 770-779.

[3] Smitha, S. D., Savier, J. S., & Chacko, F. M. (2013, June). Intelligent control system for efficient energy management in commercial buildings. In 2013 Annual International Conference on Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy (pp. 1-6). IEEE.

[4] Pi Puig, M., Paniego, J. M., Medina, S., Rodríguez Eguren, S., Libutti, L., Lanciotti, J., ... & De Giusti, L. (2019). Intelligent Distributed System for Energy Efficient Control. In Cloud Computing and Big Data: 7th Conference, JCC&BD 2019, La Plata, Buenos Aires, Argentina, June 24–28, 2019, Revised Selected Papers 7 (pp. 51-60). Springer International Publishing.

[5] Hoja técnica del microcontrolador ESP8266EX versión 7.0, Espressif Systems. Año 2023.

[6] Hoja técnica del circuito integrado HLW8012 versión 1.3, Hiliwei Tech. Año 2015.

[7] Dias, R., Scaramutti, J., Arrojo, C. D., Nastta, H. A. (2013). Análisis comparativo de sistemas de medición inteligentes en el contexto de las redes inteligentes. In II Jornadas de Investigación y Transferencia de la Facultad de Ingeniería, La Plata, Buenos Aires, Argentina, May XX-YY, 2013.

[8] Página oficial del Ente Nacional Regulador de la Electricidad.

[9] Página oficial del Organismo de Control de Energía Eléctrica de la Provincia de Buenos Aires.

[10] Página oficial del software KiCad.

[11] Página oficial del estándar IEEE 802.11.

[12] Documentación de la versión 11 del lenguaje C++.

[13] Definición de "Framework", Code Institute. Año 2015.

[14] Código fuente de la librería `AsyncTelegram2.h`, Tolentino Cotesta.

[15] Código fuente de la librería `UniversalTelegramBot.h`, Brian Lough.

[16] Definición de "TLS", Cloudflare.

[17] Documento RFC del protocolo TCP. Año 1981.

[18] Código fuente de la librería `Preferences.h` específica al ESP8266, Volodymyr Shymanskyy.

[19] Código fuente de la librería `EEPROM.h` específica al ESP8266, ESP8266 Community Forum.

[20] Página oficial del Laboratorio de Ensayos y Mediciones Eléctricas.

[21] Catálogo de medidores patrones marca ZERA.

[22] Página web del multímetro FLUKE 289.

Anexo A: Detalles de Implementación

A.1 Codificación de Resultados

En los resultados se indica la tensión, corriente, potencias activas, reactiva absoluta y aparente, factor de potencia absoluto y energía activa. Entonces, se usó la librería `ArduinoJson.h` [A1] para simplificar la creación de documentos JSON y MessagePack.

Para cada uno de los canales de comunicación (instancias de subclases de `Stream`), se permite configurar dinámicamente en qué formato codificar los resultados. Este parámetro es importante, ya que elegir una codificación compacta reduce la cantidad de bytes enviados, y por ende, el ancho de banda y energía requeridos.

Objetos JSON comprimidos (`"JSONc"`)

Este formato se codifica usando caracteres Unicode, por lo que son legibles por un humano, y también fácilmente interpretables por un algoritmo. Se eliminan los caracteres blancos para reducir la cantidad de bytes necesarios. En la figura A1 se observa un ejemplo de resultado emitido de esta forma.

```
{"tensión_V":220,"corriente_A":0.222770149,"potenciaActiva
_W":48,"potenciaReactivaAbsoluta_VAr":9,"potenciaAparente_
VA":49,"factorPotenciaAbsoluto":0.979591837,"energiaActiva
_Wh":931}
```

Fig. A1: Objeto JSON comprimido con claves largas.

Si la bandera `USAR_CLAVES_COMPACTAS` fue definida, se usan claves de menor longitud. En la figura A2 se observa un ejemplo de resultado emitido de esta forma:

```
{"V":220,"A":0.222770149,"W":48,"VAr_abs":9,"VA":49,"fp_ab
```

```
s":0.979591837,"Wh":931}
```

Fig. A2: Objeto JSON comprimido con claves cortas.

Objetos JSON indentados (`"JSONi"`)

Este formato es similar al anterior, pero no se eliminan los caracteres blancos. Así, se prioriza la legibilidad, a cambio de una mayor longitud de las cadenas resultantes. En la figura A3 se observa un ejemplo de resultado emitido de esta forma.

```
{
      "tensión_V": 220,
      "corriente_A": 0.222770149,
      "potenciaActiva_W": 48,
      "potenciaReactivaAbsoluta_VAr": 9,
      "potenciaAparente_VA": 49,
      "factorPotenciaAbsoluto": 0.979591837,
      "energiaActiva_Wh": 931
}
```

Fig. A3: Objeto JSON indentado con claves largas.

Si la bandera `USAR_CLAVES_COMPACTAS` fue definida, se usan claves de menor longitud. En la figura A4 se observa un ejemplo de resultado emitido de esta forma:

```
{
      "V": 220,
      "A": 0.222770149,
      "W": 48,
      "VAr_abs": 9,
      "VA": 49,
      "fp_abs": 0.979591837,
```

```
    "Wh": 931
}
```

Fig. A4: Objeto JSON indentado con claves cortas.

Objetos MessagePack ("MP")

En este caso, se usan bytes en binario, por lo que el nivel de compresión es mayor, sacrificando la legibilidad. En la figura A5 se muestra Representando los bytes en notación hexadecimal, los objetos quedan:

```
87 AA 74 65 6E 73 69 C3 B3 6E 5F 56 CC DC AB 63 6F 72 72
69 65 6E 74 65 5F 41 CB 3F CC 83 BB 74 3D 72 2F B0 70 6F
74 65 6E 63 69 61 41 63 74 69 76 61 5F 57 30 BC 70 6F 74
65 6E 63 69 61 52 65 61 63 74 69 76 61 41 62 73 6F 6C 75
74 61 5F 56 41 72 09 B3 70 6F 74 65 6E 63 69 61 41 70 61
72 65 6E 74 65 5F 56 41 31 B6 66 61 63 74 6F 72 50 6F 74
65 6E 63 69 61 41 62 73 6F 6C 75 74 6F CB 3F EF 58 D0 FA
EA FE 77 B0 65 6E 65 72 67 69 61 41 63 74 69 76 61 5F 57
68 CD 03 A3
```

Fig. A5: Objeto MessagePack con claves largas.

Si la bandera USAR_CLAVES_COMPACTAS fue definida, se usan claves de menor longitud. En la figura A6 se observa un ejemplo de resultado emitido de esta forma, también en hexadecimal:

```
87 A1 56 CC DC A1 41 CB 3F CC 83 BB 74 3D 72 2F A1 57 30
A7 56 41 72 5F 61 62 73 09 A2 56 41 31 A6 66 70 5F 61 62
73 CB 3F EF 58 D0 FA EA FE 77 A2 57 68 CD 03 A3
```

Fig. A6: Objeto MessagePack con claves cortas.

Ninguno (`"OFF"` o `""`)

Como su nombre lo indica, no se envían los resultados en ninguna codificación.

A.2 Framework Propio

A.2.1 Persistencia de Configuraciones

Para brindar esta funcionalidad, se optó por implementar una base de datos de tipo Clave-Valor [A2], donde las claves serán cadenas de texto, y los valores pueden ser de los siguientes tipos de datos:

- Números:
  - `bool`
  - `float`, `double`
  - `signed char`, `unsigned char`
  - `signed int`, `unsigned int`
  - `signed short`, `unsigned short`
  - `signed long`, `unsigned long`
  - `signed long long`, `unsigned long long`
- Cadenas de texto:
  - `StringAbstracta`
  - `const char *`
- Serializables a JSON y MessagePack:
  - `JsonVariant`
  - `JsonArray`
  - `JsonObject`

Se usó el patrón *Template Method* [A3], donde la superclase:

- Declara los métodos abstractos protegidos (a ser implementados por las subclases), que representan operaciones en bajo nivel.

- Define los métodos concretos públicos (los cuales llaman a los anteriores durante su ejecución), que son las operaciones implementadas en alto nivel.

Las clases relevantes son:

- **Interfaz `Inicializable`:** Es implementada por los objetos que no puedan inicializarse completamente en sus constructores (especialmente las instanciadas en variables globales), sino que requieren que parte de esas instrucciones se ejecuten dentro de una función global como `setup()` o `loop()`.
- **Clase abstracta `BaseDatos<size_t MAX_LONGITUD_CLAVES, size_t MAX_LONGITUD_STRINGS>`:** Subclase de `Printable` e `Inicializable`, que a su vez, es la superclase explicada anteriormente.
- **Clase concreta `BaseDatosEEPROM<size_t CAPACIDAD_JSON, size_t MAX_LONGITUD_CLAVES, size_t MAX_LONGITUD_STRINGS, typename T_EEPROM>`:** Subclase de `BaseDatos<MAX_LONGITUD_CLAVES, MAX_LONGITUD_STRINGS>`, que usa una instancia de `StaticJsonDocument<CAPACIDAD_JSON>` para almacenar las claves y valores en un objeto MessagePack, y persistirlos en la memoria EEPROM (o equivalente) del Módulo Inteligente a través de un objeto de tipo `T_EEPROM`.

A.2.2 Implementación de Callbacks

Las clases relevantes son:

- **Interfaz `CallbackResultado<typename... T>`:** Subclase de `Printable`, que representa una secuencia de instrucciones a ejecutar al producirse un evento determinado.
- **Clase concreta `CallbackCompuesto<size_t CAPACIDAD_HIJOS, typename... T>`:** Implementa la interfaz `CallbackResultado<T...>`, y conoce a una lista de punteros a

objetos que también implementan esta interfaz, formando el patrón *Composite* [A4].

- **Clase concreta `CallbackEmisorVarianteJSON<typename T_VARIANTE>`:** Implementa la interfaz `CallbackResultado<T_VARIANTE>`, y conoce a una instancia de alguna subclase de `Print` y un puntero a una función que recibiendo un objeto de tipo `T_VARIANTE` y otro subclase de `Print`, imprime al segundo una representación del primero, retornando la cantidad de bytes usados.

A.2.3 Subsistema de Medidores

Las clases relevantes son:

- **Interfaz `CondicionResultado<typename... T>`:** Subclase de `Printable`, que representa una condición a satisfacer.

- **Clase abstracta `Medidor<typename T_RESULTADO>`:** Subclase de `Printable`, que provee una base para crear objetos que puedan medir parámetros, realizando acciones con los valores medidos, sólamente cuando éstos cumplen una condición determinada opcional. Para ello, conoce a una instancia que implementa a `CallbackResultado<T_RESULTADO>` y otra (la cual puede ser un puntero nulo) que implemente la interfaz `CondicionResultado<T_RESULTADO>`.

- **Clase abstracta `TareaMedidora<typename T_RESULTADO>`:** Subclase de `Medidor<T_RESULTADO>` y de `Task`, que inicia la medición al habilitarse la tarea, y cuando ésta se ejecuta, obtiene un resultado y finaliza la medición.

- **Clase abstracta `MedidorInstantaneo<typename T_RESULTADO>`:** Subclase de `TareaMedidora<T_RESULTADO>`, que provee una base

para crear medidores que obtienen sus resultados instantáneamente (en lugar de esperar un determinado tiempo tras habilitarse).

A.2.4 Subsistema de Entrada/Salida

Las clases relevantes son:

- **Clase abstracta `Pulsable`:** Subclase de `Printable`, que provee una base para crear objetos que puedan encenderse y apagarse.
- **Clase abstracta `Pin<byte MODO_PIN>`:** Subclase de `Printable` e `Inicializable`, que provee una base para crear objetos que representan pines de entrada o salida.
- **Clase concreta `PinEntradaDigital<byte MODO_PIN[1]>`:** Subclase de `Pin<MODO_PIN>` y `EntradaDigital`, que encapsula en un objeto a un pin de este tipo, permitiendo leer su valor actual (`HIGH` o `LOW`) y vincular una función global a ser invocada cuando la señal recibida genere interrupciones de CPU.
- **Clase concreta `PinSalidaDigital`:** Subclase de `Pin<OUTPUT>` y `SalidaDigital`, que encapsula en un objeto a un pin de este tipo, permitiendo encenderlo y apagarlo a través de los métodos de `Pulsable`.

A.2.5 Subsistema de Comandos

A través de los puertos UART e I²C, así como el bot de Telegram (si fue habilitado), se permite ingresar comandos con un formato definido. En la figura A7 se puede observar dos ejemplos de sintaxis válidas.

```
/nombreComando
/nombreComando "arg1" 2 {"arg3": null} [4.56, 7.89, true]
```

Fig. A7: Sintaxis de los comandos y sus argumentos.

Algunas consideraciones:

- El nombre del comando debe componerse de uno o más caracteres no-blancos. Es decir, cualquier tipo de caracter excepto espacios, tabuladores, saltos de línea, etc.
- Un comando puede invocarse con 0 o más argumentos, pero esta cantidad puede limitarse a un rango determinado al crearlo (y se validará que dicha cantidad sea la adecuada al invocarlo).
- Los tipos de datos soportados para los argumentos son los mismos que los del Subsistema de Persistencia, excepto por `StringAbstracta`. Sin embargo, el usuario debe ingresarlos cumpliendo la sintaxis JSON [A5], para que puedan ser interpretados y convertidos correctamente.

Las clases relevantes son:

- **Registro `CanalBidireccional<typename T_ENTRADA, typename T_SALIDA>`:** Representa un canal de comunicación bidireccional a partir de un canal de entrada (del cual leer datos) y uno de salida (al cual escribir resultados).
- **Clase concreta `DetectorStream`:** Subclase de `MedidorInstantaneo<CanalBidireccional<Stream, Print>>` que dado un objeto subclase de `Stream`, cuando éste tiene bytes disponibles para leer, "mide" retornando un `CanalBidireccional<Stream, Print>` con ese `Stream` como canal de entrada y salida.
- **Clase concreta `InterpreteComandos<size_t CAPACIDAD_JSON_FINAL, size_t CAPACIDAD_JSON_INTERMEDIO, size_t CAPACIDAD_STRING_COMANDO, size_t CAPACIDAD_STRING_NUMERO>`:** Subclase de `Medidor<CanalBidireccional<JsonDocument, Print>>` y de `CallbackResultado<CanalBidireccional<Stream, Print>>`,

que verifica que los bytes disponibles del `Stream` cumplan la sintaxis de comandos y argumentos, y en ese caso, construye un objeto JSON que contiene el nombre del comando y el array de argumentos.

- **Clase concreta `Comando<size_t CAPACIDAD_NOMBRE>`:** Subclase de `Printable` que encapsula a los metadatos de ese comando, como su nombre, cantidad de argumentos que recibe, y la función global a ejecutar cuando el usuario invoca al comando.
- **Clase concreta `InvocadorCallbacks<size_t CAPACIDAD_COMANDOS, size_t CAPACIDAD_NOMBRE_COMANDOS, size_t CAPACIDAD_PILA_JSON, size_t CAPACIDAD_MENSAJE_JSON>`:** Implementa la interfaz `CallbackResultado<CanalBidireccional<JsonDocument, Print>>`, buscando al `Comando<CAPACIDAD_NOMBRE>` cuyo nombre sea el especificado, realizando las validaciones necesarias y ejecutando a la función asociada con los argumentos ingresados.

A.2.6 Conexión a Wi-Fi

Las clases relevantes son:

- **Clase concreta `PulsableCompuesto<size_t CAPACIDAD_HIJOS>`:** Subclase de `Pulsable` que conoce a una lista de punteros a objetos que también implementan esta interfaz, formando el patrón *Composite* [A4].
- **Clase concreta `ConectadorWiFi<size_t TAMAÑO_NOMBRE, size_t TAMAÑO_CONTRASEÑA>`:** Subclase de `Task` e `Inicializable` que inicia el proceso de emparejamiento Wi-Fi y verifica periódicamente si se estableció o perdió la conexión, encendiendo o apagando a un `Pulsable` respectivamente.

A.2.7 Tareas Extras

Las clases relevantes son:

- **Clase abstracta `IniciadorTareas`:** Subclase de `Task` y `Printable` que provee una base para crear objetos que puedan iniciar la ejecución de otra tarea, cada cierta cantidad de tiempo.
- **Clase concreta `IniciadorInfinito`:** Subclase de `IniciadorTareas` que realiza lo anterior una cantidad infinita de veces (o, si se especificó una `CondicionResultado<IniciadorTareas>`, mientras ésta sea falsa).

A.3 Reporte Periódico de Mediciones

Las clases relevantes son:

- **Clase concreta `ConversorMedicionHLW8012JSON<size_t CAPACIDAD_DOCUMENTO>`:** Implementa la interfaz `CallbackResultado<WrapperPuntero<HLW8012>>`, convirtiendo los resultados de las mediciones del HLW8012 a `JsonObject`s, usando las claves mencionadas anteriormente.
- **Clase concreta `MedidorHLW8012`:** Subclase de `MedidorInstantaneo<WrapperPuntero<HLW8012>>` que "mide" retornando a la instancia de la clase `HLW8012` que conoce.
- **Clase concreta `PausadorMedicionesHLW8012<typename... T_HIJO>`:** Implementa la interfaz `CallbackResultado<T_HIJO...>`, encendiendo a un `Pulsable`, enviando los resultados al `CallbackResultado<T_HIJO...>` hijo, y apagando a ese `Pulsable`. Así, junto a otras clases, se forma el patrón *Decorator* [A6].

A.4 Estructura del Proyecto

- Carpeta `/`:
  - Archivo `MedidorConsumoHLW8012.ino`: Definición de macros con argumentos, manejadores de interrupciones de CPU (si la

bandera `USAR_INTERRUPCIONES_HLW8012` fue definida) y funciones globales (`setup()` y `loop()`).

- Carpeta `src/`:
  - Archivo `Constantes.h`: Las banderas y otras constantes (o macros sin argumentos) de preprocesador.
  - Archivo `DeclaracionVariablesGlobales.h`: Importación de archivos (del Framework y del Proyecto) y declaración de todas las variables globales.
  - Archivo `InicializacionVariablesGlobales.cpp`: Inicialización de las variables del archivo anterior.
  - Carpeta `Dominio/`:
    - Archivo `Comandos.h`: Declaración de las funciones globales que serán llamadas cuando el usuario ejecute los comandos respectivos.
    - Archivo `Comandos.cpp`: Definición de las funciones del archivo anterior.
    - Carpeta `Mediciones/`:
      - Archivo `CLASE_ConversorMedicionHLW8012JSON.h`: Declaración y definición de la clase `ConversorMedicionHLW8012JSON`, y de las macros según la bandera `USAR_CLAVES_COMPACTAS`.
      - Archivo `CLASE_MedidorHLW8012.h`: Declaración y definición de la clase `MedidorHLW8012`.

    - Archivo `CLASE_PausadorMedicionesHLW8012.h`: Declaración y definición de la clase `PausadorMedicionesHLW8012`.

- Carpeta `MatyLabs/`: Enlace simbólico a la carpeta raíz del Framework.

A.5 Bibliografía del Anexo A

[A1] Documentación de la versión 6 de la librería `ArduinoJson.h`, Benoît Blanchon.

[A2] Definición de "Base de Datos Clave-Valor", Redis.

[A3] Patrón de diseño "Template Method", Refactoring Guru.

[A4] Patrón de diseño "Composite", Refactoring Guru.

[A5] Documentación oficial del estándar JSON, ECMA International.

[A6] Patrón de diseño "Decorator", Refactoring Guru.

Anexo B: Esquemáticos Detallados

B.1 Módulo Inteligente

Este módulo usa un microcontrolador ESP8266 para realizar el procesamiento y comunicación requeridos, a través del firmware desarrollado. En la figura B1 se puede observar su esquemático:

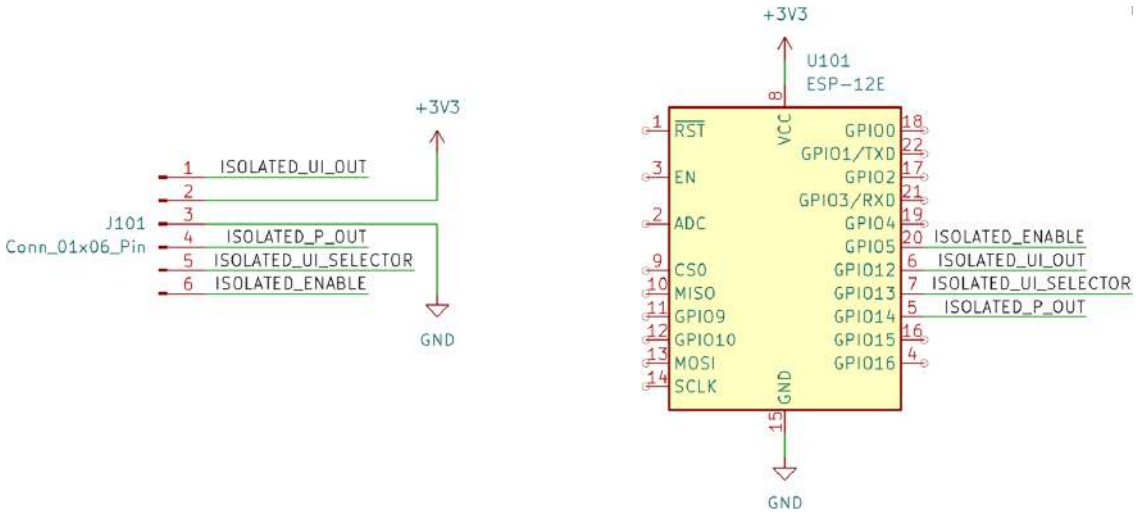

Fig. B1: Esquemático desarrollado, de bajo nivel, del Módulo Inteligente.

B.2 Módulo Medidor

Este módulo está basado en el circuito integrado HLW8012. En este caso, se usó un producto ya fabricado, cuyo esquemático (extraído del *datasheet*) se puede observar en la figura B2.

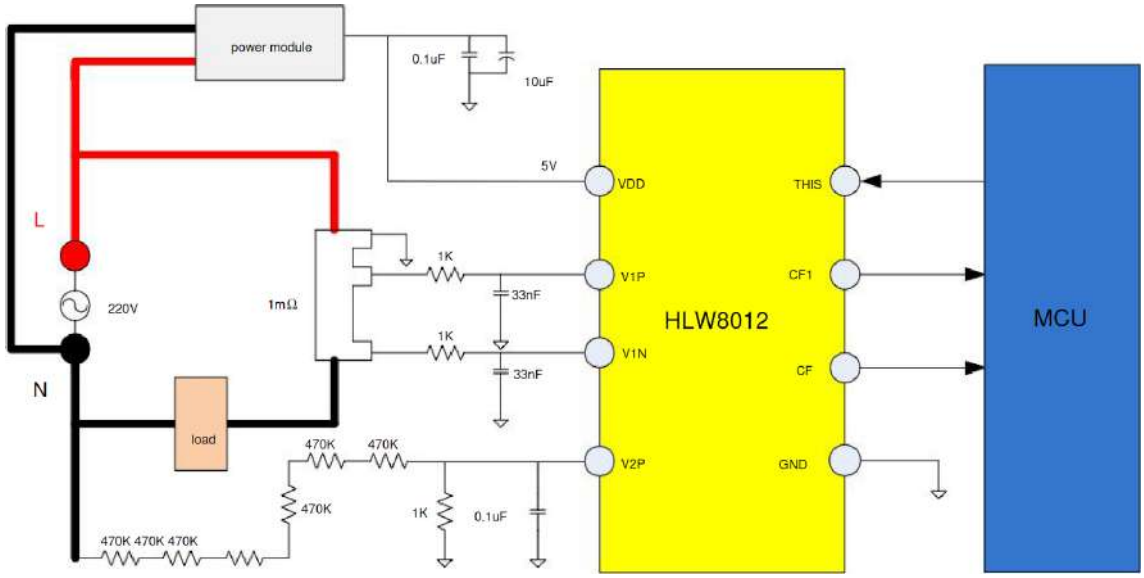

Fig. B2: Esquemático extraído, de bajo nivel, del Módulo Medidor y su posible conexión a la carga a medir (“load”) y a un microcontrolador (MCU).

B.3 Módulo Aislante

Este módulo funciona a través de optoacopladores 4N25 [B1], conectándose entre el módulo inteligente y el medidor.

Para no duplicar el esquemático entre los distintos canales de comunicación aislados unidireccionales, se creó un subcircuito `Canal Aislado.kicad_sch`, donde:

- `VCC_ENT` y `GND_ENT` son los polos positivo y negativo (respectivamente) de la fuente que alimenta al circuito de entrada.
- De la misma forma, `VCC_SAL` y `GND_SAL` corresponden al circuito de salida.
- La señal de `ENTRADA` “controla” a la de `SALIDA`, lo cual puede observarse en la tabla B1.

Tensión en ENTRADA	Tensión en SALIDA	Estado de los LEDs
GND_ENT	VCC_SAL	Apagados
VCC_ENT	GND_SAL	Encendidos

Tabla B1: Relación entre la señal de ENTRADA y SALIDA.

En la figura B3 se muestra el esquemático de un canal de comunicación unidireccional:

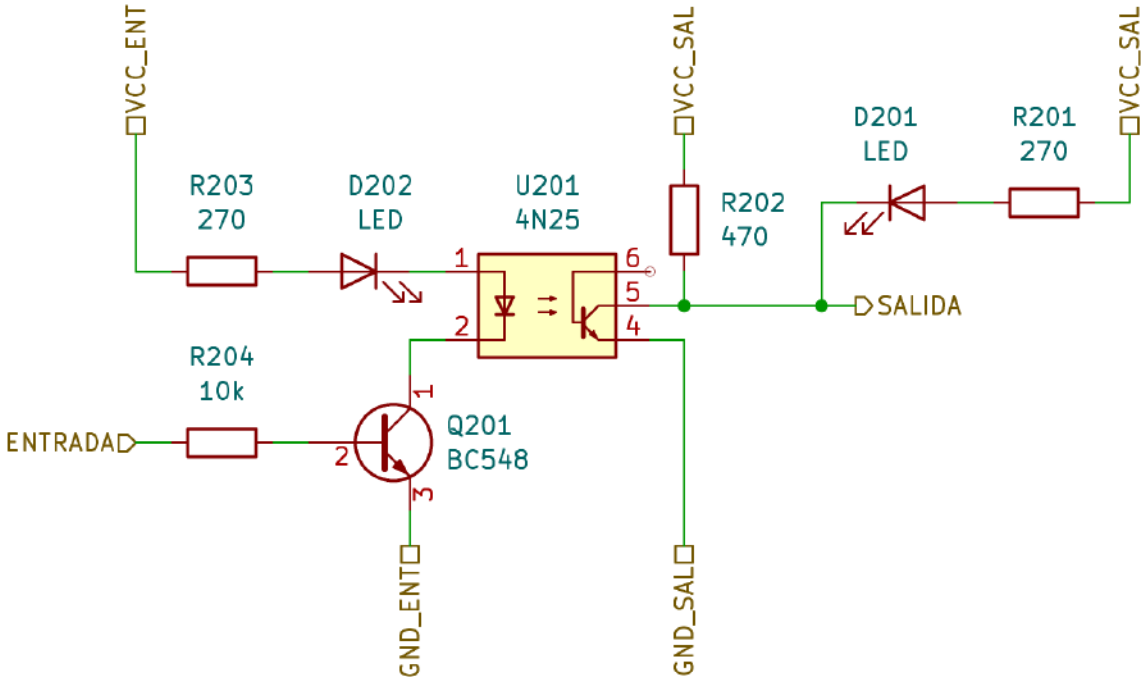

Fig. B3: Esquemático desarrollado, de bajo nivel, de un Canal Aislado Unidireccional.

Este subcircuito se instanció tres veces, lo cual se puede observar en la figura B4.

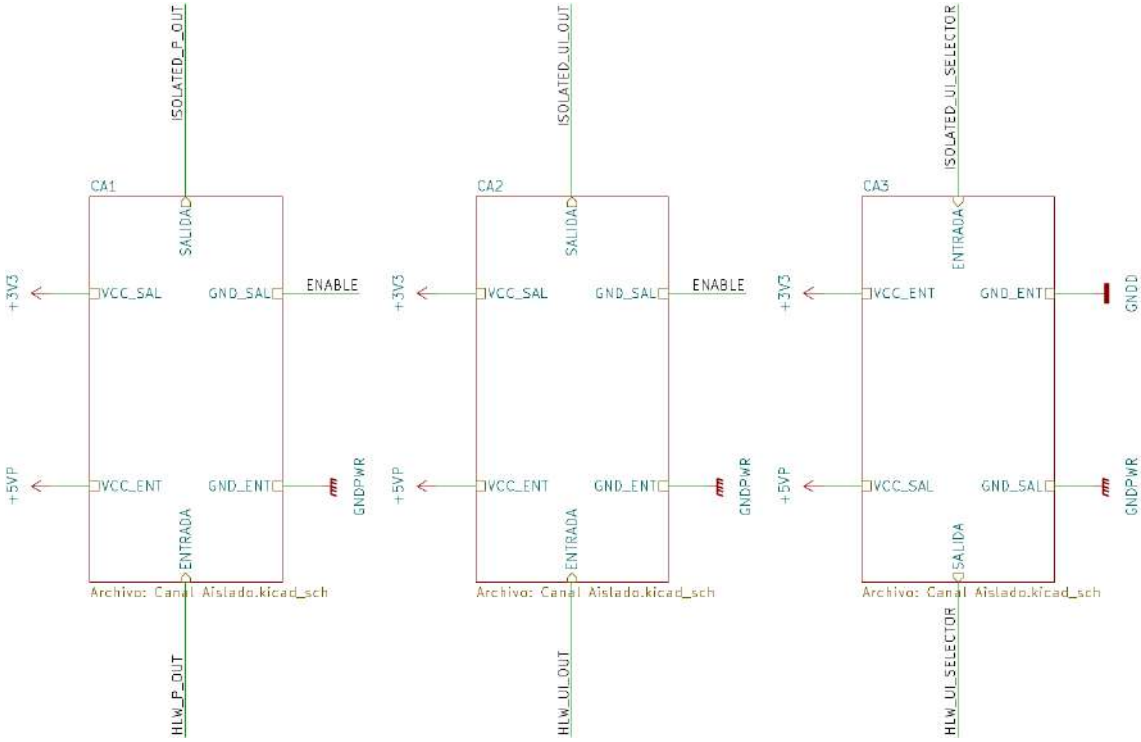

Fig. B4: Esquemático desarrollado, de medio nivel, que ilustra la instanciación de los canales.

El esquemático restante se puede observar en la figura B5.

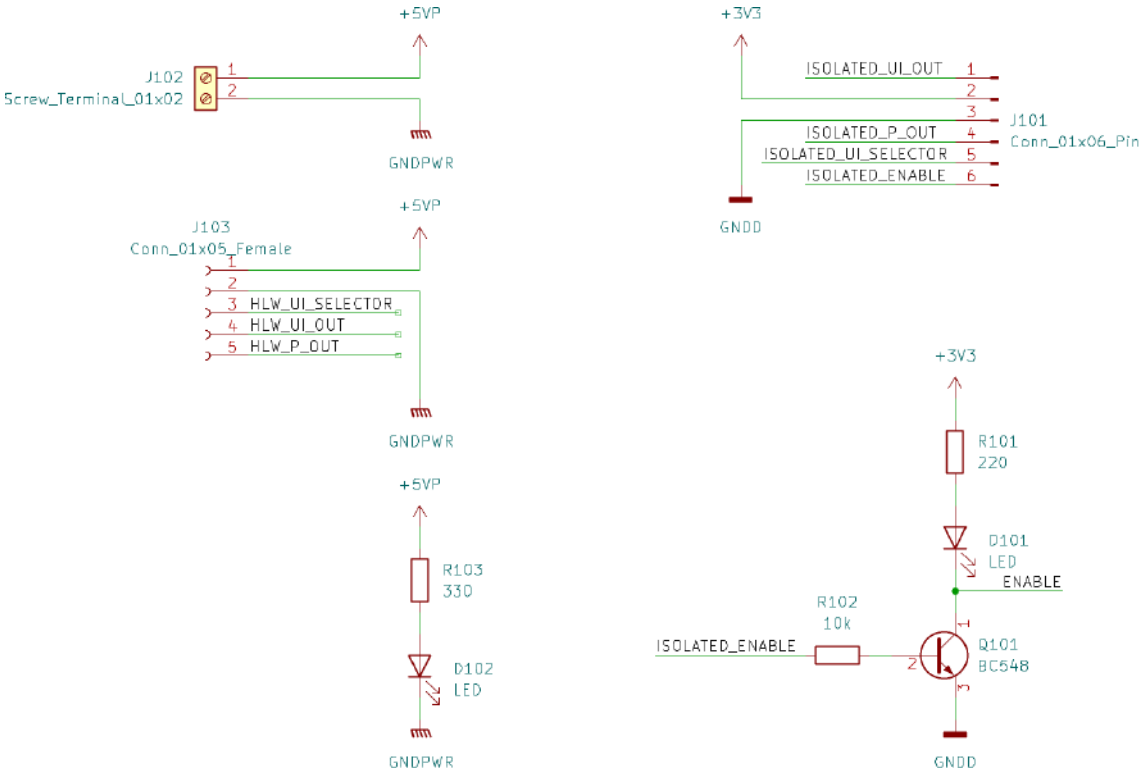

Fig. B5: Esquemático desarrollado, de bajo nivel, de los componentes restantes del Módulo Aislante.

B.4 Módulo de Potencia

Este módulo está formado por dos subcircuitos, cada uno alimentado por un cable independiente del otro. Esto maximiza la seguridad de uso, ya que al conectar sólamente el cable gris, permite verificar que los módulos anteriores funcionen correctamente, sin riesgos de electrocución.

B.4.1 Alimentación de los Módulos

Los componentes son:

1. El cable de alimentación de este módulo (de color gris), que permite suministrar hasta $6\,[Amperes]$ a $220\,[Volts]$ de Corriente Alterna de

$50\ [Hertz]$, similar a la tensión domiciliaria monofásica estándar de la República Argentina.

2. El transformador de tensión, que permite suministrar hasta $0,5\ [Amperes]$ a $12\ [Volts]$ de Corriente Alterna de $50\ [Hertz]$.

3. La fuente de alimentación del Módulo Medidor, capaz de entregar hasta $0,5\ [Amperes]$ a $5\ [Volts]$ de Corriente Continua.

Su esquemático se puede observar en la figura B6.

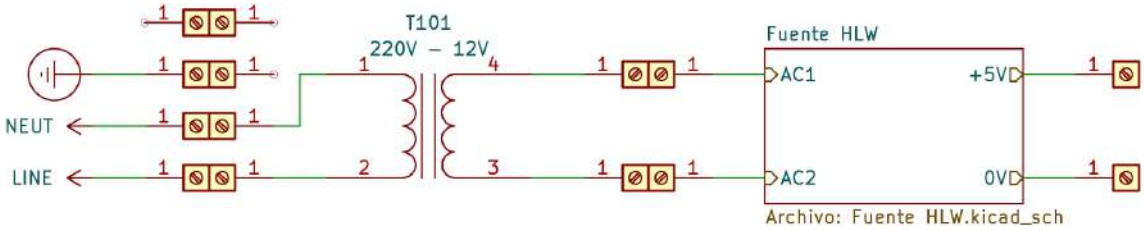

Fig. B6: Esquemático desarrollado, de medio nivel, de los componentes del subcircuito de alimentación de los módulos.

A su vez, el esquemático de la fuente de alimentación del Módulo Medidor (y parte del Aislante) se puede observar en la figura B7.

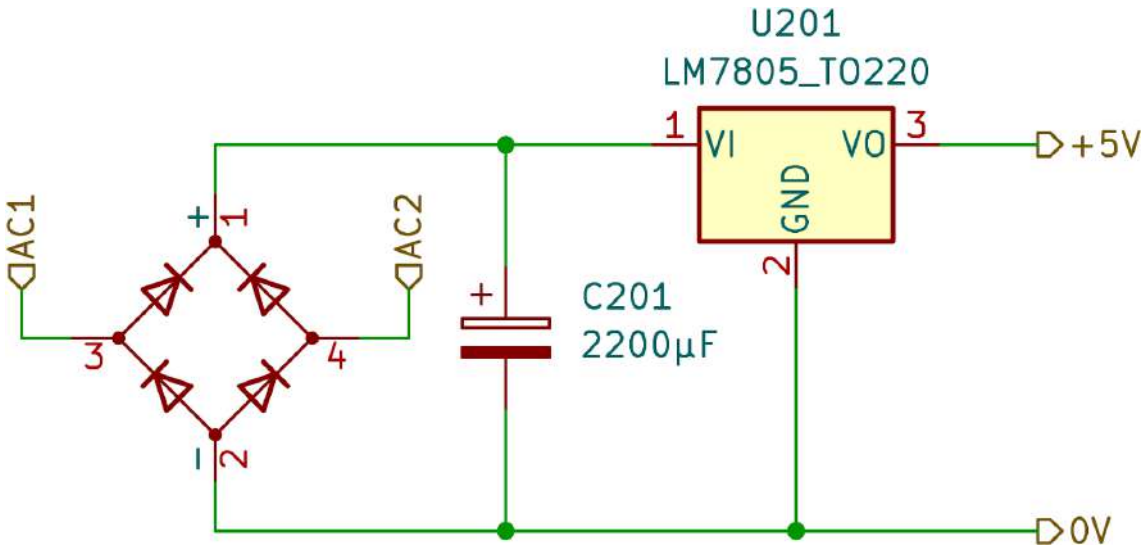

Fig. B7: Esquemático desarrollado, de bajo nivel, de la fuente de alimentación de 5 volts aislada.

B.4.2 Alimentación de la Carga

Los componentes son:

1. El cable de alimentación de este módulo (de color blanco), que permite suministrar hasta $10\ [Amperes]$ a $220\ [Volts]$ de Corriente Alterna de $50\ [Hertz]$.
2. Las borneras del Módulo Medidor.
3. El tomacorrientes en el que se conectará la carga (o circuito eléctrico) a medir, compatible con distintos zócalos de conexión.
4. El portafusibles (con fusible de $10\ [Amperes]$), las borneras y los cables que interconectan los subcircuitos y módulos anteriores.

El esquemático de este subcircuito se puede observar en la figura B8.

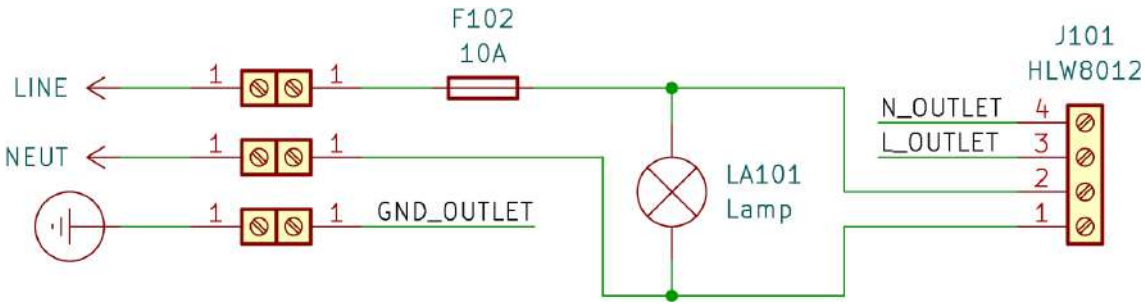

Fig. B8: Esquemático desarrollado, de medio nivel, de los componentes del subcircuito de alimentación de la carga.

B.5 Bibliografía del Anexo B

[B1] Hoja técnica del optoacoplador 4N25 versión 1.8, Vishay Intertechnology. Año 2010.