Universidad Zaragoza
1542

Trabajo Fin de Grado

Sistema Cloud de monitorización de energía basado en un microcontrolador de doble núcleo.

Cloud Energy Monitoring System based on a dual core microcontroller.

Autor/es

Ana García Maestre

Director/es

Natalia Ayuso Escuer
Enrique Torres Moreno

Escuela de Ingeniería y Arquitectura EINA / Universidad de Zaragoza 2022

Resumen.

En los últimos años, debido a la crisis ambiental y económica a la que se enfrenta el planeta, el hecho de crear un mundo sostenible es uno de los principales retos internacionales. Para lograrlo, se debe compaginar tanto la investigación de nuevas fuentes de energía como la optimización de las ya existentes con el uso responsable y eficiente de las mismas. Para ello, es primordial tomar medidas del consumo de energía de la manera lo más precisa posible.

En el ámbito doméstico, los medidores permiten la monitorización de la energía eléctrica consumida por los aparatos eléctricos conectados a la red eléctrica de la vivienda, facilitando al usuario conocer, controlar y contrastar dicho consumo con el indicado en la factura de la luz.

El propósito de este trabajo es desarrollar un sistema monitorizador sencillo y de bajo coste capaz de medir y enviar datos del consumo eléctrico a la nube para su posterior visualización. El sistema propuesto se realiza desde cero diseñando tanto la parte hardware como software. Para ello, utilizando un microcontrolador (ESP8266) se han generado ondas sinusoidales de tensión y corriente para poder emular una carga presente en una vivienda sin la necesidad de tener dicha carga físicamente. Además de evitar riegos debidos a trabajar con valores de red, permite controlar las señales a medir para la caracterización del sistema diseñado.

El cálculo del consumo eléctrico y el envío de datos a la nube se ha realizado con un microcontrolador de doble núcleo (ESP32). Esto permite que ambas tareas puedan realizarse en paralelo. Además, el uso del sistema operativo FreeRTOS permite gestionar dichas tareas utilizando uno de los núcleos para los cálculos y el otro, para todo lo relacionado con las comunicaciones WiFi.

Por otro lado, con la ayuda de la librería *Emonlib* se calcula el consumo eléctrico con las diversas ondas generadas y se realiza el estudio de precisión mediante el programa *Matlab*. Una vez obtenidos los datos relativos al consumo se envían por el interfaz WiFi a la plataforma *Emoncms* para su posterior visualización.

Con todo ello, se tiene un prototipo final de bajo coste con el que el usuario puede variar las ondas de tensión y corriente para conocer la energía eléctrica que se estaría consumiendo en una situación determinada. De este modo, se puede planificar el consumo de la vivienda y maximizar el rendimiento de la instalación beneficiando al usuario con un ahorro económico en la factura eléctrica.

Índice

1. Introducción.

1.1 Consumo eléctrico y eficiencia energética.

En las últimas décadas se ha producido un desarrollo tecnológico, económico y social de manera global, aspecto que acarrea un aumento del consumo energético mundial (Figura 1). Paralelamente a este hecho, ha surgido el concepto de modelo energético sostenible, basado en el equilibrio entre la producción, el consumo y este desarrollo económico y social. Consta de tres aspectos fundamentales: 1) Seguridad energética: que garantice el suministro a precios razonables; 2) Competitividad: no debe suponer un peligro para la economía; 3) Sostenibilidad ambiental: no debe suponer un impacto perjudicial para el entorno.

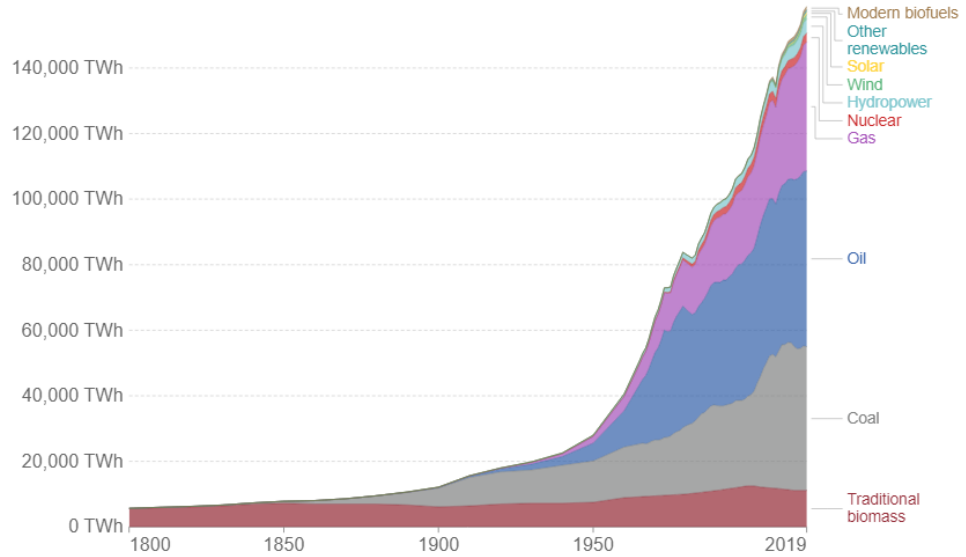


Figura 1: Aumento del consumo global de la energía [1].

Desde finales del siglo XVIII, con la revolución industrial, la principal fuente de emisiones de dióxido de carbono (CO_2) proviene de la combustión de petróleo y derivados en las centrales eléctricas, de los automóviles y del sector industrial. El aumento de las emisiones de dichos gases ha acarreado consecuencias nefastas para el planeta como, por ejemplo, el efecto invernadero que encabeza las causas de la crisis climática mundial que se vive actualmente. Frente a este grave problema, se han establecido acuerdos internacionales como el protocolo de Kyoto, de Montreal, la Cumbre de la Tierra o el Acuerdo de París. Los fundamentos de dichos acuerdos se basan en el aumento de la capacidad de generación de energía a partir de fuentes y tecnologías renovables además de dar un uso responsable a la energía utilizada, también denominada eficiencia energética [2].

Cabría destacar que, en la actualidad, hay una gran escasez de materiales que no permite la deseada transición energética y además solamente el 24% de la producción mundial de electricidad representa a energías renovables. Aun así, gracias a los avances en fuentes de energías limpias, se prevé que, en el 2040, el 40% de la demanda sea abastecida a través de estas energías. Aun así, el modelo energético futuro, no se puede basar solamente en estas fuentes renovables [3] [4]. Es por esto por lo que la Unión Europea plantea unos objetivos sostenibles en el sector energético: 1) Reducción de las emisiones de gases de efecto invernadero en un 20%. 2) Aumento del 20% en las fuentes renovables. 3) Mejorar la eficiencia energética en un 20% [5] [6].

Todo lo planteado anteriormente y el exponencial crecimiento del precio de la electricidad, deriva a la necesidad de desarrollar sistemas que permitan mejorar la eficiencia energética y reducir su consumo, tanto a nivel industrial como a nivel doméstico [7]. Por lo tanto, es primordial que se desarrollen dispositivos capaces de contribuir con estos objetivos, capaces de monitorizar el uso de la energía mediante medidores del consumo eléctrico. Solamente si se conoce cuánta y dónde se utiliza realmente la energía, será posible plantear soluciones para reducir el uso y los consecuentes costes.

1.2 Medidores de consumo eléctrico comerciales para uso doméstico: descripción y tipos.

Un medidor de consumo eléctrico doméstico es un aparato digital o analógico que informa periódicamente del consumo en tiempo real de la instalación eléctrica pudiendo calcular el consumo diario, semanal y mensual. Este dispositivo permite no solo tratar de reducir el gasto de energía o el impacto medioambiental sino ahorrar además en la factura eléctrica. Éste se instala en el momento en el que el cliente da de alta la electricidad en la vivienda y se dividen en dos grandes grupos:

- Individual: este tipo de medidor se conecta a un enchufe determinado y es capaz de calcular el consumo eléctrico generado en aquellos aparatos conectados a éste.
- Global: recopila datos de toda la casa. Se coloca en el cuadro eléctrico y hace la función de monitor para controlar el gasto en kWh.

El contador de luz digital o inteligente es muy distinto al contador de luz analógico, aunque ambos sirvan para medir el consumo eléctrico. El contador analógico no puede cuantificar el precio del kilovatio hora, sino que solo cuantifica la cantidad de kWh consumidos en una vivienda sin diferenciar el horario en el que se realizaba el gasto. En la actualidad, las empresas eléctricas cambiaron los contadores eléctricos analógicos por digitales, ya que así lo puso la ley, RD 1110/2007 (Plan de Sustitución de Contadores), que establecía fecha límite hasta el 31 de diciembre de 2018 [8]. En la siguiente tabla (Tabla 1) se presentan las diferencias entre contadores digitales y analógicos.

Contador Digital	Contador Analógico
Funcionamiento: utilizan convertidores analógico-digitales para hace la conversión. Se puede comprobar mediante un sistema de luces: apagada (no se está dando consumo eléctrico en la vivienda), parpadea ligeramente (ha saltado la electricidad) o encendida y fija (la potencia eléctrica contratada ha sido superada en la vivienda o local)	**Funcionamiento:** se basa en la creación de campos magnéticos mediante el uso de dos juegos de bobinas que actúan directamente sobre el disco, el cual produce una corriente al girar. Cada giro del disco es proporcional a la potencia consumida por el circuito.
Tiene una pantalla digital donde se muestra el número de kWh consumidos.	Tiene discos metálicos que giran los números para mostrar el número de kWh consumidos.
Permite realizar las lecturas en remoto. No es necesaria la visita de un técnico para realizar la lectura del contador.	Para realizar la lectura del contador se necesita la presencia de un técnico o del propio usuario.
Puede realizar lectura real del precio de kWh.	El consumo que ofrece es estimado.
Admite la tarifa PVPC[1] y tarifas con discriminación horaria.	Solo admite la modalidad de la PVPC. No admite tarifas con discriminación horaria.
Puede estar instalado en el interior de la vivienda.	Estará instalado siempre en el cuarto de contadores.
 *Medidor digital monofásico ENEL Endesa CERM1.*	 *Medidor analógico monofásico BeMatik.*

Tabla 1: Comparación de las características del contador digital (izquierda) y el contador analógico (derecha) [9] .

[1] PVPC: Precio Voluntario para el Pequeño Consumidor.

Como se ha comentado anteriormente los medidores digitales son capaces de recoger con exactitud el consumo de electricidad (kWh) en tiempo real y enviarlo a la distribuidora eléctrica. Éstos incorporan una pantalla digital que permite controlar el equipo de medida. En la esquina superior derecha de la pantalla se registras los kWh consumidos en cada periodo (Figura 2). En la izquierda aparece el periodo de integración horaria siendo:

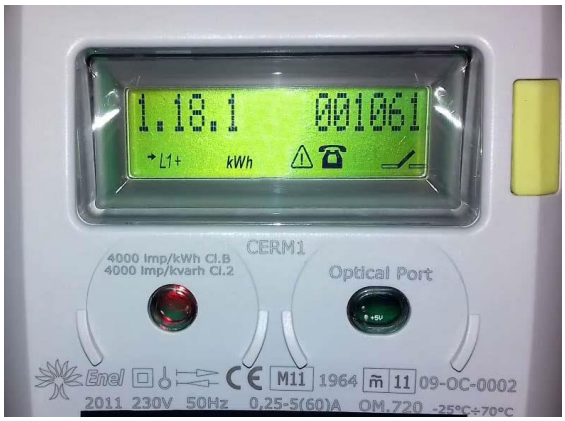


Figura 2: Pantalla de un medidor digital [10].

✓ 1.18.0: energía total consumida.

✓ 1.18.1: lectura de contador en horas punta.

✓ 1.18.2: lectura de contador en horas valle.

✓ 1.18.3: lectura de contado en horas supervalle.

De esta manera es posible obtener una representación gráfica del consumo eléctrico diario de una vivienda (Figura 3) para poder tener un control exacto y eficaz que permita a la vivienda ser sostenible tanto energética como económicamente.

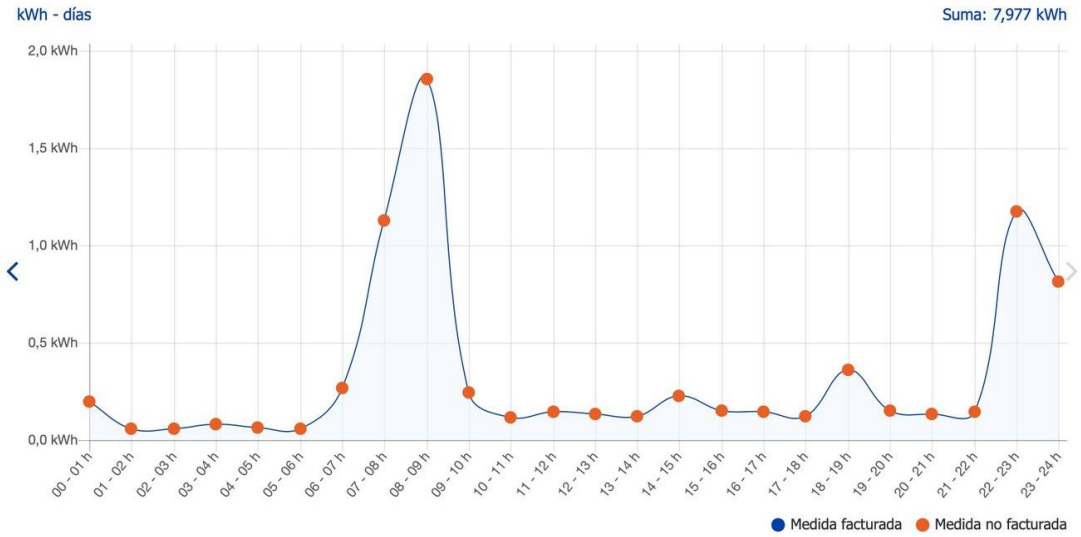


Figura 3: Representación del consumo eléctrico diario de una vivienda. [11]

1.3 Objetivos del trabajo fin de grado.

La elaboración de este proyecto surge del interés de hacer un uso eficiente de la energía y así desarrollar un medidor del consumo eléctrico para uso doméstico de bajo coste y desarrollado con filosofía Open Software. De esta manera se trata de contribuir a los siguientes objetivos de desarrollo sostenible (ODS) [12]:

- 7.1: de aquí a 2030, garantizar el acceso universal a los servicios energéticos asequibles, fiables y modernos.
- 7.3: de aquí a 2030, duplicar la tasa mundial de mejora de la eficiencia energética.
- 12.2: de aquí a 2030, lograr la gestión sostenible y el uso eficiente de los recursos naturales.

Cabría explicar que el presente trabajo se ha llevado a cabo en el Área de Arquitectura y Tecnología de Computadores del departamento de Informática e Ingeniería de Sistemas de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza donde sus directores tienen una línea de investigación en Internet of Things (IoT) para la consecución de los ODS.

Así, se ha diseñado y configurado un prototipo de medidor sencillo, de tamaño pequeño y de bajo coste, que permitiría monitorizar el consumo en tiempo real de las partes de una instalación eléctrica y/o de los dispositivos conectados a la red eléctrica en el ámbito doméstico. Para ello, utilizando un microcontrolador se han generado ondas sinusoidales de tensión y corriente para poder emular una carga presente en una vivienda sin la necesidad de tener dicha carga físicamente. Además, se evita el trabajar con valores de red.

Una vez generadas las ondas de tensión y corriente, se obtienen los parámetros relacionados con la energía y se estudia la precisión de dichas medidas. Además, el dispositivo está dotado de WiFi por lo que cabe la posibilidad de conectarlo y almacenar sus datos a través de Internet.

Partiendo de cero, se ha podido desarrollar un dispositivo que permitiría simular las diferentes cargas domésticas y a su vez, controlar de manera eficaz el consumo de la instalación de una vivienda. En el marco de este Trabajo de Fin de Grado se ha analizado el nivel de precisión de los valores necesarios para dicho cálculo y, de esta manera, ayudar a mejorar el consumo de la vivienda. Todo esto ha sido posible gracias al uso de tecnologías disponibles en el laboratorio, plataformas gratuitas (*OpenEnergyMonitor: Emoncms*) y software de código abierto (*Arduino IDE*, *Matlab* y la librería *EmonLib*).

1.4 Estructura de la memoria.

La memoria se ha estructurado de la siguiente manera:

- Comienza en la memoria, capítulo en el que nos encontramos, con una introducción al campo de la energía, los problemas que se plantean actualmente y las posibles soluciones, entre ellas los medidores del consumo eléctrico.
- En el capítulo 2, se describe el diseño y montaje del dispositivo medidor mediante el cual se obtendrá la señal y los valores de consumo.
- El capítulo 3 se puede dividir en dos bloques: 1) La síntesis de las señales de tensión y de corriente mediante el software, en *Arduino*, explicando el código desarrollado para obtener la señal PWM generada por la tarjeta ESP8266. A continuación, se muestra el filtro de paso bajo que se utiliza y el seguidor de tensión a través del cual se obtienen las ondas finales. 2) Cálculo de los parámetros deseados para la obtención del consumo eléctrico, reproduciendo las señales generadas por el microcontrolador ESP8266, pero esta vez mediante código en el programa *Matlab* e imponiendo valores reales de la red eléctrica.
- El capítulo 4 también se pueden diferenciar tres bloques: 1) Obtención de los parámetros deseados para el cálculo del consumo eléctrico (potencia activa y reactiva, tensión y corriente eficaz y el factor de potencia) utilizando la librería *Emonlib.* La tarjeta ESP32 es la responsable del tratamiento de las señales de tensión y corriente y, por lo tanto, de dicho cálculo con *Emonlib.* 2) Cálculo del error entre los parámetros obtenido a través de la librería *Emonlib* (valores experimentales) y las medidas que representan una instalación real de una vivienda calculadas en *Matlab.* 3) Por último, los parámetros calculados por *Emonlib* indicandos anteriormente (potencia activa, potencia reactiva, tensión y

corriente eficaz y factor de potencia) y obtenido el valor del consumo eléctrico, se envían a la plataforma *OpenEnergyMonitor: Emoncms*, en la web.

- Se finaliza con las conclusiones y se plantean las líneas futuras de este proyecto en el capítulo 5.
- En el anexo I: Tipos de medidores en el mercado. Compara distintos tipos de medidores eléctricos presentes en el mercado. Además, se describe con detalle el dispositivo *Efergy Elite Classic* ya que representa en la realidad, el prototipo al que se asemeja la idea de este proyecto.
- En el anexo II: Microcontroladores utilizados. Aquí se indican muy brevemente las características principales de los microcontroladores usados para este proyecto a partir de sus *Datasheets* y finalmente, se comparan ambos chips.
- En el anexo III: Herramientas de software utilizadas. Consta de tres partes dedicadas a herramientas de Software que se han usado durante el proyecto: *Arduino IDE*, *FreeRTOS* y la librería *EmonLib* responsable de los cálculos de potencia.
- En el anexo IV: Breve explicación de la plataforma *Open Energy Monitor*: *Emoncms* donde se envían los datos para ser visualizados.
- En el anexo V: Se presentan las tablas Excel con los valores calculados con la ayuda de la librería *Emonlib* y cálculo de los errores.
- En el anexo VI: Muestra una tabla con el desglose del coste de cada componente utilizado en el sistema.
- Finalmente, en el anexo VII: Se expone la dedicación en horas de las distintas tareas del proyecto.

2. Elementos del Sistema Cloud de monitorización de energía.

2.1 Introducción.

Como se ha indicado anteriormente, el objetivo de este proyecto es desarrollar desde cero un prototipo de medidor de consumo eléctrico a partir de la toma de valores de tensión y corriente de la red eléctrica de una vivienda. Los contadores que se encuentran actualmente en el mercado miden dichos valores utilizando diferentes tipos de sensores de tensión y corriente, y quedarían instalados en el cuadro eléctrico de la vivienda. Dichos contadores quedan explicados en el anexo I. En el presente proyecto, se simulará la tensión y corriente de la red eléctrica mediante el uso de un microcontrolador. De esta manera, se puede modificar dichas ondas para emular distintas cargas presentes en una vivienda.

2.2 Arquitectura del sistema.

En este apartado, se muestra un esquema que representa el sistema diseñado (Figura 4) y una explicación de cada una de las partes y procesos que se llevan a cabo.

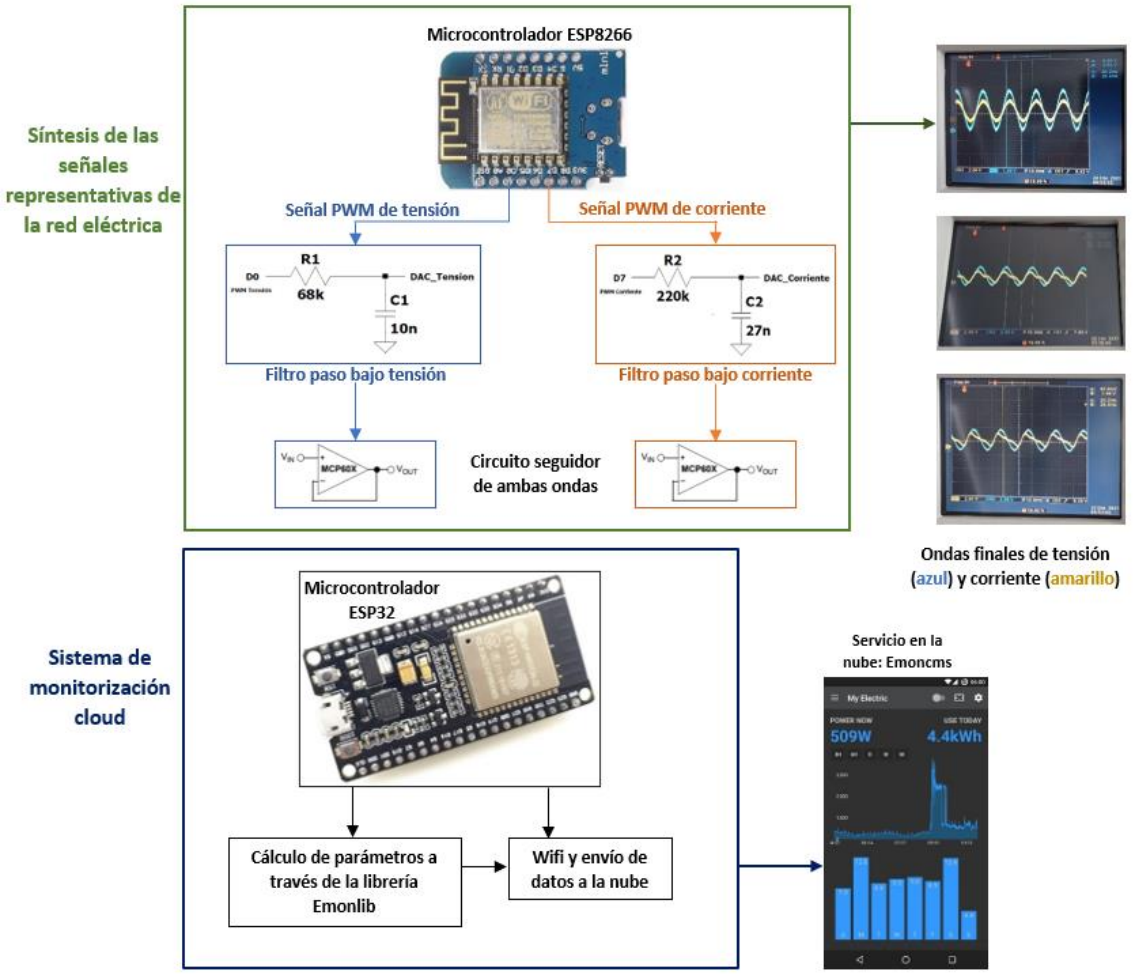


Figura 4: Esquema del sistema desarrollado.

Para la obtención de los valores de consumo eléctrico y su posterior almacenamiento en la nube, se parte de un microcontrolador ESP8266 ya que se ha decidido implementar las ondas representativas de la red eléctrica para así evitar trabajar con valores de red de tensión y corriente y de esta manera evitar riesgos humanos. Además, permite modificar las señales y así simular el funcionamiento de las distintas cargas presentes en el hogar. A continuación, se

diseña un código en el programa *Arduino IDE* para generar una señal PWM (pulse-width modulation/ modulación por ancho de pulsos) de tensión y corriente ya que el microcontrolador no dispone de un convertidor digital-analógico.

A continuación, las señales generadas pasan por un filtro paso bajo de primer orden obteniendo las ondas senoidales, propia de la red eléctrica española.

Para adaptar las ondas de salida del microcontrolador ESP8266 con otros dispositivos hardware del sistema, se utiliza un circuito seguidor de tensión. Éstas serán la entrada al microcontrolador ESP32 donde se da el tratamiento y el cálculo de parámetros necesarios para la obtención consumo eléctrico a través de la librería *Emonlib*. Dichos parámetros quedan reflejados en la sección 3.3 de esta memoria. Además de lo expuesto anteriormente, el ESP32 también es responsable de la conexión WiFi y del envío de datos a la nube (*Open Energy Monitor*: *Emoncms*) donde se pueden visualizar en gráficas.

Por último, se realiza un análisis de precisión de los datos calculando el error entre las medidas reales (obtenidas mediante el sistema y la librería *Emonlib*) y las teóricas (calculadas mediante *Matlab*), donde se han generado los valores teóricos mediante la implementación de código. Se han impuesto valores de tensión y corriente eficaz obteniendo el consumo eléctrico y, de esta manera, se comprueba que los valores obtenidos mediante el sistema diseñado son correctos.

2.3 Hardware: Descripción componentes.

El sistema en el que se basa este trabajo fin de grado está construido en una placa prototipo *Protoboard* (Figura 5).

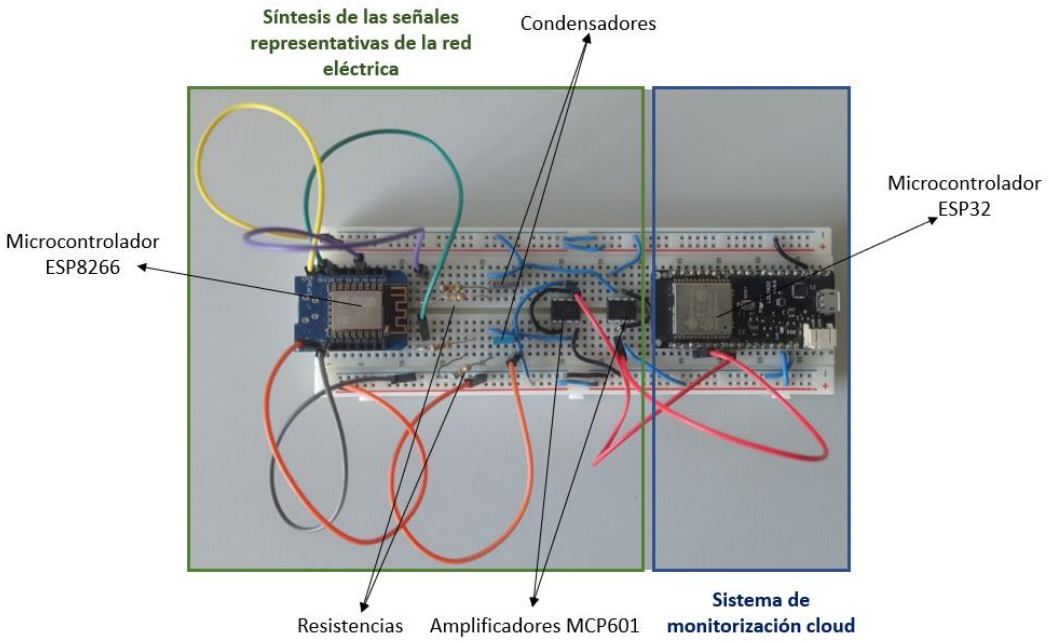


Figura 5: Foto del sistema, tomada en el laboratorio.

Los componentes principales son:

- Tarjeta ESP8266: microcontrolador responsable de la implementación de las señales PWM tanto de tensión como de corriente y su posterior modificación mediante la

instrucción de armónicos en la señal de corriente. Las características fundamentales de esta tarjeta están descritas en el anexo II.
- Resistencias y condensadores: se combinan estos dos componentes para realizar un filtro de paso bajo.
- Amplificadores: en este caso se ha utilizado el amplificador MCP601 para hacer un circuito seguidor y adaptar las ondas finales de salida del microcontrolador ESP8266 con los pines de entrada del chip ESP32 para el tratamiento de datos y cálculo del consumo eléctrico.
- Tarjeta ESP32: se ha elegido este microcontrolador ya que cuenta con WiFi y es compatible con la librería Emonlib encargada del cálculo de la energía eléctrica. Además, se realiza también el envío de datos a la nube, a la plataforma *Emoncms*. Las características de interés están enumeradas también en el anexo IV. Se ha elegido esta tarjeta ya que tiene dos núcleos, de esta manera se pueden ejecutar diversas tareas en paralelo.
- Cables: utilizados para la unión entre todos los compontes electrónicos, la alimentación y la línea de tierra.
- Protoboard: para la disposición de los elementos descritos.

El coste aproximado del sistema implementado asciende a 19,7 €. En el anexo VII, se puede ver desglosado por componentes.

2.4 Herramientas Software utilizadas.

En este proyecto, se han utilizado los programas *Matlab* y *Arduino IDE* y la librería *Emonlib*. Además, se ha realizado el estudio del sistema operativo *FreeRTOS.* En el anexo III se expone de forma más detallada las características principales de cada programa y la explicación completa del código de la librería utilizada.

1) *Emonlib*: librería utilizada para la obtención de los parámetros necesarios para el cálculo del consumo eléctrico: tensión eficaz, corriente eficaz, potencia activa, potencia aparente y factor de potencia. El código completo de dicha librería aparece reflejado con más detalle en el anexo III.
2) *Arduino IDE*: multiplataforma utilizada para la implementación de las ondas de tensión y corriente, la lectura de dichas señales y el tratamiento de dichas ondas. Por otro lado, se realiza el envío de datos obtenidos mediante *Emonlib* a la nube.
3) *Matlab*: plataforma de programación y cálculo numérico con un código propio mediante el cual se ha calculado los parámetros necesarios para la obtención del consumo eléctrico.
4) *FreeRTOS*: es un sistema operativo de tiempo real multitarea que permite administrar los recursos hardware y los tiempos de ejecución de las tareas implementadas en un microcontrolador. De este modo, permite ejecutar varias tareas de forma simultánea además de multitud de servicios en segundo plano, haciendo que parezca que todo se ejecuta al mismo tiempo [13]. En el apartado 4.2 se muestra dónde se ha aplicado *FreeRTOS* en este proyecto.

2.5 Servicio en la nube seleccionado.

2.5.1 Aspectos generales.

Este trabajo fin de grado se basa en el proyecto *Open Energy Monitor* [14], sistema que tiene la capacidad del monitorizar el uso de energía eléctrica en la vivienda y así controlar el consumo de energía. Además, es posible subir datos a la nube a través de una aplicación web

que se ejecuta en *Emoncms*. De esta manera, el usuario puede ver el consumo de energía en tiempo real, en kWh.

2.5.2 Emoncms.

Emoncms es una aplicación web de código abierto, utilizada el proyecto *Open Energy Monitor,* capaz de procesar y permitir la visualización de parámetros como la energía o la temperatura entre otros. Se ha elegido esta plataforma (reflejada con mayor detalle en el anexo III) ya que se pueden enviar datos en tiempo real o fijando intervalos temporales que varían desde los diez segundos a envío diario de valores. En la Figura 6 se observa la pantalla de registro para acceder a esta aplicación [15].

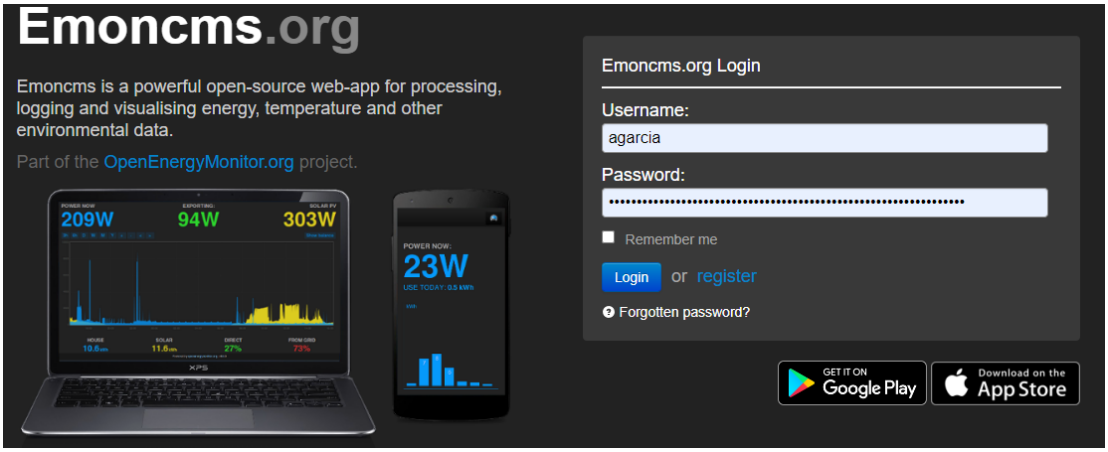


Figura 6: Aplicación web Emoncms.org [17]

3. Implementación del sistema.

3.1 Síntesis de las ondas de tensión y corriente.

Se parte de señales tanto de tensión como de corriente que representen, de la manera más realista posible, la red eléctrica. Por lo tanto, los valores reales de tensión y corriente difieren de la onda sinusoidal ya que pueden incluir armónicos y ruido. Además, en el caso de la red eléctrica se debe tener en cuenta que debe alimentar un gran número de cargas que alteran la forma de onda de la corriente que consumen. La consecuencia de los consumos no senoidales es que la tensión sufre también una cierta distorsión debido a las caídas de tensión en las impedancias y transformadores.

En la Figura 7 se muestra el consumo típico de una red monofásica (Figura 7, izquierda) y otra trifásica (Figura 7, derecha). En el caso de la tensión (en azul) se observa una leve distorsión (THD: *total harmonic distortion,* de 1,3%) en la red monofásica y una distorsión mayor (THD: 8,3%) en el ejemplo trifásico que aleja la forma de onda de la señal senoidal.

Por otro lado, las ondas de corriente están formadas por una componente de 50 Hz (frecuencia fundamental de la red) y una serie de armónicos, es decir, componentes de frecuencias múltiplos de ésta. La frecuencia de la red eléctrica real puede desviarse del valor nominal y también puede causar errores adicionales en la medición y las características de potencia. En la Figura 7, la forma de onda de corriente (en rojo) tanto en el caso monofásico como en el trifásico difiere en gran medida de la onda senoidal, consecuencia de las componentes descritas anteriormente.

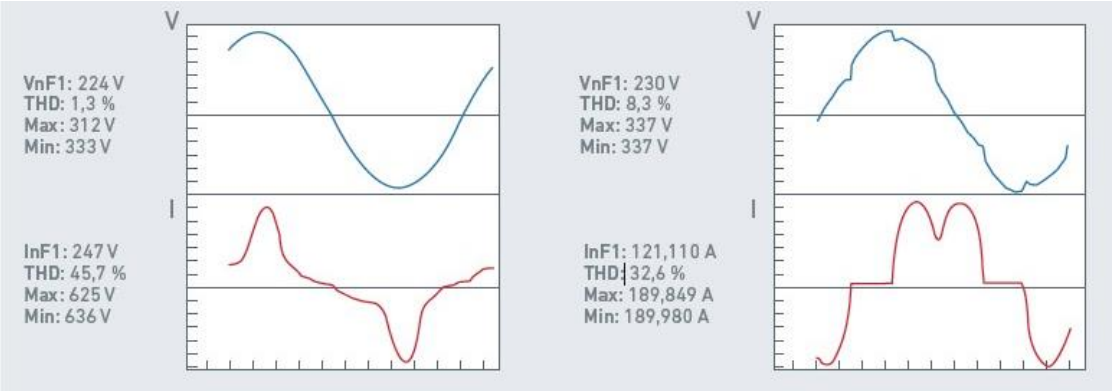


Figura 7: Formas de onda típicas de redes distorsionadas: monofásica (izquierda) y trifásica (derecha) [16].

Para una mayor simplicidad, para este proyecto solo se ha alterado la señal de corriente incluyendo solamente los 3 primeros armónicos (f_0 = 50 Hz, f_1 = 100 Hz y f_2 = 150 Hz). Por lo tanto, se obtendrá una señal de tensión sinusoidal idénticas para los tres casos y tres señales diferentes de corriente.

En la Figura 8, se muestran las distintas gráficas generadas en *Matlab*, mostrando la salida esperada del microcontrolador ESP8266, cuya amplitud varía entre 0 y 3,3 V, valores de tensión mínimo y máximo respectivamente que éste nos puede ofrecer. Se muestran la onda de tensión (azul) y las tres ondas de corriente (rojo) modificando la amplitud y combinando las frecuencias escogidas.

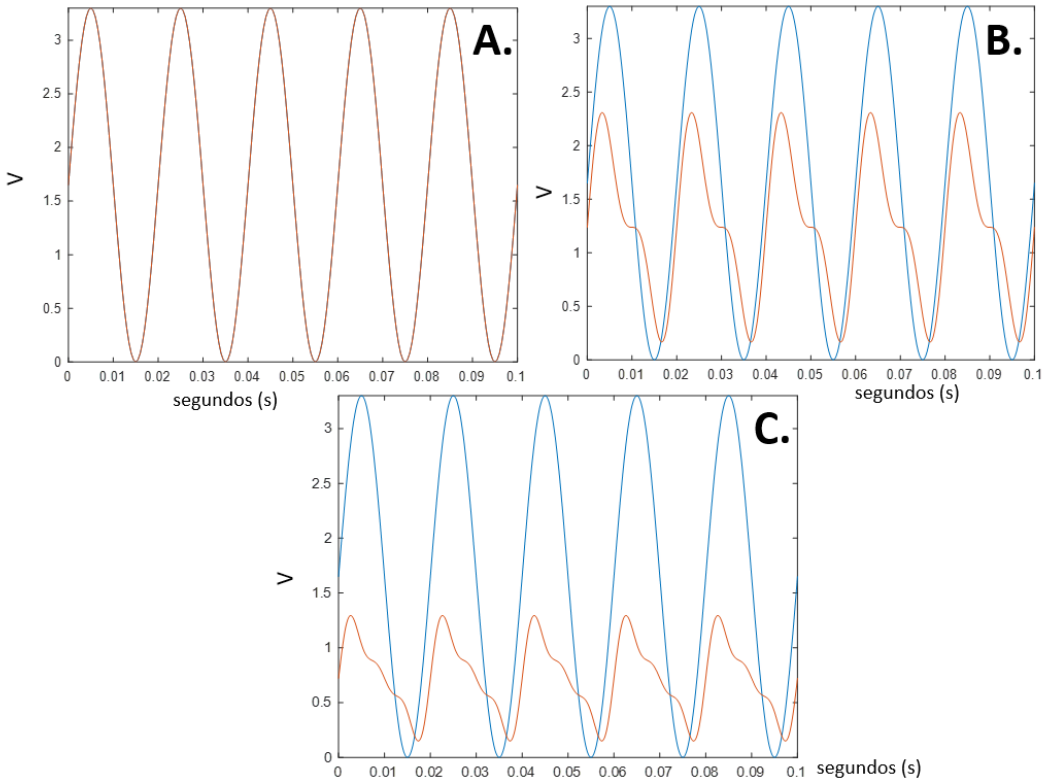


Figura 8: Señales tensión y corriente implementada en Matlab (A.) armónico fundamental, 50 Hz. (B.) Dos armónicos, 100 Hz. (C.) Tres armónicos, 150 Hz.

En la Figura 8.A (armónico fundamental), se observa que la tensión y la corriente coinciden, que era de esperar. En cambio, en la Figura 8.B, correspondiente a la combinación de dos armónicos (f_0 = 50 Hz y f_1 = 100 Hz), la señal de corriente está distorsionada al igual que en el caso de la combinación de tres armónicos (f_0 = 50 Hz, f_1 = 100 Hz y f_2 = 150 Hz), Figura 8.C. Dichas distorsiones son consecuencia de la introducción de las componentes armónicas. Esta forma de ondas son las que se desean obtener en la parte experimental de este proyecto y dichas ondas se observan en la Figura 15. Se puede concluir que las señales generadas por el sistema coinciden con las sintetizadas en *Matlab*.

3.2 Implementación de las ondas de tensión y corriente.

Una vez determinada la forma de onda que se van a generar en el proyecto, se implementan las ondas de tensión y de corriente. Para ello y a diferencia de los contadores comerciales, que utilizan sensores desde el cuadro eléctrico, se generan dichas ondas con el microcontrolador ESP8266 y así evitar el peligro de trabajar con valores dados directamente de la red eléctrica. Dicho microcontrolador no cuenta con un conversor digital analógico para conseguir las dos ondas así que, para ello, se debe trabajar a partir de una señal PWM y un filtro paso bajo.

1) Señal PWM: para transmitir una señal, ya sea analógica o digital, se debe transmitir sin perder potencia o sufrir distorsión. Es por esto por lo que se debe modular utilizando la técnica PWM. Por lo tanto, la señal digital modulará la señal analógica deseada.

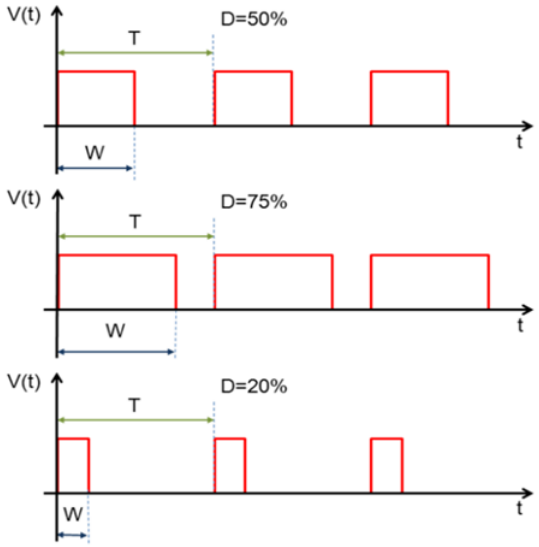


Figura 9: Señal PWM *[17]*

La señal PWM (Figura 9) se obtiene de modificar el ciclo de trabajo de una onda periódica (en este caso sinusoidal). El ciclo de trabajo D se define a través del cociente entre el ancho relativo de su tiempo en alto, W, función positiva y el periodo de la señal T.

$$D = \frac{W}{T}$$

D: Ciclo de trabajo (típicamente definido en porcentaje)

W: Tiempo en que la función es positiva (ancho de pulso).

T: Es el período de la señal.

2) Filtro paso bajo: se utiliza además un filtro de paso bajo que permite el paso de las frecuencias más bajas y atenúa las frecuencias más altas. De esta manera queda la componente continua y los armónicos deseados.

Desarrollando un código en el programa *Matlab*, se ha conseguido crear una señal PWM unipolar partiendo de una señal sinusoidal de referencia y una señal portadora en diente de sierra. Cuando el valor instantáneo de la sinusoide de referencia es mayor que la portadora en diente de sierra, la salida está en el valor máximo 1 (nivel alto) y cuando la referencia es menor que la portadora, la salida está en el valor mínimo 0 (nivel bajo).

$v_0 = 1$ para $v_{seno} > v_{tri}$

$v_0 = 0$ para $v_{seno} < v_{tri}$

Se pueden observar las tres señales en la Figura 10: la señal sinusoidal de referencia (en rojo), la señal triangular (en azul) y la onda PWM unipolar (en verde). Se aprecia que para valores positivos de la señal seno, el ancho del pulso correspondiente, también es mayor que cuando, a medida que la amplitud de la señal senoidal decrece. Por lo tanto, decrece también el ancho de pulso, cambiando así su ciclo de trabajo, evento que se repite en el tiempo.

```
s=sawtooth(2*pi*10*t+pi);
m=0.75*sin(2*pi*1*t);
n=length(s);
for i=1:n
if (m(i)>=s(i))
pwm(i)=1;
elseif (m(i)<=s(i))
pwm(i)=0;
end
end
plot(t,pwm,'-g',t,m,'--r',t,s,'--b');
grid on;
ylabel('Amplitude');
xlabel('Time index');
title('PWM Wave');
step = 100;
t=[0:1/step:1-(1/step)];
A=0.25;
sine=A*sin(2*pi*t)+0.25
```

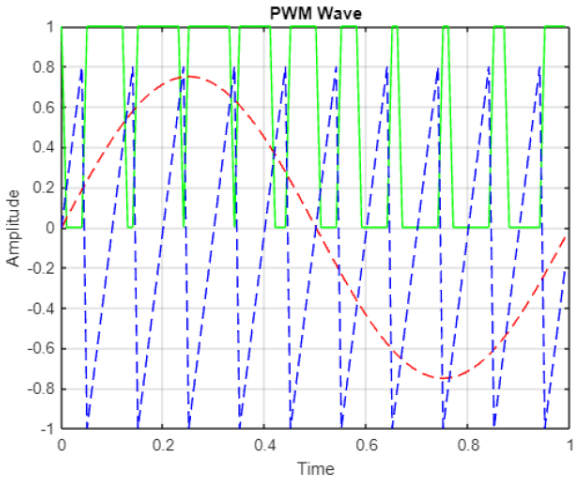


Figura 10: Código en Matlab (izquierda). Onda sinusoidal de la señal de referencia (en rojo), señal portadora triangular (azul) y onda PWM (en verde) (derecha).

3.2.1 Estructura del código en *Arduino*.

Una vez obtenidos resultados satisfactorios a partir del código generado en *Matlab*, se utiliza el programa *Arduino IDE* para el desarrollo del *Software* en el módulo ESP8266 para las distintas señales deseadas. Los códigos completos aparecen en el anexo III.

3.2.1.1 Generación de ondas con ausencia de armónicos.

En primer lugar, se ha implementado un código sencillo en *Arduino* para generar dos ondas senoidales de tensión y corriente con la tarjeta ESP8266 y cuyo valor máximo es 3,3 V. Dicho código queda reflejado en el anexo III. En la Figura 11 se observa el diagrama de flujo que muestra de forma genérica el proceso de la generación de dichas ondas.

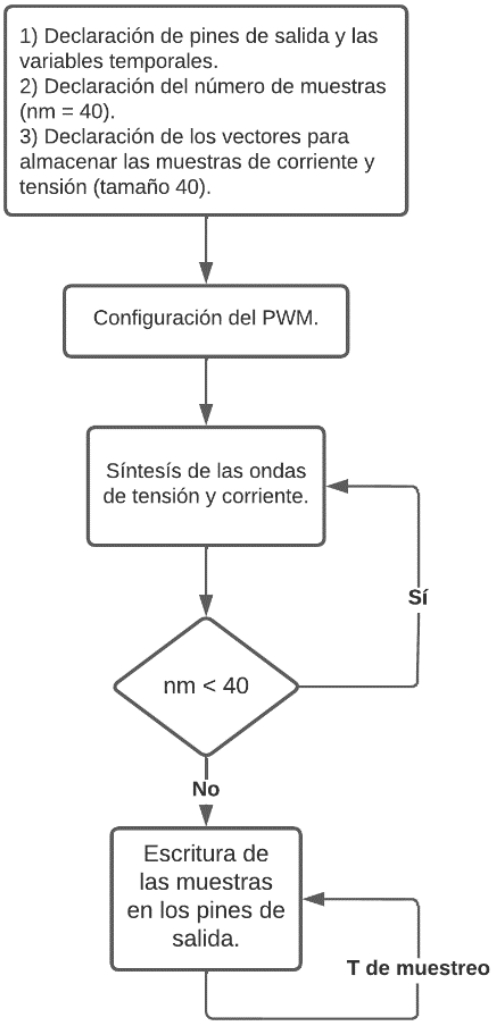


Figura 11: Diagrama de flujo sin armónicos.

Para la obtención de las ondas deseadas, se ha realizado:

1) La declaración de los pines de salida del microcontrolador ESP8266, D0 y D7, de tensión y corriente respectivamente.
2) En este caso, se quieren generar dos señales sinusoidales a una frecuencia de 50 Hz (armónico fundamental). Para ello, se declara la frecuencia (50 Hz), el número de muestras que coincide con la frecuencia de muestreo (40 muestras) y distintas variables relacionadas con el tiempo (*sampling_interval*, *t_actual* y *t)*.

3) Las muestras tanto de corriente como de tensión se almacenan en dos vectores de tamaño igual al número de muestras.
4) El microcontrolador ESP8266 genera una señal PWM de 8 bits (0 – 256), pero dispone de una función *analogWriteRange(1023)* que se configura de manera que genere una señal PWM de 10 bits que presentará una mayor precisión.
5) La frecuencia del PWM es de 1000 Hz por defecto, pudiendo variar entre 0,1 kHz y 40 kHz. A través de la *función analogWriteFreq(40000)* se establece la máxima frecuencia a la que puede trabajar para generar el PWM.
6) El número de muestras se ha establecido de forma experimental variando la cantidad y de esta manera determinar que el microcontrolador era capaz de generar la señal PWM a esa frecuencia. Con la ayuda del osciloscopio se ha comprobado que son suficientes para generar las dos señales. Se ha establecido un número de muestras de 40 siendo incapaz de generar ondas PWM para una cantidad superior de muestras.
7) A continuación, se implementa un código para generar las dos señales, a través de la función *sin(),* teniendo en cuenta que el rango de valores que ofrece el microcontrolador ESP8266 varía entre 0 y 3,3 V o lo que es lo mismo, entre 0 – 1023 bits. Para eliminar la parte negativa de la onda senoidal ya que el valor mínimo es de 0 bits, se reduce la amplitud de la onda a la mitad (512 bits) y se introduce un *offset* del mismo valor. De esta manera se obtiene la onda de amplitud entre 0 y 3,3 V.

```
samples_v[i] = (int)(512 * sin(2 * PI * t) + 512);
samples_i[i] = (int)(512 * sin(2 * PI * t) + 512);
```

8) Cada periodo de muestreo se almacenan las muestras de tensión y corriente en los vectores correspondientes. Se multiplica por 10^5 ya que se va a trabajar en unidades de microsegundos.

```
sampling_interval = (unsigned long) 1000000 / (f * ns);
```

9) En el bucle principal (*loop())* se escriben las muestras almacenadas en cada vector en los pines de salida D0 y D7 utilizando la función *analogWrite().*
   Por último, para conseguir una frecuencia de 50 Hz y eliminar el tiempo que tarda en ejecutar las líneas de código (*micros()-t_actual*), se resta al periodo de muestreo (*sampling_interval*) dicho tiempo.

```
void loop() {
   for (int j = 0; j < ns; j++) {
      analogWrite(v_out, samples_v[j]);
      analogWrite(i_out, samples_i[j]);
      delayMicroseconds(sampling_interval-(micros()-
t_actual));
      t_actual = micros();
      }
}
```

3.2.1.2 Generación de ondas en presencia de armónicos.

Para la generación de ondas en presencia de armónicos, el código no difiere en gran medida del caso anterior. Solamente varía en las líneas de código correspondientes a la implementación de la señal de corriente ya que en los dos casos se introducen armónicos (combinación de dos y tres armónicos). Por lo tanto, la onda de corriente no será puramente senoidal y presentará distorsiones. En la Figura 12 que se muestra a continuación, se observan los dos diagramas de flujo que muestran los pasos que se han seguido para la implementación de dichas ondas.

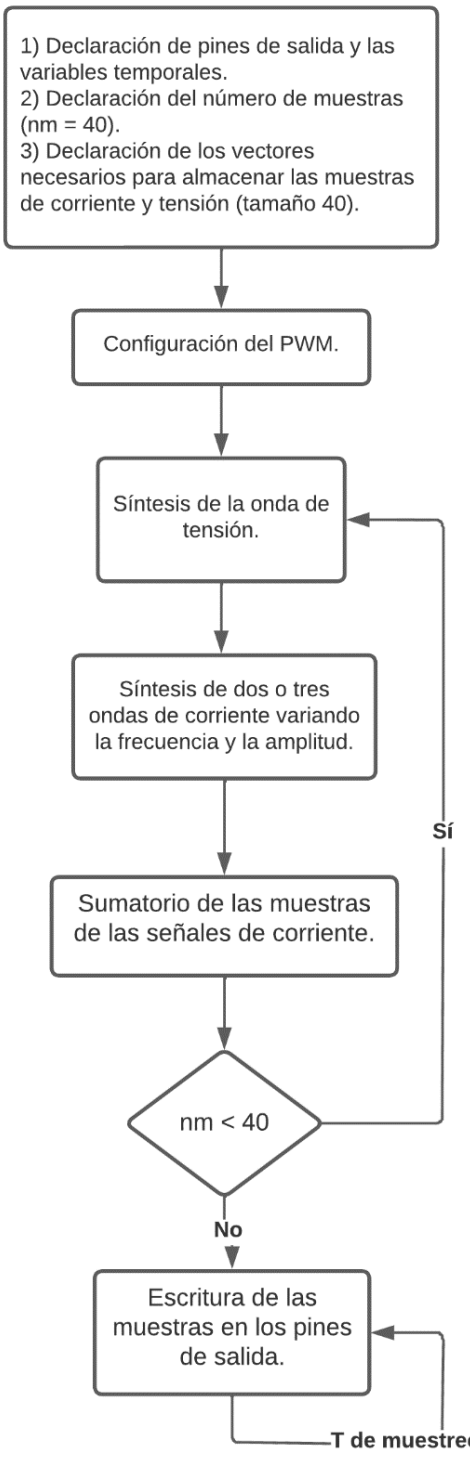


Figura 12: Diagrama de flujo para la generación de señales con la presencia de 2 ó 3 armónicos.

En el caso de la presencia de dos armónicos, Figura 12, se genera una señal de tensión del mismo modo que en el caso sin armónicos y dos señales de corriente donde se ha variado tanto la frecuencia como la amplitud. 1) Una onda de frecuencia de 50 Hz y su amplitud reducida a la mitad (256). 2) Onda de frecuencia de 100 Hz y de amplitud reducida a la cuarta parte (128). La onda de corriente resultante es la suma de las muestras de estas dos señales

seno. De esta manera, se evita sobrepasar el valor máximo de amplitud de 1023 una vez sumadas.

```
samples_f0[i] = (int)(256 * sin(2 * PI * t) + 256);
samples_f1[i] = (int)(128 * sin(2 * PI * 2 * t) + 128);
samples[i] = samples_f0[i] + samples_f1[i];
samples_v[i] = (int)(512 * sin(2 * PI * t) + 512);
```

En el caso de la combinación de tres armónicos (Figura 12) se varía la implementación de la señal de corriente, ya que, en este caso, se combinan tres armónicos. Por lo tanto, se han sumado las muestras de tres ondas senoidales cuyas frecuencias son f_0 = 50 Hz, f_1 = 100 Hz y f_2 = 150 Hz. Al igual que en el caso anterior y para que la amplitud no sobrepase el rango que ofrece el microcontrolador ESP8266 (0-1023), se han generado 3 ondas: 1) Señal seno generada a 50 Hz y de amplitud la mitad (256). 2) Señal seno a 100 Hz y de amplitud reducida a la cuarta parte (128). 3) Señal seno a 150 Hz y de amplitud reducida a la octava parte respecto al valor máximo.

```
samples_f0[i] = (int)(256 * sin(2 * PI * t) + 256);
samples_f1[i] = (int)(128 * sin(2 * PI * 2 * t) + 128);
samples_f2[i] = (int)(64 * sin(2 * PI * 3 * t) + 64);
samples[i] = samples_f0[i] + samples_f1[i] + samples_f2[i];
samples_v[i] = (int)(512 * sin(2 * PI * t) + 512);
```

La señal de tensión no sufre ninguna modificación y su amplitud corresponde al máximo valor posible, es decir, un valor de tensión de pico a pico de 3.3 V o, al ser valores digitales, 1023 bits.

3.2.2 Filtrado de la señal: Filtro paso bajo.

Una vez generadas las señales moduladas por ancho de pulso en el microcontrolador ESP8266, es necesario realizar un filtrado. De esta manera, se convierte el PWM en una señal analógica mediante un filtro paso bajo de primer orden, compuesto por una resistencia en serie y un condensador en paralelo en ambas salidas (Figura 13).

En este proyecto se ha utilizado un osciloscopio para observar las señales. Se han ido variando los parámetros según la documentación [18] , tanto del condensador como el de la resistencia, con los elementos disponibles en el laboratorio, para así obtener los parámetros óptimos para generar las señales de tensión y corriente (Figura 13, izquierda y derecha respectivamente). Finalmente se han fijado 132 kΩ para la resistencia (a partir de dos resistencias en paralelo R1 = 220 kΩ y R2 = 320 KΩ) y 27 nF para el condensador en el filtro de la señal de tensión y 84 kΩ (con dos resistencias en serie R1 = 15 kΩ y R2 = 69 kΩ) y C1= 10 nF para el filtro de la señal de corriente.

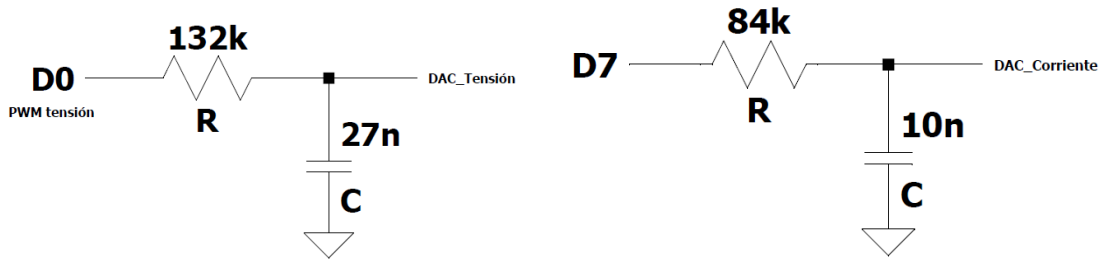


Figura 13: Filtro señal de la tensión: R= 132 kΩ y C1= 27 nF. f_c = 43 Hz (izquierda). Filtro señal de la corriente: R= 84 kΩ y C1= 10 nF. f_c = 189 Hz (derecha).

3.2.3 Adaptación de señales: seguidor de tensión.

En la parte final del *hardware*, se realiza un seguidor de tensión (Figura 14, izquierda) entre ambas ondas filtradas y la entrada de los pines de la tarjeta ESP32 con la que se leen los valores de ambas ondas. De esta manera se acoplan los circuitos sin modificar dichas ondas y se asegura la corriente de entrada mínima necesaria.

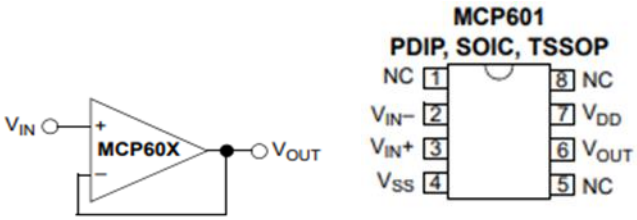
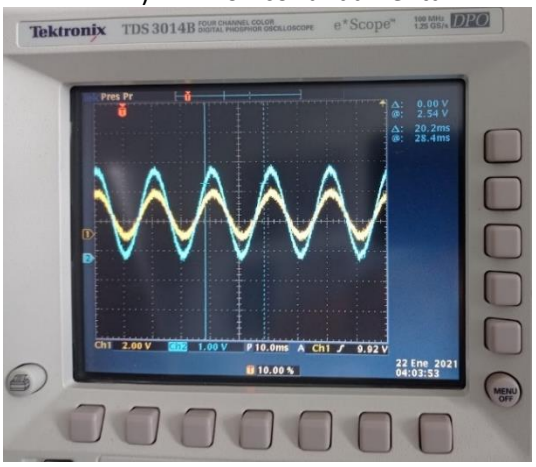


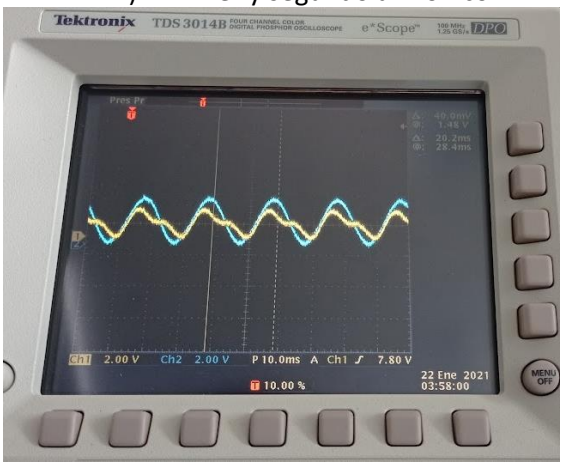
Figura 14: Circuito seguidor de tensión (Izquierda). Entradas y salidas del amplificador MCP601 (Derecha) [19]

Para ello, se utiliza el amplificador operacional MCP601 (Figura 14, derecha) cuya alimentación varía de 2,7 V a 6 V que se encuentra en el rango de tensiones que nos puede proporcionar tanto la tarjeta ESP82 como la ESP32. El resultado de las ondas finales tanto de tensión (en azul) como de corriente (amarillo) es muy satisfactorio (Figura 15) ya que se distingue el perfil de la onda coincidente con la sintetizada en el programa *Matlab* (Figura 8).

1) Armónico fundamental.



2) Primer y segundo armónico.



3) Primero, segundo y tercer armónico.

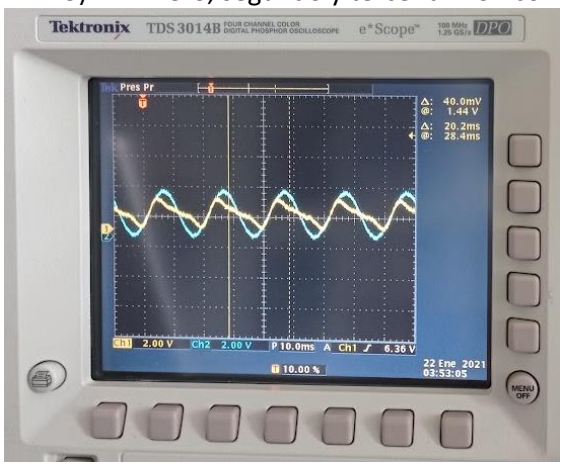


Figura 15: Ondas finales de tensión (en amarillo) y corriente (en azul).

3.3 Cálculo del consumo de energía.

3.3.1 Introducción.

En este capítulo se presentan los parámetros principales necesarios para el cálculo del consumo de energía: potencia activa, potencia aparente, tensión eficaz, corriente eficaz y factor de potencia.

Se parte de las ondas generadas por el microcontrolador ESP8266 que representan las ondas de tensión y corriente de la red eléctrica de una vivienda. Una vez implementadas, la tarjeta ESP32 es la encargada de tratar estos datos y calcular los parámetros necesarios con la ayuda de la librería *Emonlib* para el cálculo posterior del consumo eléctrico. Esta librería no tiene en cuenta la presencia de componentes frecuenciales, armónicos, en dichas ondas, por lo que, las siguientes fórmulas solo son aplicables a ondas de un solo armónico. A continuación, se explican brevemente los parámetros necesarios:

- Potencia Activa (P): es aquella que produce un trabajo útil en el circuito [20]. Esta potencia es, por lo tanto, la realmente consumida por los circuitos. Se utiliza para determinar la demanda eléctrica, en Vatios (W) [21] y se calcula como:

$$P = V \cdot I \cdot \cos(\theta)$$

siendo θ el desfase entre la señal de tensión y corriente.

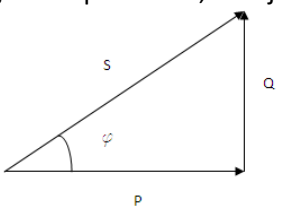
- Potencia Reactiva (Q): aparece en los circuitos de corriente alterna cuando existen bobinas y/o condensadores, es decir, las cargas no son puramente resistivas. No realiza trabajo útil ya que su valor medio es nulo, por lo que interesa reducirla al máximo [20]. Esta potencia es intercambiada por las cargas inductivas con la red y durante este proceso aparecen pérdidas. Se calcula, en voltamperios reactivos (VAr), como:

$$Q = V \cdot I \cdot \text{sen}(\theta)$$

- Potencia Aparente (S): es el producto de la tensión y corriente eficaz. Para cargas puramente resistivas, la potencia activa y aparente coinciden, pero para todas las demás cargas, la potencia activa será menor que la aparente [20]. Además, es el resultado del módulo entre la potencia activa y reactiva, en voltamperio (VA).

$$S = \sqrt{P^2 + Q^2}$$

- Factor de potencia (PF): es un indicador del rendimiento o nivel de eficiencia de una instalación. Se define como la relación entre la potencia activa (P), potencia que produce un trabajo útil, y la potencia total demandada a la red eléctrica, es decir, la potencia aparente (S). La compañía eléctrica aplica una multa si el factor de potencia es inferior a 0,85, mejorando así el rendimiento de las líneas. Este parámetro se calcula a través del triángulo de potencias, reflejado en la Figura 16.



$$\cos\theta = \frac{P}{S}$$

Figura 16: triangulo de potencias [21].

Una vez se han determinado estos parámetros, el consumo de energía (E) será el producto de la potencia aparente (S) por unidad de tiempo, en kWh. Esta energía se puede denominar energía aparente y es la que se refleja en las facturas de la luz [22] .

$$E = \left[S(VA) \cdot \frac{1\ (kVA)}{1000\ (VA)} \cdot \left(1\ (s) \cdot \frac{1\ (h)}{3600\ (s)}\right)\right]$$

$$E = \frac{S}{3600000} \quad [kWh]$$

Como se ha indicado anteriormente, este proyecto tiene dos objetivos principales, entre otros:

1) Simular una instalación eléctrica de una vivienda a partir de dos ondas tipo de tensión y corriente (explicadas en el apartado 3.1 de la memoria).
2) Realizar el cálculo y análisis de la precisión del consumo eléctrico que mediría un contador conectado al cuadro eléctrico de la vivienda. Como no se parte de valores reales de la red eléctrica doméstica, se determina la precisión de los cálculos a través de la librería *Emonlib* y la tarjeta ESP32. Para ello, se parte de cuatro potencias ya establecidas y normalizadas para instalaciones monofásicas (1,15 kW, 2,3 kW, 3,45 kW, 4,6 kW) [23]. Esta potencia se denomina potencia contratada. Este término hace referencia a la potencia acordada con la compañía eléctrica y es la máxima que se va a consumir. Se establece teniendo en cuenta, de manera estimada, el número de equipos que están trabajando al mismo tiempo en la vivienda.

3.3.2 Cálculos según la potencia contratada.

3.3.2.1 Aspectos generales.

El análisis de la precisión se realiza entre los valores teórico calculado mediante *Matlab* y los valores experimentales obtenidos por la librería *Emonlib*.

El cálculo de los valores en *Matlab* se han implementado las ondas de tensión y corriente generadas por el microcontrolador ESP8266 de la misma manera que en *Arduino*. En cambio, en este caso, los valores que se obtienen dependen de la potencia contratada mientras que en *Arduino* solamente dependen del rango de tensión que ofrece el microcontrolador.

A su vez, se han calculado los parámetros necesarios para la obtención del consumo eléctrico de la misma forma que lo realiza la librería *Emonlib*. Se supone constante la tensión de red, cuyo valor en España es de 230 V eficaces y se determina la corriente eficaz a partir de las distintas potencias contratadas que corresponden a la potencia activa, es decir la potencia útil. A continuación, se presenta los resultados que calcula *Matlab*. El código completo se puede analizar en el anexo III.

3.3.2.2 Cálculo de parámetros deseados con el programa Matlab.

Como se ha indicado anteriormente se van a calcular los parámetros deseados en 4 casos distintos imponiendo las potencias según la normativa española (1,15 kW, 2,3 kW, 3,45 kW, 4,6 kW). A continuación, se muestran las tablas que reflejan los resultados obtenidos a partir de las ondas implementadas en *Matlab*.

Los cálculos de los valores se realizan en *Matlab* partiendo de la potencia contratada y la tensión eficaz a partir de las cuales se obtiene la corriente eficaz (I_{rms}). Para la generación de las ondas de tensión y corriente, se establece una a amplitud de onda de valor:

$$V_{pico} = \sqrt{2} \cdot V_{rms} \;\; ; \;\; I_{pico} = \sqrt{2} \cdot I_{rms}$$

Cabría destacar que en el caso 4 se ha obtenido una corriente eficaz (I_{rms}) de 20 A, valor que no puede darse en una red eléctrica española ya que el máximo está establecido en 16 A. Es por esto que se ha impuesto este último valor para el cálculo de los parámetros.

Con relación al factor de potencia (PF), se debe de tener en cuenta la presencia o ausencia de armónicos. Si es el caso de tenerlos, se debe considerar le triángulo de potencias en tres dimensiones (Figura 17).

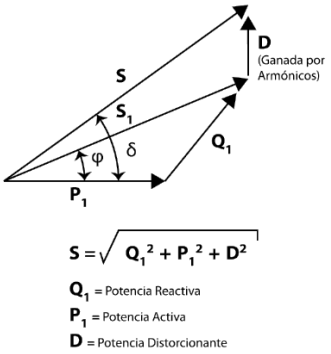


Figura 17: triángulo de potencias con la presencia de armónicos [24].

Se calcula la distorsión armónica total (THD) que depende del número de armónicos y se calcula como:

$$THDI = \sqrt{\frac{I_1^2 + I_2^2 + I_3^2 + \cdots + I_n^2}{I_s}} \cdot 100$$

Siendo:

I_n: corriente eficaz en cada armónico (Irms).

I_s: corriente eficaz de la señal fundamental.

Para calcular el factor de potencia final (PF_f), se debe incluir la distorsión armónica total de la siguiente manera:

$$PF_f = \frac{P}{S} \cdot \frac{1}{\sqrt{1 + \left(\frac{THDI}{100}\right)^2}}$$

Los valores obtenidos en todos los casos se comparan con los calculados experimentalmente a través de las ondas generadas por el sistema y la librería *Emonlib* para comprobar que son correctos.

1) Caso 1: Potencia contratada 1 (1,15 kW), Tabla 2: Parámetros del caso 1 (potencia = 1150 W).

P1 = 1150 W, Irms1 = 5 A, Vrms = 230 V					
	Irms1	**Vrms**	**P1**	**S1**	**PF1**
Armónico fundamental	4,987 A	229,427 V	1144,300 W	1144,300 VA	1,000
1º y 2º armónico	2,788 A	229,427 V	572,139 W	639,671 VA	0,800
1º, 2º y 3º armónico	2,805 A	229,427 V	572,139 W	643,656 VA	0,790

Tabla 2: Parámetros del caso 1 (potencia = 1150 W).

2) Caso 2: Potencia contratada 2 (2,3 kW), Tabla 3:

P2 = 2300 W, Irms2 = 10 A, Vrms = 230 V					
	Irms2	Vrms	P2	S2	PF2
Armónico fundamental	9,975 A	229,427 V	2288,6 W	2288,600 VA	1,000
1º y 2º armónico	5,562 A	229,427 V	1144,6 W	1279,300 VA	0,800
1º, 2º y 3º armónico	5,611 A	229,427 V	1144,3 W	1287,300 VA	0,790

Tabla 3: Parámetros del caso 2 (potencia = 2300 W).

3) Caso 3: Potencia contratada 3 (3,45 kW), Tabla 4:

P3 = 3450 W, Irms3 = 15 A, Vrms = 230 V					
	Irms3	Vrms	P3	S3	PF3
Armónico fundamental	14,962 A	229,427 V	3432,8 W	3432,8 VA	1,000
1º y 2º armónico	8,364 A	229,427 V	1716,4 W	1919,0 VA	0,800
1º, 2º y 3º armónico	8,416 A	229,427 V	1716,4 W	1931,0 VA	0,790

Tabla 4: Parámetros del caso 3 (potencia = 3450 W).

4) Caso 4: Potencia contratada 4 (4,6 kW), Tabla 5:

P4 = 4600 W, Irms4 = 16 A, Vrms = 230 V					
	Irms4	Vrms	P4	S4	PF4
Armónico fundamental	15,9601 A	229,427 V	3661,7 W	3661,7 VA	1,000
1º y 2º armónico	8,922 A	229,427 V	1830,8 W	2046,9 VA	0,800
1º, 2º y 3º armónico	8,9776 A	229,427 V	1830,8 W	2059,7 VA	0,790

Tabla 5: Parámetros del caso 4 (potencia = 4600 W).

4. Resultados.

4.1 Análisis de precisión.

Una vez calculados a través de *Matlab* los parámetros deseados, se comparan con los valores obtenidos por la librería *Emonlib* y de esta manera se comprueba si son correctos.

En primer lugar, se parte de las señales de tensión y corriente generadas por el microcontrolador ESP8266 con forma sinusoidal positiva de hasta 3,3 V de amplitud. A continuación, la librería *Emonlib* (explicada con un mayor detalle en el anexo III) transforma las señales seno positivas en señales senoidales bipolares al introducir un *offset.* Por lo tanto, se obtiene un valor máximo de amplitud en las ondas de tensión y corriente de ± 3,3/2 V. Se observa que este valor máximo de amplitud en las señales generadas de entrada dista de los valores de las ondas reales que se encuentran en la red eléctrica.

Para solventar esta diferencia, la librería *Emonlib* ofrece un factor de corrección, que generalmente se usa para dicha calibración si el sistema utiliza sensores, y se determina a partir del *datasheet* del sensor seleccionado.

En este caso, como no se utilizan sensores reales, se ha realizado un cálculo de valores con un factor de corrección 1 para ambas ondas y de esta manera obtener los parámetros sin calibrar. En este punto, se determina el valor máximo obtenido de tensión y corriente y se calibra de tal manera que devuelva los parámetros de tensión y corriente eficaz con la mayor precisión posible. Esta calibración se lleva cabo mediante los métodos *emon1.voltage()* y *emon1.current(),* propios de la librería *Emonlib*.

En la Tabla 6 quedan indicados los índices de calibración tanto de la tensión como de la corriente para cada caso.

Potencias	**P1 = 1150 W**		**P2 = 2300 W**		**P3 = 3450 W**		**P4 = 4600 W**	
Parámetros	V1	I1	V2	I2	V3	I3	V4	I4
Índice de calibración	210	4,4	210	8,8	210	13,2	210	14,08

Tabla 6: Índices de calibración de tensión y corriente en función de la potencia.

Por último, una vez calibrados los valores de tensión y corriente, se calculan los parámetros necesarios para la obtención del consumo eléctrico, entre ellos el factor de potencia. Cabría destacar que cuando las ondas de corriente presentan armónicos, se debe aplicar la distorsión armónica total (THD), calculada previamente en *Matlab* (50 para ondas de corriente combinando dos armónicos y 51,5388 para tres armónicos). Como *Emonlib* no tiene en cuenta este índice se debe implementar en el código de forma manual.

4.1.1 Resultados en *Emonlib*.

La configuración descrita anteriormente, se implementa en *Arduino IDE* para calibrar, calcular y mostrar por línea serie dichos parámetros (Figura 18). En el anexo III se muestra el código completo.

```
14:43:01.346 -> Real Power: 1129.014 W  Apparent Power: 1136.954 VA     Vrms: 230.141 V Irms: 4.940 A   Power Factor: 0.99
14:44:01.442 -> Real Power: 1127.910 W  Apparent Power: 1135.999 VA     Vrms: 229.944 V Irms: 4.940 A   Power Factor: 0.99
14:45:01.590 -> Real Power: 1127.623 W  Apparent Power: 1135.760 VA     Vrms: 229.782 V Irms: 4.943 A   Power Factor: 0.99
14:46:01.724 -> Real Power: 1127.590 W  Apparent Power: 1135.740 VA     Vrms: 229.740 V Irms: 4.944 A   Power Factor: 0.99
14:47:01.772 -> Real Power: 1127.967 W  Apparent Power: 1136.026 VA     Vrms: 229.543 V Irms: 4.941 A   Power Factor: 0.99
14:48:01.856 -> Real Power: 1127.993 W  Apparent Power: 1135.903 VA     Vrms: 229.761 V Irms: 4.944 A   Power Factor: 0.99
```

Figura 18: Parámetros mostrados por línea serie en Arduino IDE.

A continuación, se expone el diagrama de flujo (Figura 19) que representa el proceso que se lleva a cabo para la obtención de los parámetros deseados.

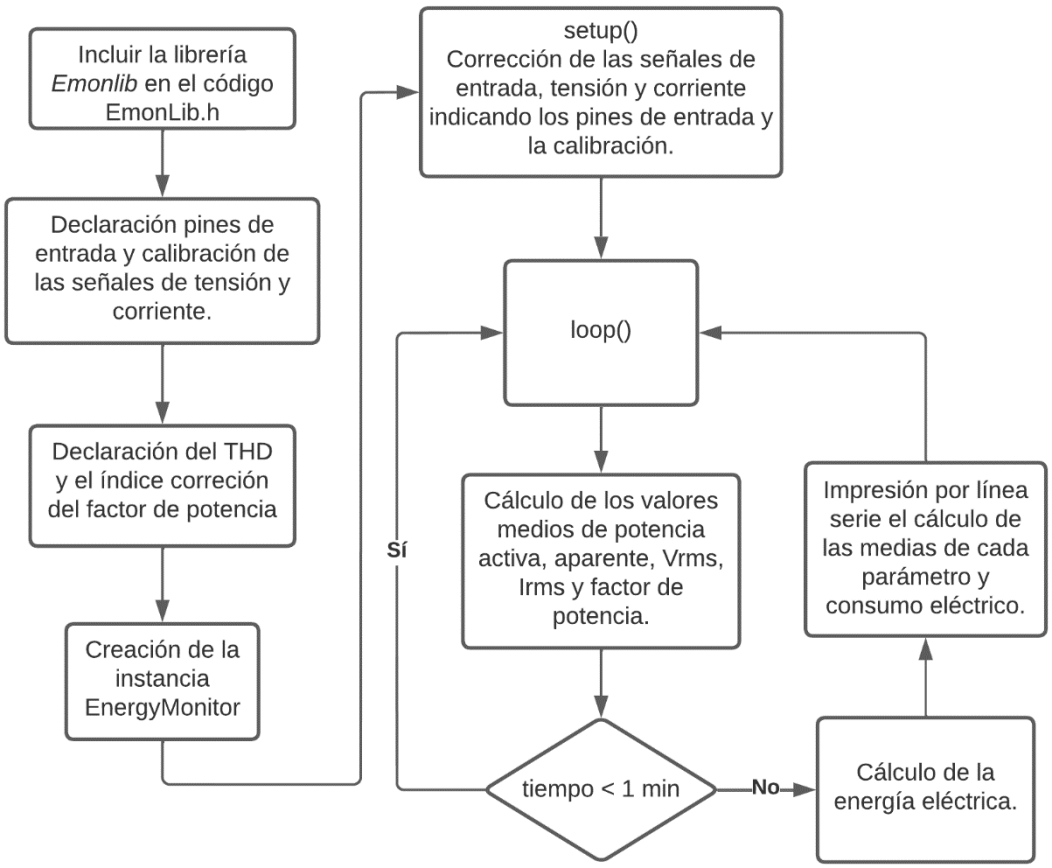


Figura 19: Diagrama de flujo para el cálculo de los parámetros deseados.

Una vez determinados dichos parámetros se calcula el consumo eléctrico siguiendo la fórmula E (kWh) = P (kW) · t (h).

4.1.1.1. Desfase de la señal de corriente respecto a la señal de tensión.

Hasta ahora, se han generado ondas de tensión y corriente representativas de la red eléctrica. Para que éstas sean lo más similares a las señales reales, se han introducido componentes armónicos en la onda de corriente recreando las cargas inductivas presentes en una vivienda. Sin embargo, se debe considerar que se genera potencia aparente (Q) ya que las señales pueden presentar un desfase entre ellas.

Por lo tanto, se han sintetizado dos ondas sin la presencia de armónicos con un desfase de 30° entre ellas. Se ha elegido este valor ya que, como se ha indicado en la explicación del factor de potencia (apartado 3.3 de la memoria), este parámetro debería ser mayor a 0,85 y, por lo tanto, el desfase máximo sería $\theta = \cos^{-1}(0{,}85) = 0{,}55\, rad$, aproximadamente 30°. Una vez

generadas las ondas finales se realizará el cálculo del error entre los valores experimentales (*Emonlib*) y los valores reales *(Matlab)*.

Para la síntesis de dichas ondas se parte del código en *Arduino* que genera las ondas fundamentales de tensión y corriente y se ha aplicado un desfase de π/6 en la onda de corriente.

```
samples_v[i] = (int)(512 * sin(2 * PI * t) + 512);
samples_i[i] = (int)(512 * sin(2 * PI * t + PI / 6) + 512);
```

Para comprobar que las ondas finales son las deseadas se han programado en *Matlab* dichas ondas reflejadas en la Figura 20 (izquierda) y se puede concluir que la salida es la deseada (Figura 20, derecha) ya que coinciden. A partir de los resultados, se comprueba que la onda de corriente (roja en *Matlab* y amarilla en el osciloscopio) está desfasada 30° respecto a la de tensión (azul). El código completo en *Arduino* y los *scripts* de *Matlab* están presentes en el anexo III.

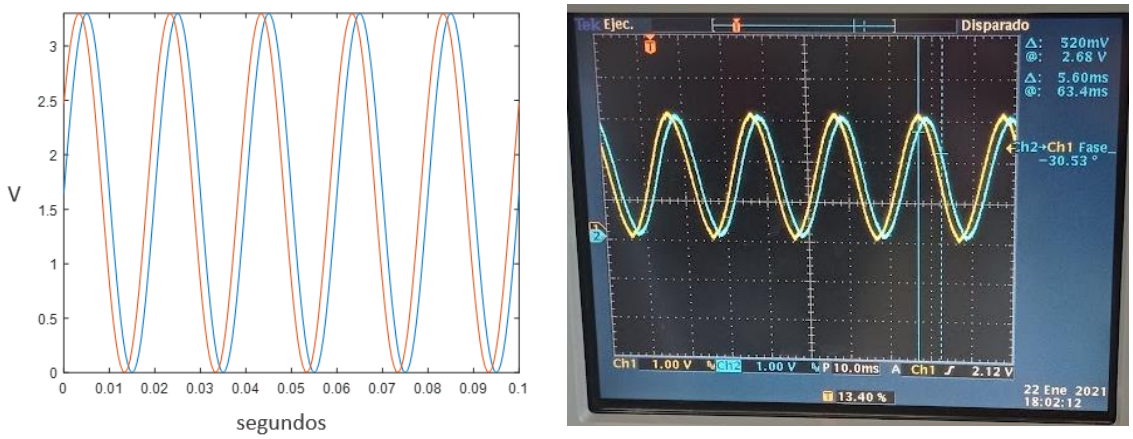


Figura 20: Generación de ondas mediante Matlab (izquierda) y con la tarjeta ESP8266 + filtro paso bajo + seguidor de tensión (derecha).

En la Tabla 7 se muestran los valores reales calculados por Matlab y los cuatro casos posibles calculando dichas ondas de tensión y corriente con las distintas potencias contratas elegidas.

Ondas desfasadas 30°	Irms (A)	Vrms (V)	P (W)	S (VA)	PF
P1 = 1150 W (caso 1)	4,993	229,427	990,974	1145,7	0.864
P2 = 2300 W (caso 2)	9,987	229,427	1981,900	2291,4	0.864
P3 = 3450 W (caso 3)	14,981	229,427	2972,900	3437,1	0.864
P4 = 4600 W (caso 4)	15,980	229,427	3171,100	3666,3	0.864

Tabla 7: Parámetros calculados con las ondas desfasadas en los cuatro casos.

4.1.1.2. Jitter

El *jitter* es un fenómeno provocado por la imprecisión del reloj de muestreo en los convertidores ADC (analógico/digital) y DAC (digital/analógico). El microcontrolador ESP32 convierte las ondas analógicas de tensión y corriente de entrada a tratar, en señales digitales mediante un conversor ADC. Una variación en el instante de muestreo puede producir una desviación en el valor de dichas ondas (Figura 21).

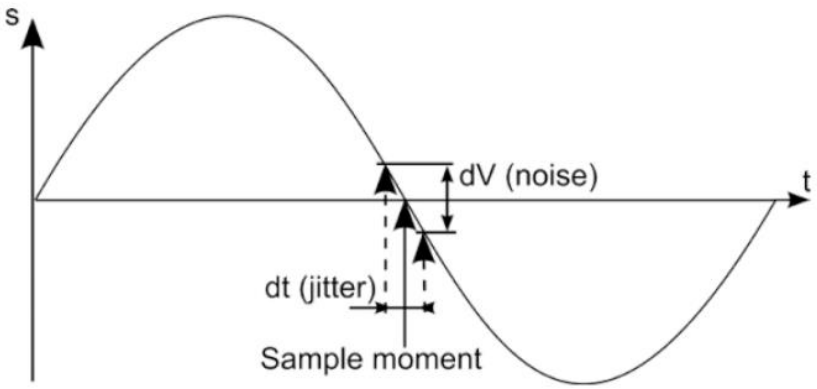


Figura 21: desviación en la medición la muestra (jitter) [25].

La librería *Emonlib* es la encargada de leer las muestras tanto de la señal de tensión como de corriente para el cálculo de los parámetros deseados y se ha concluido que la precisión de la conversión ADC del microcontrolador ESP32 es suficiente para la obtención de las medidas en este proyecto.

4.1.2 Cálculo de errores.

Para el cálculo del error entre los valores teóricos calculados en *Matlab* y los experimentales obtenidos con la ayuda de la librería *Emonlib*, se han registrado 11 muestras durante 10 minutos para que el cálculo sea lo más preciso posible. Para ello, se ha utilizado el código (detallado en el anexo III) siguiendo los pasos que se representan en el diagrama de flujo de la Figura 17, obteniendo los parámetros de potencia activa, potencia aparente, tensión eficaz, corriente eficaz y factor de potencia. El cálculo se ha llevado a cabo cada 1 minuto, durante 10 minutos a la velocidad máxima posible que permiten el microcontrolador y la librería *Emonlib.*

Para controlar la toma de medidas, se implementa un contador que aumenta en una unidad cada vez que se obtiene el cálculo de cada parámetro. Por último, se tratan estos datos calculando su media aritmética obteniendo valores más precisos y estables: se realiza un sumatorio de todos ellos y se imprime por pantalla la división del sumatorio de cada dato de interés y el número de cálculos realizados (registrado por el contador). Dichos cálculos quedan reflejados con detalle en el anexo V.

Se ha elegido para el cálculo del error, el error cuadrático medio (RMSE) o también conocido como Raíz de la Desviación Cuadrática Media. Se calcula siguiendo esta ecuación:

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(real_i - estimado_i)^2}$$

Siendo:

M: número de muestras (en este caso, 11 muestras).

$real_i$: parámetros calculados en *Matlab*

$estimado_i$: parámetros calculados en *Arduino* IDE con la ayuda de la librería *Emonlib*.

Para el cálculo del error, se plantean tres casos: 1) Ausencia de armónicos (Tabla 8). 2) Combinación de dos armónicos (Tabla 9). 3) Combinación de tres armónicos (Tabla 10). 4) Ondas desfasadas 30° (Tabla 11). En vista de los resultados obtenidos se puede concluir que el

cálculo mediante el sistema desarrollado es preciso ya que el error de los parámetros es inferior al 8 % en todos los casos. Cabría destacar que los valores más bajos de erros (entre 0,121 y 1,859 %) se dan en el caso de ausencia de armónicos mientras que el error más elevado (entre 0 y 8,716 %) se da en la combinación de tres armónicos. Este resultado se podría esperar debido a las distorsiones que aparecen en la señal de corriente a la hora de combinar tres armónicos.

Ausencia de armónicos	**Error caso 1 (P1 = 1,15 kW)**	**Error caso 2 (P2 = 2,3 kW)**	**Error caso 3 (P3 = 3,45 kW)**	**Error caso 4 (P4 = 4,6 kW)**
Potencia activa (P)	1,470 %	1,710 %	0,700 %	0,745 %
Potencia Aparente (S)	0,754 %	0,197 %	0,121 %	0,238 %
V eficaz (Vrms)	0,140 %	0,140 %	0,140 %	0,140 %
I eficaz (Irms)	0,905 %	0,911 %	0,763 %	0,655 %
Factor de potencia (PF)	1,000 %	1,859 %	1,000 %	0,798 %

Tabla 8: Errores con la señal de corriente generada sin combinación de armónicos

Dos primeros armónicos	**Error caso 1 (P1 = 1,15 kW)**	**Error caso 2 (P2 = 2,3 kW)**	**Error caso 3 (P3 = 3,45 kW)**	**Error caso 4 (P4 = 4,6 kW)**
Potencia activa (P)	5,500 %	5,397 %	5,376 %	4,920 %
Potencia Aparente (S)	3,900 %	3,399 %	2,800 %	2,834 %
V eficaz (Vrms)	0,140 %	0,140 %	0,140 %	0,140 %
I eficaz (Irms)	4,469 %	3,761%	3,426 %	3,483 %
Factor de potencia (PF)	1,554 %	2,500 %	2,500 %	2,500 %

Tabla 9: Errores con señal de corriente generada combinando dos armónicos.

Tres primeros armónicos	**Error caso 1 (P1 = 1,15 kW)**	**Error caso 2 (P2 = 2,3 kW)**	**Error caso 3 (P3 = 3,45 kW)**	**Error caso 4 (P4 = 4,6 kW)**
Potencia activa (P)	7,776 %	4,542 %	4,512 %	4,429 %
Potencia Aparente (S)	7,770 %	4,627 %	4,601 %	4,518 %
V eficaz (Vrms)	0,140 %	0,140 %	0,140 %	0,140 %
I eficaz (Irms)	8,716 %	6,006 %	5,973 %	5,930 %
Factor de potencia (PF)	1,266 %	1,266 %	0,000 %	0,000 %

Tabla 10: Errores con señal de corriente generada combinando tres armónicos.

Ondas desfasadas 30°	Error caso 1 (P1 = 1,15 kW)	Error caso 2 (P2 = 2,3 kW)	Error caso 3 (P3 = 3,45 kW)	Error caso 4 (P4 = 4,6 kW)
Potencia activa (P)	4,816 %	4,288 %	3,838 %	3,612 %
Potencia Aparente (S)	1,485 %	0,296 %	0,184 %	0,414 %
V eficaz (Vrms)	0,150 %	0,150 %	0,150 %	0,150 %
I eficaz (Irms)	1,638 %	0,49 %	0,187 %	0,241 %
Factor de potencia (PF)	3,417 %	4,025 %	3,941 %	3,899 %

Tabla 11: Errores calculados con la señal de corriente desfasada 30° respecto a la de tensión.

4.2 Visualización de los parámetros deseados y consumo eléctrico.

4.2.1. Aspectos generales.

Una vez realizado el cálculo de los parámetros deseados (potencia activa, potencia reactiva, tensión eficaz, corriente eficaz y factor de potencia) indicados en el apartado 4.1.1 de la memoria se determina el consumo eléctrico como E (kWh) = P (kW) · t (h).

Como parte final de este proyecto se desea enviar todos los parámetros calculados por la librería *Emonlib* a la plataforma *OpenEnergyMonitor*: *Emoncms* (en el anexo IV se detalla brevemente las características principales) para su posterior visualización. Con el objetivo de que los datos visualizados sean lo más realistas posibles se va a reproducir un ejemplo real del consumo eléctrico diario en una vivienda (Figura 22).

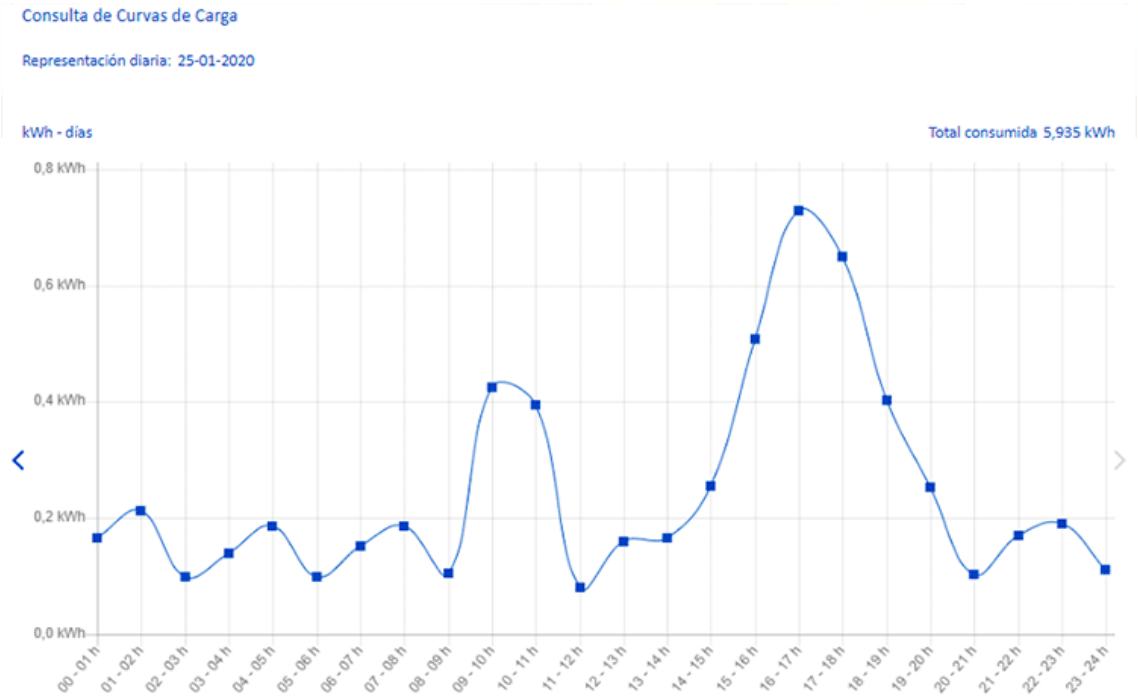


Figura 22: Curva del consumo eléctrico diario en una vivienda [26] .

4.2.2. Envío de datos a la plataforma *OpenEnergyMonitor: Emoncms*.

4.2.2.1. Estructura del código Arduino.

Una vez elegido el modelo real a reproducir (Figura 22) se utiliza el programa Arduino IDE para la implementación de código. Dicho código aparece en el anexo III.

En primer lugar, se extrapola temporalmente el modelo, es decir, enviamos los datos a la plataforma *Emoncms* cada 10 segundos y no cada hora como nos muestra el caso real. Por lo tanto, la toma completa de muestras es de 240 s (4 min).

Se establecen los valores de energía eléctrica en cada periodo de forma aproximada (Tabla 12) y a partir de dicho valor se calibran las muestras de entrada de la señal de tensión y de corriente. El factor de calibración de la tensión es fijo (210, apartado 4.1), mientras que la corriente varía de tal forma que se consiga los distintos valores de potencia y, por ende, del consumo eléctrico.

Horario (h)	**00 – 01 h**	**01 – 02 h**	**02 – 03 h**	**03 – 04 h**	**04 – 05 h**	**05 – 06 h**
Consumo eléctrico (kWh)	0,17 kWh	0,2 kWh	0,1 kWh	0,15 kWh	0,2 kWh	0,1 kWh
Horario (h)	**06 – 07 h**	**07 – 08 h**	**08 – 09 h**	**09 – 10 h**	**10 – 11 h**	**11 – 12 h**
Consumo eléctrico (kWh)	0,15 kWh	0,2 kWh	0,1 kWh	0,42 kWh	0,4 kWh	0,08 kWh
Horario (h)	**12 – 13 h**	**13 – 14 h**	**14 – 15 h**	**15 – 16 h**	**16 – 17 h**	**17 – 18 h**
Consumo eléctrico (kWh)	0,17 kWh	0,18 kWh	0,25 kWh	0,5 kWh	0,73 kWh	0,65 kWh
Horario (h)	**18 – 19 h**	**19 – 20 h**	**20 – 21 h**	**21 – 22 h**	**22 – 23 h**	**23 – 24 h**
Consumo eléctrico (kWh)	0,4 kWh	0,25 kWh	0,1 kWh	0,18 kWh	0,2 kWh	0,1 kWh

Tabla 12: Consumo eléctrico por horas.

A continuación, se expone el diagrama de flujo (Figura 23) que representa la idea general del proceso que se lleva a cabo para la imitación del modelo.

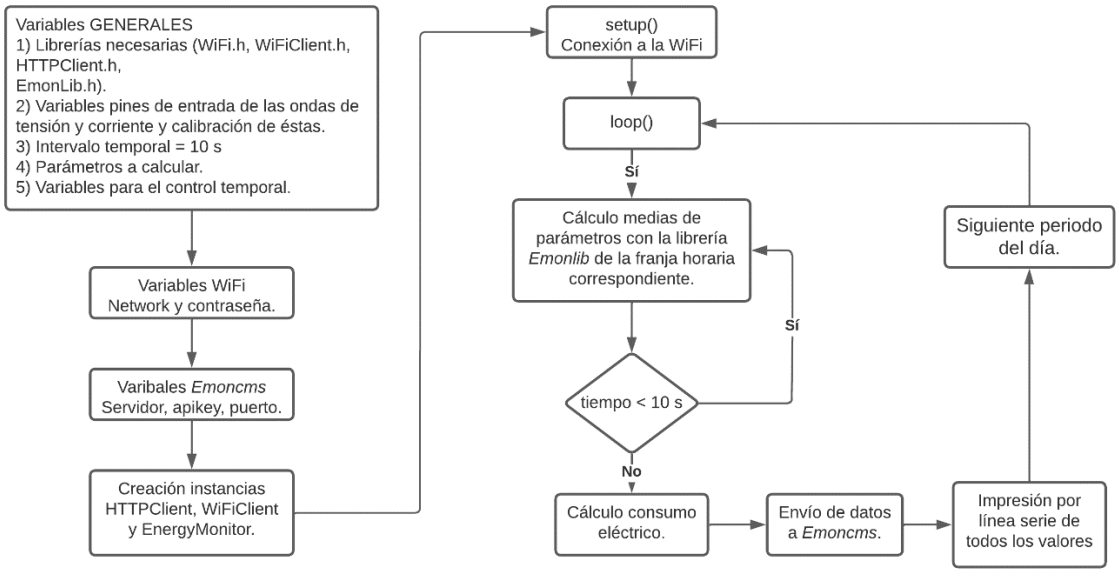


Figura 23 : Diagrama de flujo para simular el consumo diario de una vivienda.

Para la simulación del consumo eléctrico diario de una vivienda, se ha realizado:

a) Declaración de las librerías necesarias y la creación de instancias para poder acceder a éstas.
b) Declaración de variables como los pines de entrada de las ondas de tensión y corriente, variables temporales para controlar que los datos se envían cada 10 s y los parámetros deseados (potencia activa, potencia aparente, tensión y corriente eficaz y factor de potencia) y el consumo eléctrico.
c) Para cada franja horaria se realiza las medias de los parámetros durante 10 s (representando 1 h) y una vez que termina, se calcula la energía eléctrica y por último se envían a la plataforma *Emoncms* a través de la APIKey, una clave de lectura y escritura que adjudica la aplicación. Este proceso queda detallado en el anexo IV.
d) Se imprimen los valores por línea serie para comprobar que los valores calculados por la librería *Emonlib* coinciden con los que recibe la entrada generada en *Emoncms*. Posteriormente, en la plataforma, se crea un *feed* (lugar donde se registran los valores enviados) por cada entrada y se configura para que la recepción de datos sea cada 10 s. Este proceso y conceptos de la aplicación web *Emoncms* quedan reflejados en el anexo IV.

4.2.2.1. Configuración WiFi y *FreeRTOS.*

Se ha incluido este apartado para explicar brevemente el uso de *FreeRTOS* en el trabajo de fin de grado. Como se ha comentado en la descripción de los componentes utilizados (apartado 2.3) el microcontrolador ESP32, responsable del tratamiento de las muestras de las ondas de tensión y corriente, del cálculo de parámetros y del envío de datos, cuenta con dos núcleos (*core 0* y *core 1*) [27]. Por defecto, cuando se ejecuta el código en *Arduino IDE*, si no se indica al microcontrolador con qué *core* trabajar, utiliza el *core 1.* FreeRTOS permite asignar partes del código al *core* deseado mediante la creación de tareas, especificando el núcleo donde va a trabajar y la prioridad de ésta (siendo 0 la más baja). El estudio del sistema operativo *FreeRTOS* está detallado en el anexo III.

Por lo tanto, se ha implementado un código, presente en el anexo 3, creando dos tareas con la misma prioridad:

1) Tarea 1: Mantiene la WiFi conectada y se revisa la conexión cada 10 s. Esta tarea se ejecutará en el *core 0.*
2) Tarea 2: Cálculo de los valores medios de la potencia activa, aparente, tensión y corriente eficaz y factor de potencia a través de la librería *EmonLib.* Por simplicidad, se va a utilizar como entrada al microcontrolador ESP32 las ondas fundamentales de corriente y tensión y los valores impuestos a partir de la potencia contratada 1 (P1 = 1150 W). Esta tarea se ejecutará en el *core 1* y se ejecutará cada 10 s, al igual que la comprobación de la conexión WiFi.

Se utiliza la función: `xPortGetCoreID()` para que nos indique en qué *core* se está ejecutando la tarea (Figura 24). El código completo queda reflejado en el anexo III.

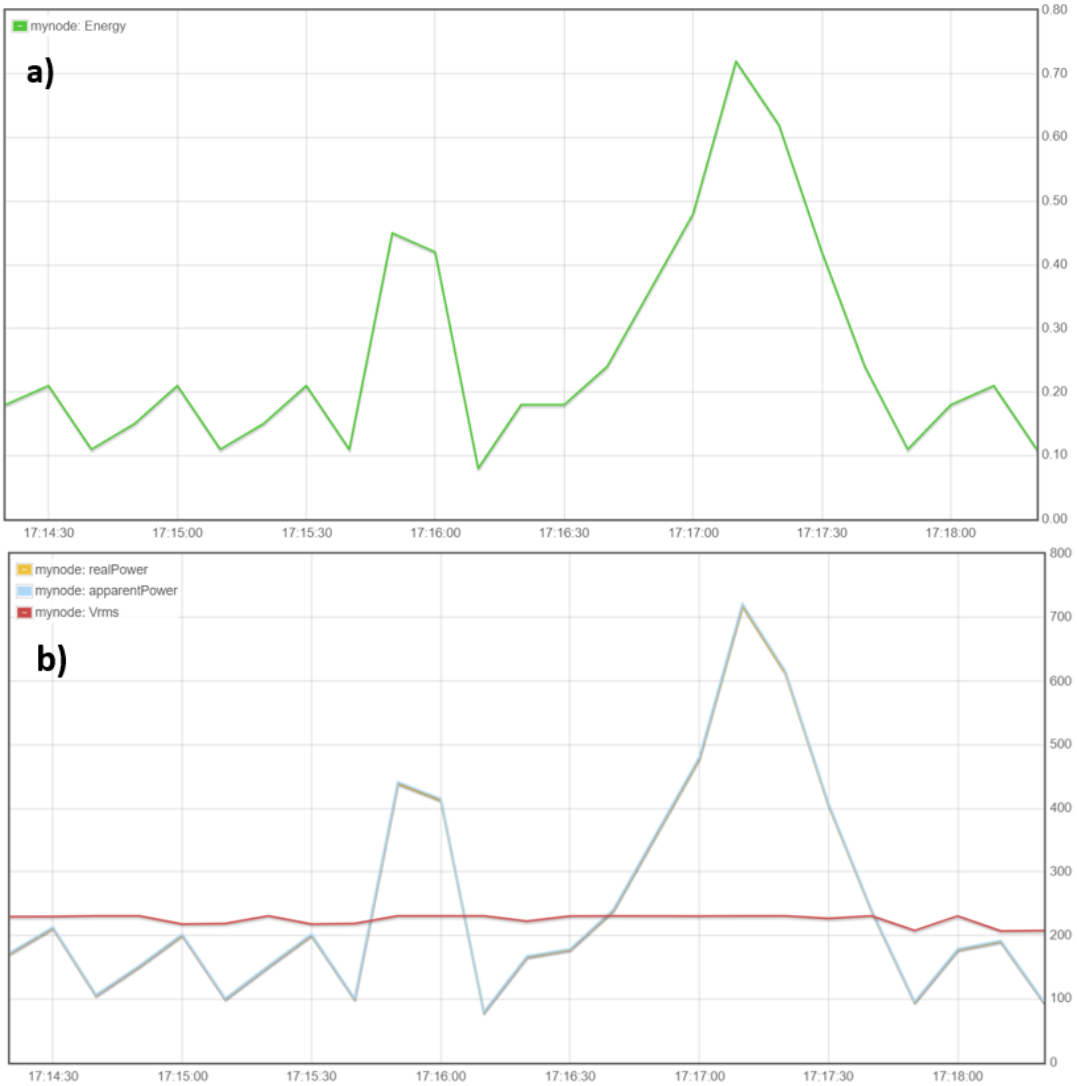
```
17:31:35.189 -> WiFi Connected: 192.168.46.194
17:31:35.189 -> WiFi: Core --> 0
17:31:35.189 -> WiFi still connected
17:31:41.054 -> Real Power: 1115.432 W  Apparent Power: 1134.949 VA     Vrms: 231.279 V Irms: 4.907 A   Power Factor: 0.983
17:31:41.054 -> Emonlib: Core --> 1
17:31:45.198 -> WiFi: Core --> 0
17:31:45.198 -> WiFi still connected
17:31:51.117 -> Real Power: 1117.264 W  Apparent Power: 1136.923 VA     Vrms: 231.621 V Irms: 4.908 A   Power Factor: 0.983
17:31:51.117 -> Emonlib: Core --> 1
17:31:55.225 -> WiFi: Core --> 0
17:31:55.225 -> WiFi still connected
17:32:01.194 -> Real Power: 1117.203 W  Apparent Power: 1136.762 VA     Vrms: 231.589 V Irms: 4.908 A   Power Factor: 0.983
```

Figura 24: Core utilizado en cada tarea (Salida línea serie).

Como se observa en la Figura 24, las tareas creadas se han ejecutado en los respectivos *cores*. Sin embargo, para este proyecto se ha decidido que el código se ejecute en el núcleo que el microcontrolador ESP32 elige por defecto ya que no se aprecian retrasos temporales significativos.

4.2.2.2. Gráficas resultantes.

Una vez realizado el envío a la nube se visualizan las gráficas generadas por *Emoncms* comprobando que se ha reproducido de forma correcta el modelo real y que los parámetros calculados se han recibido cada 10 s. En la Figura 25 que se muestra a continuación se puede observar: a) El consumo eléctrico. b) la tensión eficaz, la potencia activa y aparente que como era de esperar coinciden debido a que las ondas tanto de tensión como de corriente no presentan armónicos. c) Consumo eléctrico, corriente eficaz y factor de potencia.



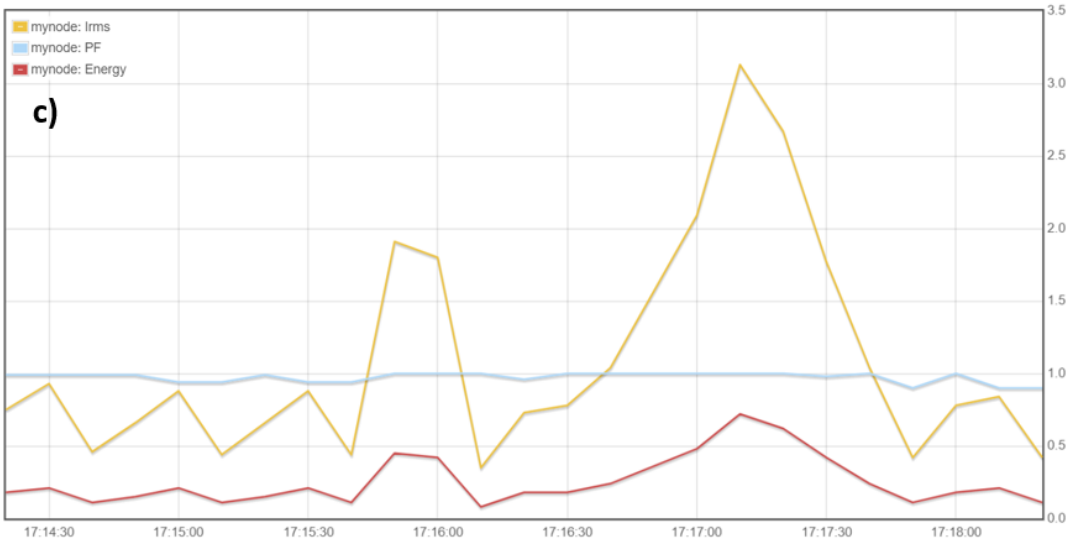


Figura 25: Gráficas mostradas por la aplicación web Emoncms.

Se debe tener en cuenta que las unidades no están presentes en los ejes de las gráficas. Por lo tanto, el eje de abscisas representa el tiempo, segundos (s), aunque en el modelo real sería el tiempo en horas (h). Respecto al eje de ordenadas, cada parámetro se mide en sus respectivas unidades: potencia activa en vatios (W), potencia aparente en voltamperios (VA), tensión eficaz en voltios (V), corriente eficaz en amperios (A) y consumo eléctrico en kWh (extrapolado al modelo real).

5. Conclusiones y líneas futuras.

5.1 Conclusiones.

A partir de lo realizado a lo largo de todo el proyecto, se puede afirmar que:

- Se ha demostrado que es posible diseñar y desarrollar correctamente un medidor de consumo eléctrico, sencillo y económico a partir de los dispositivos disponibles en un laboratorio.
- Se ha obtenido además un medidor sin hacer uso de sensores, a diferencia de los medidores del mercado. En este caso se ha sustituido el uso del sensor por la generación de una señal PWM y un filtro de paso bajo con resultados satisfactorios. Esto es de gran utilidad durante la fase de desarrollo de un posible producto final.
- Se ha diseñado un código en la plataforma *Arduino IDE* mediante el cual se consiguen dos señales de tensión y corriente variando la última señal al introducir componentes armónicos o desfases. De esta manera, se asemejan a las señales reales de la red eléctrica de una vivienda.
- Se ha estudiado la precisión de los cálculos realizados por la librería *Emonlib* con las distintas señales, pudiendo concluir que se realizan de forma correcta.
- Mediante el diseño de un código utilizando el sistema operativo *FreeRTOS* se ha gestionado correctamente, en distintos *cores,* dos tareas para la configuración de la WiFi y el cálculo parámetros (potencia activa, aparente, tensión y corriente eficaz y factor de potencia).
- Envío en tiempo real de los datos calculados por *Emonlib* y su posterior visualización en una aplicación web gratuita, *Emoncms*.
- Simulación, envío y visualización del consumo eléctrico diario que puede consumir una vivienda de forma satisfactoria.

5.2 Líneas futuras.

En primer lugar, respecto al sistema desarrollado se podrían simular casos reales del funcionamiento de los distintos aparatos eléctricos presentes en una vivienda manteniendo la onda de tensión fija y modificando la onda de corriente (variar la amplitud, el contenido de componentes frecuenciales o el desfase). De esta manera, se mediría el consumo eléctrico de dicha carga sin que esté presente físicamente.

Respecto a las herramientas Software, se podría estudiar con mayor detalle el sistema operativo *FreeRTOS* para gestionar un mayor número de tareas si fuera necesario. En cuanto a la librería *Emonlib*, se plantea el problema de que no tiene en cuenta la presencia de armónicos en la señal de entrada a la hora de calcular el factor de potencia. Para resolver este problema, se debería hacer un estudio para determinar si el microcontrolador ESP32 puede detectar la presencia de armónicos y así calcular el parámetro de distorsión armónica THD. Por último, extender la funcionalidad del código para añadir una pantalla LCD para la visualización de los datos de manera que sobre el diseño final se implementara una placa de circuito impresa y una carcasa para poder ser utilizado en el ámbito doméstico.

Para finalizar, se podría desarrollar una aplicación propia, a través por ejemplo de *Android Studio*, para que el envío de los datos se haga de manera personalizada al usuario.

6. Referencias bibliográficas.

[1] R. Hannah, P. Rosado y M. Roser, «Energy". Published online at OurWorldInData.org,» 2020. [En línea]. Available: https://ourworldindata.org/energy.

[2] K. Yoomi, T. Katsuya y M. Shunji, «Environmental and economic effectiveness of the Kyoto Protocol.,» *PLOS ONE,* 21 July 2020.

[3] «Global Energy Statistical Yearbook (2017),» 12 Abril 2018. [En línea]. Available: https://yearbook.enerdata.net/.

[4] A. Turiel, «csic,» [En línea]. Available: https://www.csic.es/es/actualidad-del-csic/antonio-turiel-la-escasez-de-materiales-es-una-estaca-en-el-corazon-de-la.

[5] «Comisión Europea.,» 6 Abril 2018. [En línea]. Available: https://ec.europa.eu/clima/policies/strategies/2020_e.

[6] M. Vijay y P. F. Hernán, «Objetivo de Desarrollo Sostenible para la energía y la tecnología de la información y las comunicaciones».

[7] M. Steiner y J. Scholten, «Energy Storage in board of railway vehicles.,» 2005.

[8] 24 08 2007. [En línea]. Available: https://www.boe.es/eli/es/rd/2007/08/24/1110/dof/spa/pdf.

[9] [En línea]. Available: https://comparadorluz.com/faq/contador-luz.

[10] [En línea]. Available: https://nergiza.com/contadores-inteligentes-todo-lo-que-tienes-que-saber/.

[11] [En línea]. Available: https://www.genbeta.com/web/como-saber-consumo-electrico-tiempo-real-casa.

[12] «ONU.,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/.

[13] 2020. [En línea]. Available: https://savjee.be/2020/02/esp32-keep-wifi-alive-with-freertos-task/.

[14] «Open Energy Monitor,» [En línea]. Available: https://guide.openenergymonitor.org/applications/home-energy/.

[15] «emoncms,» [En línea]. Available: https://emoncms.org/.

[16] «Carlos Alcaraz,» [En línea]. Available: https://carlosalcaraz.com/e/.

[17] R. Keim, «Low-Pass Filter a PWM Signal into an Analog Voltage,» *All About Circuits,* 11 Abril 2016.

[18] R. Keim, «What Is a Low Pass Filter? A Tutorial on the Basics of Passive RC Filters,» *All About Circuits.*

[19] «Datasheet AO,» [En línea]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/21314g.pdf.

[20] 27 04 2018. [En línea]. Available: https://ie2mmo.wordpress.com/2018/04/27/t02-2-potencias-en-corriente-alterna/.

[21] «ie2mmo wordpress,» 27 06 2019. [En línea]. Available: https://ie2mmo.wordpress.com/2019/06/27/t2-4-factor-de-potencia/.

[22] «InforTel,» [En línea]. Available: http://www.infortel.es/energia%20y%20potencia.pdf.

[23] «PrecioGas by Selectra,» [En línea]. Available: https://preciogas.com/faq/factura-luz/potencias-normalizadas.

[24] F. Francesc. [En línea]. Available: https://fornieles.es/perturbaciones-electricas/diferencia-coseno-phi-y-factor-potencia.

[25] 27 Mayo 2019. [En línea]. Available: https://www.studio-22.com/blog/enciclopedia/jitter.

[26] [En línea]. Available: https://selectra.es/energia/tramites/calcular-consumo-electrico.

[27] [En línea]. Available: https://randomnerdtutorials.com/esp32-dual-core-arduino-ide/.

[28] Grupo Novelec, [En línea]. Available: https://blog.gruponovelec.com/electricidad/medidores-de-consumo-electrico-guia-de-compra/.

[29] Efergy, [En línea]. Available: https://es.efergy.com/elite-classic/.

[30] «Efergy,» [En línea]. Available: https://es.efergy.com/wp-content/uploads/2019/06/Efergy_Engage_Hub_Single-Phase_Installation_Guide_04062019.pdf.

[31] [En línea]. Available: https://es.efergy.com/engage-hub-solo/.

[32] [En línea]. Available: https://es.efergy.com/engage-sub-metering-kit/.

[33] [En línea]. Available: https://www.efimarket.com/monitor-energetico-owl-cm160-usb-001-006.

[34] [En línea]. Available: https://www.xataka.com/energia/un-mes-usando-smappee-el-gadget-para-conocer-al-instante-el-consumo-de-cada-electrodomestico.

[35] J. Ceja, R. Renteria, R. Ruelas y G. Ochoa, «Módulo ESP8266 y sus aplicaciones en el internet de las cosas,» *Revista de Ingeniería Eléctrica,* nº 2,24-36, 23 Julio 2017.

[36] «WeMos ESP8266,» [En línea]. Available: https://articulo.mercadolibre.com.mx/MLM-785731818-wemos-d1-mini-arduino-esp8266-nodemcu-4m-bytes-lua-wifi-_JM.

[37] J. Ikiss, «Sistema de Adquisición de datos con ESP32,» Barcelona, 2020.

[38] [En línea]. Available: https://es.wikipedia.org/wiki/Arduino_IDE.

[39] [En línea]. Available: https://blog.330ohms.com/2021/04/06/introduccion-a-los-so-en-tiempo-real-planificacion-de-tareas/.

[40] L. Luis. [En línea]. Available: https://www.luisllamas.es/como-usar-freertos-en-arduino/.

[41] «GitHub: Emoncms,» [En línea]. Available: https://github.com/emoncms/emoncms.

[42] «APIKey,» [En línea]. Available: https://emoncms.org/user/view.

[43] «Input API,» [En línea]. Available: https://emoncms.org/input/api.

I. Anexo 1: Tipos de medidores en el mercado.

I.1 Introducción

Como se ha reflejado en la memoria, nuestro objetivo es implementar un medidor eléctrico desde componentes electrónicos e implementación de código, que sea capaz de medir el consumo eléctrico a partir de señales generadas de tensión y corriente. Una vez sintetizadas las ondas de entrada, se calculan los parámetros necesarios para la obtención de la energía eléctrica y su posterior envío a la nube. Por lo tanto, sería un contador capaz de calcular el consumo con los valores de tensión y corriente y enviarlos a alguna plataforma gratuita en Internet.

La Tabla I. 1, recopila los medidores disponibles en el mercado.

Medidor	Descripción	Especificaciones	Precio
Power meter de Schneider Electric 	Medidor digital diseñado para contar los vatios-hora que se consumen por circuito eléctrico, monofásico y trifásico.	Pantalla LCD. Mide potencia activa y reactiva, potencia aparente, tensión, corriente, energía, factor de potencia, frecuencia, THD (I)[2] y THD (U)[3].	488,32 €
Contadores de energía EMDX3 de Legrand 	Este contador digital mide la energía eléctrica consumida por un circuito monofásico o trifásico situado aguas abajo del contador de energía.	Pantalla LCD donde muestra el consumo de energía en kWh, así como la corriente, la energía activa, la energía reactiva y la potencia.	245,52 €

[2] THD (I): Distorsión armónica de corriente total.
[3] THD (U): Distorsión armónica de tensión total.

Medidor de consumo eléctrico Delta dore TYWATT-30 	Este contador digital se instala en el cuadro eléctrico para la medición del número de kilovatios-hora consumidos por una instalación eléctrica.	Doble contador (uno permanente, uno reiniciable). Pantalla para visualizar el número de kWh consumidos dentro del hogar.	127,05 €
Medidor de consumo eléctrico Efergy Elite Classic 	Este medidor digital es una solución inalámbrica (wireless) para monitorizar el consumo eléctrico. La instalación es fácil, se conecta el mini sensor (pinza) al cable de fase del cuadro eléctrico. Registra y almacena datos en tu PC.	Amplia pantalla LCD donde se muestra el coste en €, energía W/h, consumo de amperios y emisiones de CO_2.	79,95 €
Smappee 	Gadget que te permite conocer al instante el consumo eléctrico. Instalación mediante una especie de pinza en el cable de fase de nuestro cuadro eléctrico. Se conecta al WiFi, para así enviar los datos que recopila y acceder a ellos a través del ordenador o móvil.	La aplicación de Smappee va detectando los aparatos y muestra los kWh consumidos por cada dispositivo y el precio €, así como el total consumido	235 €
AC80-300V Voltímetro Medidor de Potencia 	Contador digital que se instala en el cuadro eléctrico de la vivienda, para medir la corriente se debe pasar el cable de fase por la sonda amperimétrica que lleva en el interior y el neutro hay que conectarlo para que pueda medir la tensión.	Mide los kWh que estoy consumiendo en la vivienda, el factor de potencia del consumo, voltaje, tiempo (h), amperios y la potencia (W) consumida en cada instante.	10,83 €

Tabla I. 1: Comparación de los medidores de consumo presentes en el mercado [28].

I.1.1 Efergy Elite Classic

Una vez realizada la comparativa de los distintos medidores de consumo, se observó que el monito Efergy Elite Classic era el contador que contaba con más similitudes respecto a las funciones realizadas a nuestro objetivo.

Este medidor de consumo está compuesto por: 1) un monitor, 2) un sensor, 3) un transmisor, y 4) un complemento para gestionar los datos online (*Hub*), un complemento para gestionar los datos online.

1) Monitor

El monitor muestra en una pantalla LCD portátil y compacta el consumo instantáneo, promedio e históricos en kilovatios-hora, coste (€) y huella de carbono (kg CO_2eq). Estos datos se pueden observar en la Figura I. 1. También se puede visualizar la temperatura y humedad en la pantalla LCD.

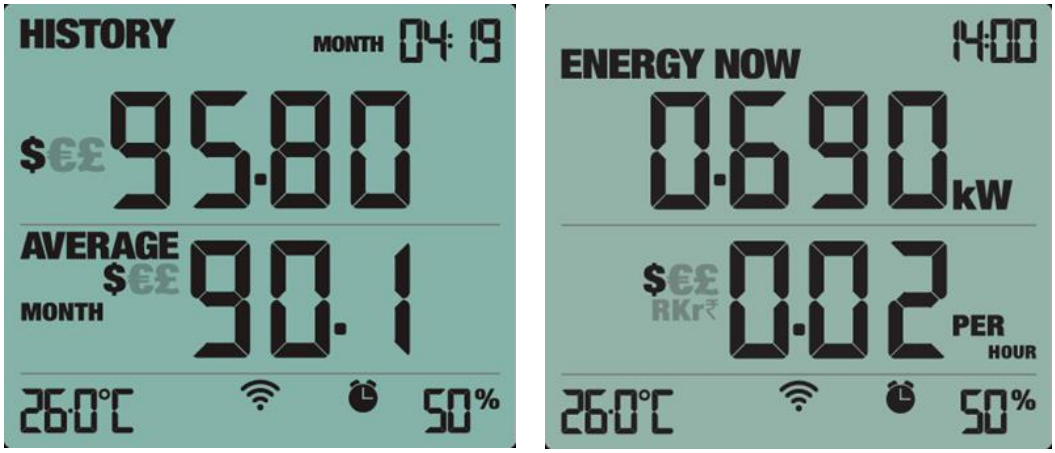


Figura I. 1: Información histórica y promedios de consumo (Izquierda). Información instantánea de consumo en kW [29].

Se debe configurar la fecha y hora, al igual que la tarifa de electricidad contratada, para ello, se aconseja tener la factura eléctrica a mano para modificar los valores por defecto del monitor.

- **Tensión:** el valor por defecto es de 240 V.
- **Moneda:** diferentes tipos de monedas ($, €, £, R, Kr, ₹).
- **Tarifa:** se pueden seleccionar hasta 4 tipos de tarifas diferentes.
    - Tarifa única: Por defecto el monitor está configurado sin discriminación horaria y el precio por defecto es de 0,14 €/kWh.
    - Tarifa con discriminación horaria: Se debe configurar los periodos de inicio y fin de cada tramo de tarifa, estableciendo así dos tramos de tarifa. El precio por defecto de la tarifa 1 es de 0,15 €/kWh y 0,05 €/kWh para la segunda tarifa.
- **Ratio de emisiones de carbono:** es la cantidad de emisiones que se emiten a la atmósfera para producir una unidad de electricidad (1kWh). La ratio de emisiones europeo es aproximadamente de 0,50 kg CO_2eq/kWh que es el valor que aparece por defecto.

- **Alarma – Potencia Máxima:** El usuario establece un límite de potencia según sus criterios. La alarma sonará cuando la potencia supere dicho valor máximo.

2) Sensor Mini o XL

El Elite Classic es fácil de instalar. Como muestra la Figura I. 2 se debe acoplar el sensor en el cable de fase del cuadro eléctrico de la vivienda y conectarlo al transmisor, éste envía de forma inalámbrica la información a la pantalla del monitor, que muestra la cantidad de electricidad que se está consumiendo en cada instante. Para localizar el cable de fase en el cuadro eléctrico hay que fijarse en el diámetro de los cables y su color. Normalmente el cable de fase es más grueso y de color negro, gris o marrón. Si el suministro fuera trifásico se necesitarían dos sensores adicionales.

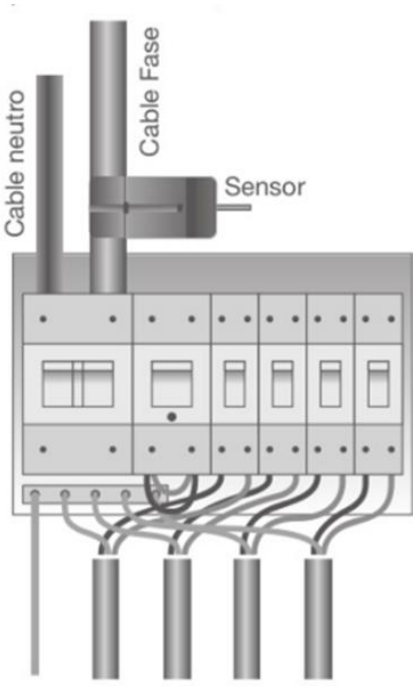


Figura I. 2: Instalación del sensor [30].

Compatible con instalaciones de paneles solares, para medir la energía que general los paneles solares, acopla el sensor alrededor del cable de alimentación del sistema solar. Las características de este sensor son las siguientes:

- Diámetro interno Sensor Mini: Max. 12mm.
- Diámetro interno Sensor XL: Max. 19Mm.
- Medición de corriente Sensor Mini: 90A-120ª.
- Medición de corriente Sensor XL: 120A-200ª.
- Rango de voltaje: 110V-300V.

3) Transmisor.

Este dispositivo (Figura I. 3) monitoriza la energía en tiempo real enviando la información de forma inalámbrica al *Hub* y al monitor cada diez segundos. Para cada instalación se necesitará una sonda y un transmisor adicional. Las características fundamentales del transmisor son:

- Alimentación a través de 3 baterías AAA.
- Frecuencia: 433,5 MHz.
- Rango de transmisión de hasta 71 m en campo abierto.

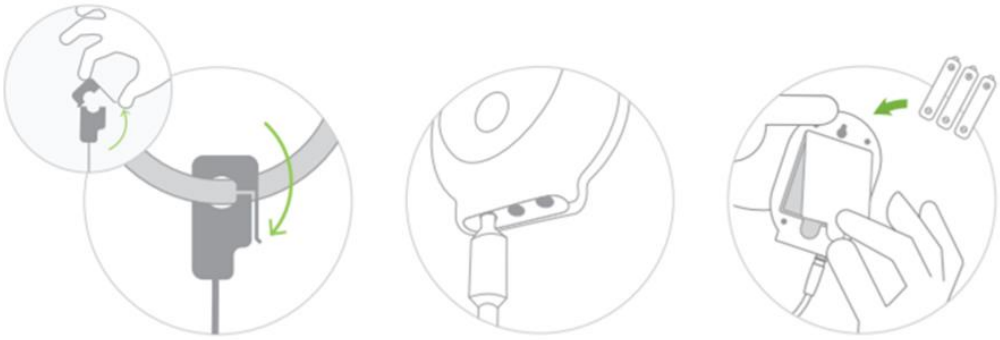


Figura I. 3: Conexión del sensor al cable de fase (Izquierda). Conexión del sensor al transmisor (Centro). Alimentación del transmisor (Derecha) [31]

4) Engage Hub Solo

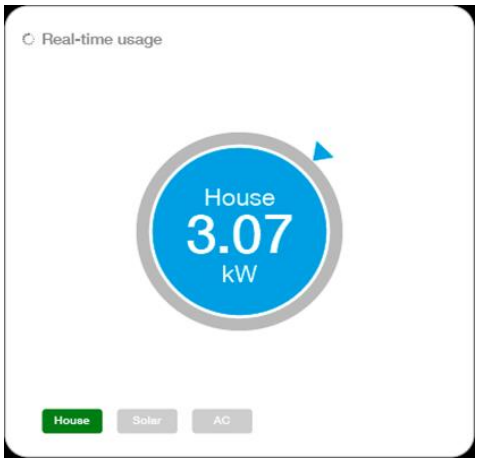
Este dispositivo es un complemento para gestionar online el monitor de energía *Efergy*. Proporciona una interfaz completa y detallada para que el usuario observe el consumo de energía de la vivienda.

Es recomendable colocar el *Hub* al lado o cerca del router y a una distancia que no supere los 70 metros con el transmisor, ya que es la distancia máxima para que la comunicación entre Hub y transmisor se produzca. El dispositivo se conecta a la alimentación con el adaptador AC/DC y una luz LED roja nos indicará que éste recibe corriente. Además, se debe conectar al router a través de un cable ethernet, representado en la Figura I. 4. Una vez conectado, una luz amarilla parpadeará, indicando que el *Hub* está buscando un transmisor dentro del rango.

Figura I. 4: Conexión del Hub al router [31].

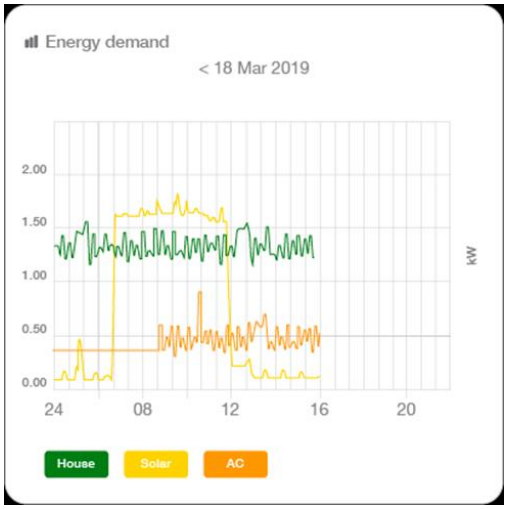
Para visualizar los datos online, el usuario debe registrase o entrar con su cuenta de usuario en engage.efergy.com. Esta plataforma detectará automáticamente la dirección MAC del *Hub*. Si el consumidor tiene que crear una cuenta, deberá rellenar distintos campos como el tipo de sensor, transmisor, la tensión, sus datos personales (nombre, apellidos, código postal y país) y la moneda. Cuando se haya realizado toda la configuración, el sensor y transmisor se emparejarán, el *Hub* se conectará al WiFi de la vivienda y, en ese momento, se podrán visualizar el consumo de energía a tiempo real mostradas en la Figura I. 5.

a)



b)

c)



d)

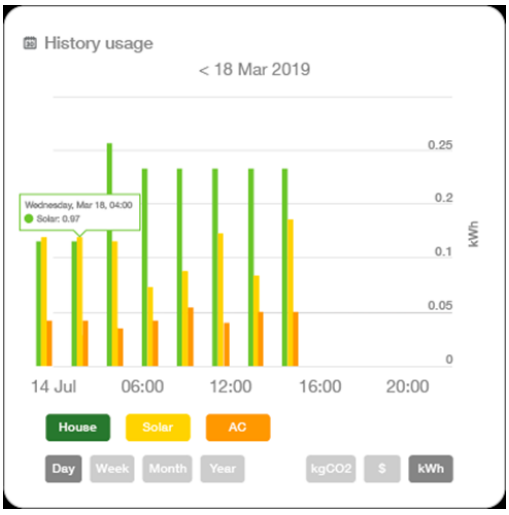


Figura I. 5: a) Consumo en tiempo real. b) Presupuesto y costes. c) Demanda de energía. d) Consumo histórico [32].

I.1.2 Medidores similares a Efergy Elite Classic.

En la Tabla I. 1 de este anexo, se han enumerado algunos de los distintos tipos de medidores de consumo eléctrico que podemos encontrar hoy en día. En este caso, se presentan medidores similares al Efergy Elite Classic ya que son aquellos que envían y/o almacenan los datos en un servidor web o aplicación donde se puedan visualizar y gestionar.

I.1.2.1 Medidor de energía Mirubee Mirubox MONO.

Es un analizador de consumos en tiempo real con conexión inalámbrica via WiFi, que muestra los datos instantáneos e históricos del consumo eléctrico mediante cualquier dispositivo smartphone, tablet o PC, con la ayuda de su App o del servidor web integrado. Además, es capaz de diferenciar el consumo de cada electrodoméstico por separado (virtual submetering[4]) y así facilita al usuario observar que aparatos consumen más. La instalación es similar al del Efergy Elite Classic ya que contiene una pinza (sonda) con un diámetro interior de 10 mm y corriente nominal de 60 A. Además, cuenta con la versión monofásica y trifásica.

[4] Identifica el consumo individual de múltiples electrodomésticos usando un único medidor general y un software inteligente. El algoritmo busca saltos de potencia consumida (corresponde al encendido o apagado de aparatos) y otras características particulares de cada electrodoméstico. Por lo tanto, lo que se mide es real, no es estimado ni estadístico.

Estos dispositivos no necesitan pilas ya que se alimentan directamente desde el cuadro eléctrico, con una conexión a los bornes eléctricos mediante imanes. Los datos del consumo se irán enviando a Internet por WiFi o Tarjeta SIM (NB-IoT), según el modelo. La plataforma permite visualizar, como se observa en la Figura I. 6, el consumo eléctrico total del hogar, así como de cada electrodoméstico y detectar posibilidades de ahorro. El precio de este monitor no se ha podido encontrar.

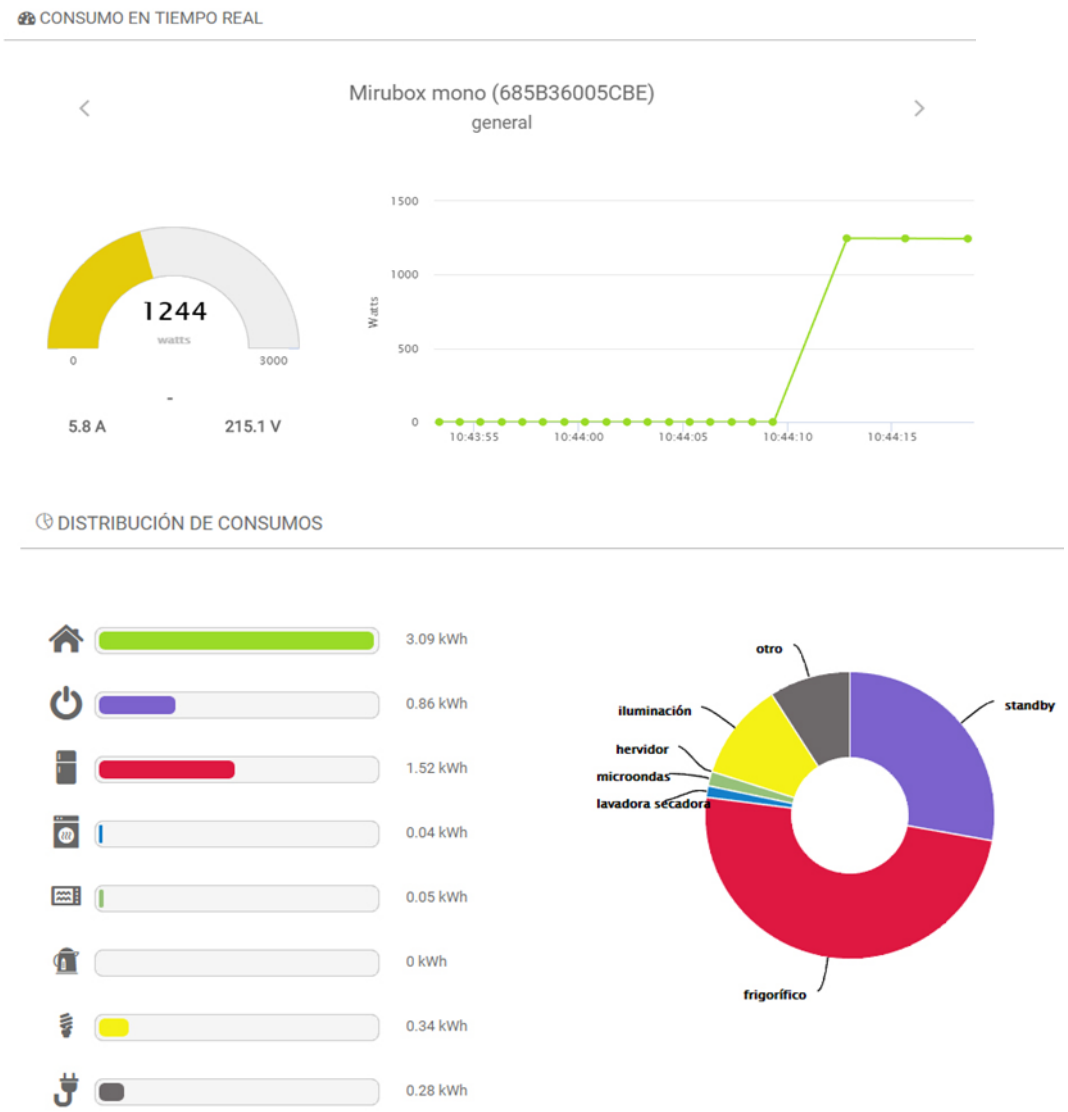


Figura I. 6: Visualización de la potencia (arriba). Visualización del consumo de cada electrodoméstico (abajo) [32].

I.1.2.2 Monitor Energético OWL CM160 USB

Este dispositivo es muy similar al Efergy Elite Classic ya que está compuesto por un sensor, un transmisor y una pantalla LCD. La instalación es idéntica, cada sensor se conecta a una fase del cuadro eléctrico de la vivienda y éstos al transmisor. El monitor muestra el consumo (kWh), el coste (€) y la huella de carbono (kg CO2). Se alimenta a través de 3 pilas AA con una duración de 2 años. Los datos son enviados por el transmisor al monitor vía WiFi, éstos no pueden estar separados más de 30 m. Una vez instalado y transcurridos 30 días, se pueden transferir los datos al ordenador a través de un mini-USB conectado al monitor. Éstos se descargarán automáticamente en la base de datos de OWL Connect2. El precio de este monitor es de 104 € [33].

I.1.2.2 Smapee

Este dispositivo, como se ha comentado brevemente la Tabla I. 1, permite conocer al instante el consumo total y el de cada electrodoméstico del hogar desde un móvil. Se instala mediante un sensor en el cable de fase del cuadro eléctrico conectado al aparato, que se alimenta a la red. Dicha instalación se muestra en la Figura I. 7, izquierda. Éste a su vez, se conecta a la red WiFi para enviar los datos que se almacenan en la cuenta del consumidor y así acceder desde el móvil o Tablet.

La aplicación muestra el consumo y coste de cada electrodoméstico y resúmenes visuales del consumo por horas o días (Figura I. 7, derecha). El precio de este medidor de consumo ronda los 200€ dependiendo de los vendedores.

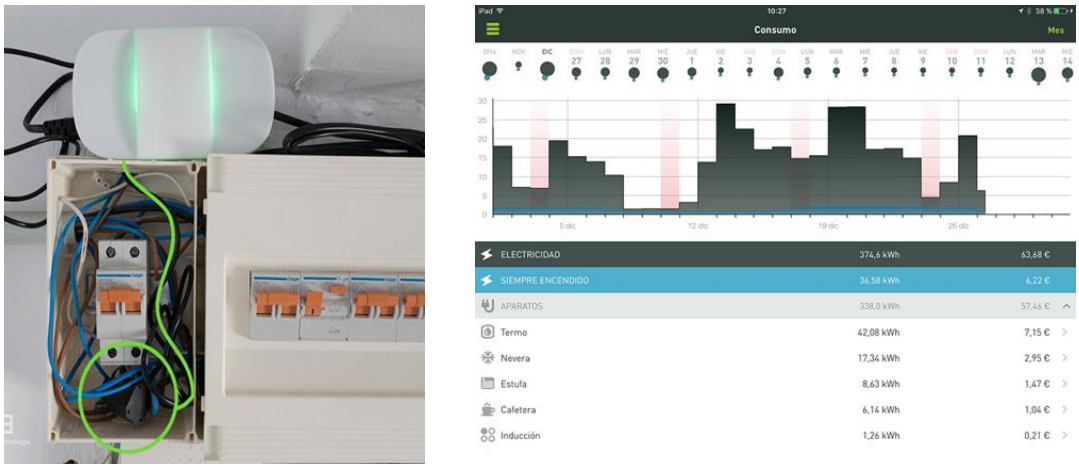


Figura I. 7: Instalación del sensor (izquierda). Visualización de los datos de consumo (derecha) [34].

II. Anexo 2: Microcontroladores utilizados.

II.1 ESP8266

Este microcontrolador está diseñado por la compañía *Espressif Systems* y es idóneo para las aplicaciones IOT por su bajo coste y sus características tal y como se muestra en la Tabla II. 1.

Voltaje	3,3 V
Consumo de corriente	10 µA – 170 mA
Memoria Flash	16 MB máx.
Procesador	Tensilica L106 32 bit
Velocidad de procesador	80 – 160 MHz
GPIOs	11 pines
Analógico a digital	1 entrada con 10 bit de resolución (1024 valores).

Tabla II. 1: Características generales [35].

La tarjeta escogida es el modelo WeMos D1 mini y en la siguiente imagen (Figura II. 1) junto a un esquema de los pines de entrada y salida del microcontrolador.

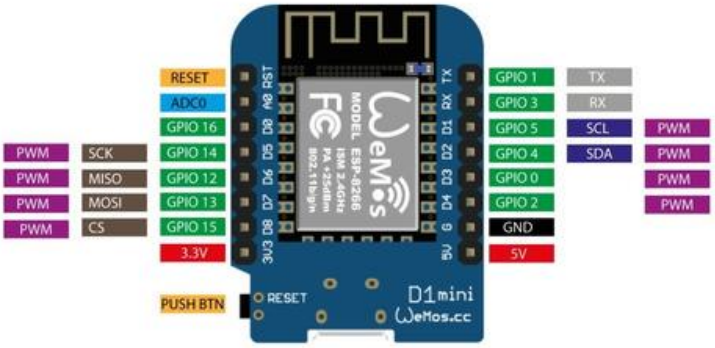


Figura II. 1: Esquema de pine E/S [36]

II.2 ESP32

Este microcontrolador al igual que el descrito anteriormente está diseñado por *Espressif Systems*, una empresa china situada en Shanghái. Al igual que la tarjeta ESP8266, éste dispone de varios modelos con diferentes características.

Esta serie está definida por la propia empresa como una solución para microcontroladores que no dispongan de conectividad, ya que la familia ESP32 se podría utilizar como puente para el acceso a la red ya que dispone de módulo WiFi. Además, es capaz de ejecutar sus propias aplicaciones de tiempo real, haciéndolo un dispositivo muy interesante.

Voltaje	3,3 V
Procesador	Xtensa LX6 de 2 núcleos, 32 bits
Velocidad de procesador	260 MHz
GPIOs	36
Analógico a digital	2 ADCs de 12 bits

Tabla II. 2 : Características generales de ESP32 [37].

Para completar las características mostradas en la Tabla II. 2, se puede añadir que tiene 4 comunicaciones SPI, 2 de I2C, 2 de I2S y 3 de UART y Bus CAB 2.0. Además, tiene 36 pines GPIO, de los cuales 16 pueden ser utilizados como salidas PWM y 18 pines como entradas analógicas. Este esquema se puede ver en la Figura II. 2. Cuenta con dos núcleos, *core 0* que generalmente se usa para radiofrecuencia y el *core 1* es el que ejecuta el código por defecto.

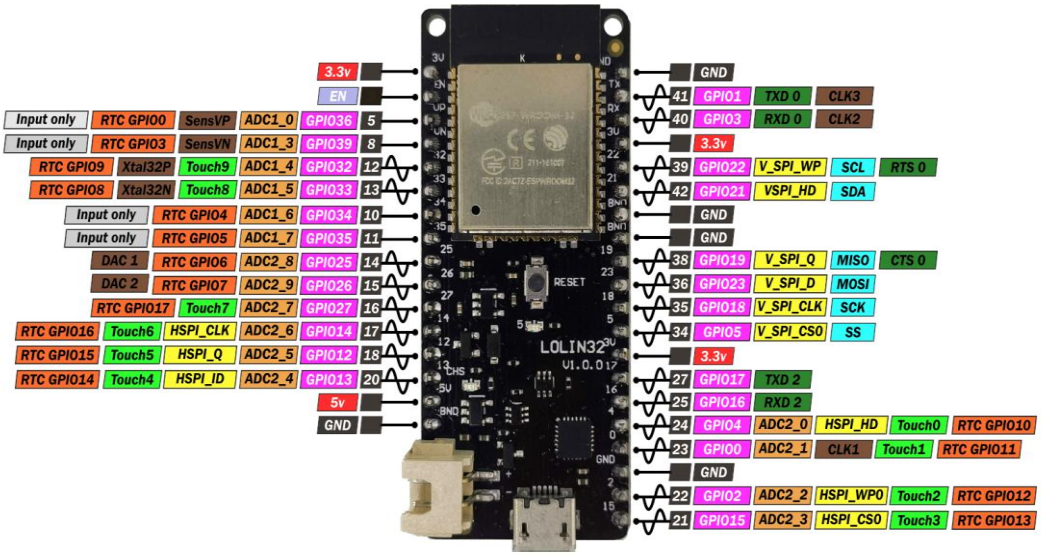


Figura II. 2: Esquema pines E/S del ESP32 [37].

II.3 Comparativa entre ambos microcontroladores

Para terminar, se realiza una comparación entre las características del microcontrolador ESP8266 y ESP32, que se muestran en la Tabla II.3.

Características	ESP8266	ESP32
Procesador	Tensilica LX106 32 bit a 80Mhz (hasta 160 MHz)	Tensilica Xtensa LX6 32 bit Dual Core a 160Mhz (hasta 240 MHz)
Memoria RAM	Hasta 4 MB	Hasta 16 MB
ROM	No	448 KB
Alimentación	3 V a 3,6 V	2,2 V a 3,6 V
Rango temperaturas	-40ºC a 125ºC	-40ºC a 125ºC
Consumo de corriente	70 mA (promedio) 225 mA máximo	70 mA (promedio) 225 mA máximo
WiFi	802.11 b/g/n (hasta +20dBm) WEP, WPA	802.11 b/g/n (hasta +20dBm) WEP, WPA
Bluetooth	No	v4.2 BR/EDR y BLE
UART	2 puertos	3 puertos
I2C	1 interfaz	2 interfaces
SPI	2 interfaces	4 interfaces
GPIO (utilizables)	32 pines	11 pines
PWM	8 canales	16 canales
ADC	1(10 bit)	2 (hasta 18 canales) (12 bit)

DAC	No	2 canales (8bit)
I2S	1 interfaz	2 interfaces
CAN 2.0	No	1 bus
Sensor de temperatura	No	Si

Tabla II.3: Tabla comparativa de ambos microcontroladores

III. Anexo 3: Herramientas software utilizadas.

III.1 *Arduino IDE*

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se trata de un entorno integrado de desarrollo de *Arduino* escrito en lenguaje de programación Java.

La ventana de código, como podemos observar en la Figura III. 1, se compone de dos funciones básicas, la primera se trata del *setup()* donde se encuentran las instrucciones de inicialización y solo se ejecuta una vez. La segunda función es el ciclo infinito *loop()* donde se encuentra la parte de programa que se va a ejecutar. Esta función se ejecuta después del *setup()* de manera repetida hasta que al microcontrolador se le quite la tensión de alimentación. En la esquina superior derecha se puede encontrar la interfaz con el puerto serie, una parte muy útil para la visualización de resultados por pantalla.

Figura III. 1: Ventana principal de la plataforma Arduino IDE [38].

Contiene numerosas librerías con sus correspondientes ejemplos. Para configurar el código realizado con el microcontrolador que quieres utilizar basta con elegir la tarjeta correspondiente en la ventana de *Herramientas* → *Placa*, se elige la tarjeta deseada y, por último, se selecciona el puerto serie donde está conectada dicha tarjeta.

Esta herramienta se ha utilizado para implementar todo el código del proyecto.

III.2 Sistema operativo *FreeRTOS.*

III.2.1 Descripción.

En este proyecto se implementa el código en *Arduino IDE* y utilizando *FreeRTOS* para un mayor control en los tiempos de lectura, cálculo y envío de datos.

FreeRTOS es un sistema operativo de tiempo real multitarea que permite administrar los recursos hardware y los tiempos de ejecución de las tareas implementadas en un microcontrolador. De este modo, permite ejecutar varias tareas paralelamente, además de multitud de servicios en segundo plano, haciendo que parezca que todo se ejecuta al mismo tiempo.

La ventaja de que *FreeRTOS* sea multitarea respecto a, por ejemplo, *Arduino* es que es capaz de dividir las tareas en unidades más pequeñas permitiéndole intercalarlas a un nivel más bajo

y aprovecha así los tiempos muertos. En cambio, si una tarea en *Arduino* tarda 200 ms, ésta bloqueará al resto hasta que se termine.

III.2.2 Tareas.

Una tarea es un programa independiente que realiza operaciones específicas utilizando uno o varios recursos del microcontrolador como por ejemplo UART, SPI, I2C, memoria RAM o la CPU.

III.2.3 Estados de una tarea.

Un microcontrolador con un solo núcleo puede ejecutar una tarea a la vez y el sistema operativo es quien se encarga de gestionar el recurso compartido (CPU) para ejecutar todas las tareas dentro de unas restricciones temporales predefinidas.

En cambio, si el microcontrolador cuenta con dos o más núcleos de procesamiento, como es en nuestro caso en la tarjeta ESP32, se podrán ejecutar dos tareas simultáneamente en los distintos núcleos, siempre y cuando los recursos necesitados por cada una de éstas estén disponibles.

Una tarea puede encontrarse en cuatro estados diferentes, esquematizados en la Figura III. 2, dependiendo de las demás o los recursos necesarios para que se ejecute:

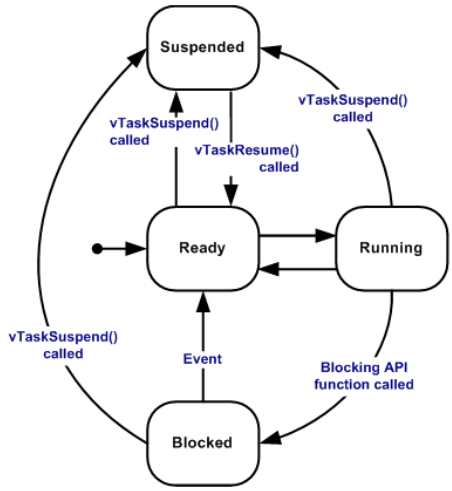


Figura III. 2: Estados de una tarea [39].

- ***Running***: la tarea se está ejecutando. En el caso de que el microcontrolador contara con un núcleo, solo una tarea podría estar en este estado.
- ***Ready***: la tarea está lista para ser ejecutada.
- ***Blocked***: una tarea puede llegar a este estado cuando está esperando por un evento temporal que generalmente es la función *vTaskDelay()* (explicada en el apartado III.2.6 de este anexo). También puede ser porque está esperando a un evento externo (interrupción) o a un recurso utilizado por otra tarea y tiene que ser liberado (colas, semáforos…). Cuando la espera termina, la tarea pasa al estado "*Ready*".
- ***Suspended***: es un estado muy similar a "*blocked*", con la diferencia de que para entrar o salir de este estado deben ser utilizadas respectivamente estas dos funciones: *vTaskSuspend()* y *vTaskResume()* (explicadas en el apartado III.2.6 de este anexo).

III.2.4 Prioridades de una tarea.

Cada tarea tiene una prioridad que se debe asignar en el momento de ser creada cuyos valores van de 0 al valor establecido en la configuración (*configMAX_PRIORITIES* – 1) que se encuentra en el archivo "*FreeRTOSConfig.h*". Esto significa que una prioridad 0 corresponde a una tarea con la prioridad más baja y 3 a aquella con mayor prioridad. Si dos tareas o más tienen la misma prioridad, pasará al estado de ejecución la primera tarea en llegar al estado "*ready*" ya que sigue una planificación *FIFO* (*First In First Out*).

En cambio, como se observa en la Figura III. 3, si se tienen dos tareas con distinta prioridad, tarea 1 cuya prioridad es menor que la prioridad de la tarea 2, en el momento que la tarea de mayor prioridad está disponible pasa a ser ejecutada. La tarea 1 tendrá que esperar hasta que las de mayor prioridad pases a estar bloqueadas o suspendidas.

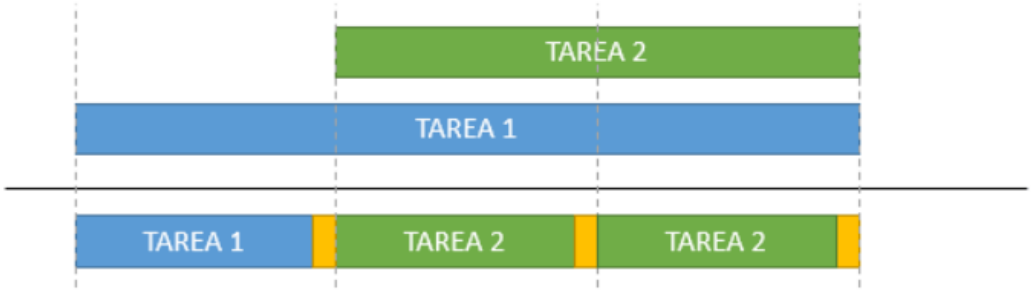


Figura III. 3: Scheduler de dos tareas con distinta prioridad [40].

III.2.5 Creación de una tarea.

Para crear una tarea se pueden utilizar dos funciones *xTaskCreate()* y *xTaskCreatePinnedToCore()*. La diferencia entre ellas es que la primera función crea una tarea libre y *FreeRTOS* la vinculará en el núcleo que quiera. En cambio, con la segunda función es posible adjudicar una tarea en el núcleo donde se quiere que se ejecute. Una tarea necesita un bucle permanente para que nunca deje de ejecutarse hasta ser eliminadas.

Ambas funciones reciben 6 parámetros comunes que son:

1) *pvTaskCode*: Nombre de la función que se define como tarea.

2) *pcName*: se indica el nombre de la tarea, principalmente para tareas de depuración. El número máximo de caracteres es de 13 caracteres por defecto, pero se puede configurar con *configMAX_TASK_NAME_LEN* en el archivo "*FreeRTOSConfig.h*".

3) usStackDepth: tamaño de la memoria RAM que hay que asignarle a la tarea. No es un valor que se conozca por lo que hay que ir probando con múltiplos de 1024. Si el microcontrolador se resetea es porque hay que aumentar este valor.

4) pvParameters: es un parámetro de tipo puntero. El valor asignado en *pvParameters* será pasado a la tarea en el momento de iniciar su ejecución.

5) uxPriority: prioridad asignada a la tarea.

6) pxCreatedTask: puede ser *NULL* si la tarea es permanente y se utiliza para devolver un identificador mediante el cual se puede hacer referencia a la tarea creada. Este parámetro se indica cuando se quiere crear una tarea y que se ejecute por un núcleo específico, por lo tanto, se incluirá como parámetro en la función *xTaskCreatePinnedToCore()*.

7) *xCoreId*: parámetro que indica a qué núcleo se le asigna cada tarea.

Ejemplo de la creación de una tarea asignada al núcleo 1

```
xTaskCreatePinnedToCore(
    TaskBlink,
    "Taskname",
    1024,
    NULL,
    1,
    NULL,
    1
  );
```

III.2.6 Otras funciones.

- *vTaskDelay():* esta función pausa las tareas y en lugar de mantener al microcontrolador ocupado hasta que termine este instante de tiempo, devuelve el control al Sistema Operativo para que aproveche este tiempo en otras tareas. Requiere de un solo parámetro, los ticks, que son la cantidad de divisiones de tiempo realizados por *FreeRTOS*. Cada división suele ser de 10 ms en la tarjeta ESP32 pero se puede utilizar pdMS_TO_TICKS() que calcula el número de ticks correspondientes a un tiempo en ms.

```
// Delay the task 500 ms
   vTaskDelay(pdMS_TO_TICKS(500));
```

- *vTaskSuspend():* permite pausar una tarea indefinidamente sin llegar a eliminarla, por lo que no usará tiempo del procesador y podremos pasar al estado ready.

```
void MyTaask(void *pvParameters) {
// Pause the task just at startup.
vTaskSuspend(NULL);
for (;;) { // A Task shall never return or exit.
...
}
}
```

- *vTaskResume():* hace que la tarea pase del estado suspendida a estar lista para ejecutarse. Para usar esta función se necesita crear un handler para dicha tarea ya que al estar pausada no se ejecutará dentro de la propia tarea.

Colas: las colas son buffers de tipo FIFO, que permiten intercambiar variables entre diferentes tareas.

- *xQueueSend():* una vez creada la cola, con esta función se puede enviar objetos al final de ésta, los cuales deben ser del mismo tipo que se ha indicado a la hora de crear la cola. La función devuelve pdTRUE si el valor ha sido enviado correctamente o errQUEUE_FULL si la cola está llena.

```
// Create a queue of 10 object, with a size of uint32_t
QueueHandle_t xQueue1 = xQueueCreate( 10, sizeof( uint32_t ) );
uint32_t ulVar = 10UL;
if( xQueue1 != 0 ) {
  // Queue is correct
```

```
    if (xQueueSend( xQueue1, ( void * ) &ulVar, ( TickType_t ) 0 ) == pdTRUE)
  {
      // The message was sent sucessfully
    }
  }
```

III.3 Librería *EmonLib*.

En este apartado se explica la librería *EmonLib* que se ha utilizado para el cálculo de los parámetros necesarios para obtener el consumo eléctrico.

III.3.1 Descripción.

EmonLib es una librería de monitorización que mide continuamente en segundo plano tanto los valores de tensión, como todos los canales de entrada de corriente (hasta 3 canales posibles).

Esta librería proporciona una medición precisa de la tensión y la corriente eficaces, la potencia activa, aparente y el factor de potencia.

III.3.2 Estructura y explicación del código.

III.3.2.1 Emonlib.h.

Este archivo presenta la declaración de todas las variables y funciones. Lo más reseñable que podemos comentar de éste, es que la librería *EmonLib* por defecto, trabaja con una resolución de 10 bits. Esto conlleva un problema en nuestro caso, porque el ADC del microcontrolador ESP32 es de 12 bits, por lo que se añadieron unas líneas de código para fijar el ADC a 12 bits.

La explicación del objetivo de cada función se detalla en el apartado IV.3.2.2 (*Emonlib.cpp*) de este anexo.

```
// Código añadido para la configuración de la resolución del ADC a 12 bits
#if defined(ESP32)
#define ADC_BITS    12
#endif

#define ADC_COUNTS (1<<ADC_BITS)

// Declaración de las funciones y variables
class EnergyMonitor
{
  public: // Funciones y variables públicas
    // Instancias de tension y corriente
    void voltage(unsigned int _inPinV, double _VCAL, double _PHASECAL);
    void current(unsigned int _inPinI, double _ICAL);

    void calcVI(unsigned int crossings, unsigned int timeout); // Calcula la
potencia real, la potencia aparente, el factor de potencia y la tensión
eficaz.
    double calcIrms(unsigned int NUMBER_OF_SAMPLES); // Cálculo de la
corriente eficaz

    void serialprint(); // Función que saca por línea serie los valores
calculados.

    long readVcc(); //valor de alimentación del microcontrolador ESP32 (3.3 V)
    //Useful value variables
    double realPower, //Potencia activa
      apparentPower,  //Potencia aparente
```

```
      powerFactor,    //Factor de potencia
      Vrms,           //Tensión eficaz
      Irms;           //Corriente eficaz

  private: // Variables privadas

    //declaración de los pines de entrada de tensión y corriente
    unsigned int inPinV;
    unsigned int inPinI;
    //Coeficientes de calibración
    double VCAL;
    double ICAL;
    double PHASECAL;

    // Lecturas de los valores de tensión y corriente desde los pines de
entrada
    int sampleV;
    int sampleI;

    double lastFilteredV,filteredV; //valor filtrado una vez restado el offset
    double filteredI;
    // Para conseguir una señal senoidal bipolar
    double offsetV;
    double offsetI;

    double phaseShiftedV;  //fase

    //sq = cuadrado, sum = sumatorio, inst = instantánea
    double sqV,sumV,sqI,sumI,instP,sumP;

    int startV;      //Instantaneous voltage at start of sample window.

    boolean lastVCross, checkVCross;  //Número de veces que se cruza el rango
(cruces por cero).
};

#endif
```

III.3.2.2 EmonLib.cpp.

Aquí se aborda la explicación del objetivo de las funciones utilizadas, los parámetros necesarios para cada una de ellas y los cálculos realizados.

Para la creación de la instancia de tensión, los parámetros que debemos pasarle son el pin de entrada de la señal, el valor de la calibración calculada del sensor de tensión y la fase. Estas dos variables se describen con más detalle en el siguiente apartado, una vez terminada la explicación del código.

```
void EnergyMonitor::voltage(unsigned int _inPinV, double _VCAL,double
_PHASECAL) //pin de entrada, calibración y fase
{
  inPinV = _inPinV;
  VCAL = _VCAL;
  PHASECAL = _PHASECAL; //fase, hay que introducirla a mano y así desplazar la
onda
  offsetV = ADC_COUNTS >> 1; //offset de tensión, ADC_COUNTS definido en el .h
como (1<<ADC_BITS), ADC_BITS= 12 bits (divide para 2)
}
```

Los parámetros necesarios para crear la instancia de corriente son el pin de entrada de la onda de corriente y la calibración calculada del sensor de corriente.

```
void EnergyMonitor::current(unsigned int _inPinI, double _ICAL) {
 //pin de entrada, calibración
 inPinI = _inPinI;
  ICAL = _ICAL;
```

```
  offsetI = ADC_COUNTS >> 1; //dividimos para 2 (como no hay cruce por cero
tendremos que restarle la mitad de su valor de pico)
}
```

La siguiente función de tipo *void()* es esencial ya que obtiene los valores necesarios para el cálculo de la energía eléctrica que son la potencia real, la potencia aparente, factor de potencia y la tensión y corriente eficaces. Los parámetros necesarios son el número de veces que la onda se acerca al valor cero y el periodo de tiempo en el que medimos. A continuación, se comentarán las líneas de código más importantes para la compresión de ésta.

```
void EnergyMonitor::calcVI(unsigned int crossings, unsigned int timeout)
//crossings y timeout lo establecemos nosotros
{
#if defined emonTxV3
  int SupplyVoltage = 3300; //mV --> 3.3 V
#else
  int SupplyVoltage = readVcc();
#endif

  unsigned int crossCount = 0;  // Número de veces que la onda pasa por cero
  unsigned int numberOfSamples = 0;  // Número de muestras que más tarde se
irá incrementando.
```

Esta función lee el valor de tensión desde su pin de entrada mediante la *analogRead(inPinV)* y establece un rango para ver los valores que se acerquen a cero. Por lo tanto, esta función no es aplicable al cálculo de potencias con corriente y tensión continuas ya que no cumpliría el rango y se quedaría en este bucle indefinidamente*:*

*if ((startV < (ADC_COUNTS * 0.55)) && (startV > (ADC_COUNTS * 0.45)))*

```
  //-------------------------------------------------------------------------
  // 1) Espera a que la onda esté cerca de cero.
  //-------------------------------------------------------------------------
  unsigned long start = millis();    //millis()-start makes sure it doesnt get
stuck in the loop if there is an error.

  while (1)
  {
    startV = analogRead(inPinV); //leemos tensión desde el pin de entrada

    if ((startV < (ADC_COUNTS * 0.55)) && (startV > (ADC_COUNTS * 0.45)))
break;
    //check its within range, rango para ver cuando se acerca al paso por cero
    if ((millis() - start) > timeout) break;
  }

  //-------------------------------------------------------------------------
  // 2) Main measurement loop
  //-------------------------------------------------------------------------
  start = millis();

  while ((crossCount < crossings) && ((millis() - start) < timeout)) //cada
cruce y que cumpla el tiempo el número de muestras incrementa en 1
  {
    numberOfSamples++;                       //Count number of times looped.
    lastFilteredV = filteredV;               //Used for delay/phase
compensation

   // Leemos valores de tension y corriente desde los pines de entrada
    sampleV = analogRead(inPinV);                 //Read in raw voltage signal
    sampleI = analogRead(inPinI);                 //Read in raw current signal
```

El cálculo del *offset* es muy importante porque en este proyecto se parte de dos ondas de tensión y corriente unipolares. Mediante el *offset* conseguimos transformarlas en dos ondas bipolares.

```
offsetV = offsetV + ((sampleV - offsetV) / ADC_COUNTS);
filteredV = sampleV - offsetV;
offsetI = offsetI + ((sampleI - offsetI) / ADC_COUNTS);
filteredI = sampleI - offsetI;
```

El valor eficaz, tanto de corriente como de tensión, se define como la raíz cuadrada de la media de los valores instantáneos al cuadrado, que varían en el tiempo.

$$X_{rms} = \sqrt{\frac{x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2}{Nn}}$$

X_{rms} = tensión o corriente eficaz.
X_n = tensión o corriente instantánea.
Nn = número de muestras.

Para ello, se hacen los cuadrados de los valores tanto de tensión y corriente ya habiendo tenido en cuenta el *offset* y en ambos casos se acumulan esos valores en las variables *sumV* y *sumI* para su posterior utilización.

```
// C) Root-mean-square method voltage

sqV = filteredV * filteredV;                //1) square voltage values
sumV += sqV;                                //2) sum

// D) Root-mean-square method current
sqI = filteredI * filteredI;                //1) square current values
sumI += sqI;                                //2) sum

// E) Phase calibration
phaseShiftedV = lastFilteredV + PHASECAL * (filteredV - lastFilteredV);
```

El cálculo de la potencia instantánea es el producto del valor de tensión filtrado, aplicando la corrección de la fase, y el valor de corriente también con el *offset* aplicado. Una vez hecho esto, se suman y acumulan estos resultados en la variable *sumP*.

```
// F) Instantaneous power calc

instP = phaseShiftedV * filteredI;
sumP += instP;                              //Sumatorio de la potencia
```

La siguiente función comprueba el número de veces que la onda de tensión ha cruzado el valor de la tensión inicial (*startV*). Se puede afirmar que cada vez que se crucen se habrá muestreado medio periodo.

```
// G) Find the number of times the voltage has crossed the initial voltage

lastVCross = checkVCross;
 if (sampleV > startV) checkVCross = true;
 else checkVCross = false;
 if (numberOfSamples == 1) lastVCross = checkVCross;

 if (lastVCross != checkVCross) crossCount++;
}
```

El último paso de esta función es la utilización de la calibración calculada para cada sensor *VCAL* e *ICAL* obteniendo un valor *V_RATIO* e *I_RATIO* respectivamente, que sirve para el cambio de valores digitales a analógicos. Aplicando este índice se calcula la tensión eficaz (Vrms), la corriente eficaz (Irms), potencia real (P = el sumatorio de las potencias instantáneas/ número de muestras), potencia aparente (S = Vrms·Irms) y el factor de potencia (PF=P/S).

```
 // 3) Post loop calculations
 //Calculation of the root of the mean of the voltage and current squared
(rms)
  //Calibration coefficients applied.

  double V_RATIO = VCAL * ((SupplyVoltage / 1000.0) / (ADC_COUNTS));
  Vrms = V_RATIO * sqrt(sumV / numberOfSamples);

  double I_RATIO = ICAL * ((SupplyVoltage / 1000.0) / (ADC_COUNTS));
  Irms = I_RATIO * sqrt(sumI / numberOfSamples);

  //Calculation power values
  realPower = V_RATIO * I_RATIO * sumP / numberOfSamples;
  apparentPower = Vrms * Irms;
  powerFactor = realPower / apparentPower;

  //Reset accumulators
  sumV = 0;
  sumI = 0;
  sumP = 0;
}
```

Hay una función independiente que nos devuelve el cálculo de la corriente eficaz (Irms) que sigue los mismos pasos que en el caso de la obtención de la tensión eficaz. Hay que pasarle como parámetro el número de muestras que es un valor que elige el usuario.

```
double EnergyMonitor::calcIrms(unsigned int Number_of_Samples)
{

#if defined emonTxV3
  int SupplyVoltage = 3300;
#else
  int SupplyVoltage = readVcc();
#endif

  for (unsigned int n = 0; n < Number_of_Samples; n++)
  {
    sampleI = analogRead(inPinI);

    // Digital low pass filter extracts the 2.5 V or 1.65 V dc offset,
    // then subtract this - signal is now centered on 0 counts.
    offsetI = (offsetI + (sampleI - offsetI) / ADC_COUNTS);
    filteredI = sampleI - offsetI;

    // Root-mean-square method current
    // 1) square current values
    sqI = filteredI * filteredI;
    // 2) sum
    sumI += sqI;
  }

  double I_RATIO = ICAL * ((SupplyVoltage / 1000.0) / (ADC_COUNTS));
  Irms = I_RATIO * sqrt(sumI / Number_of_Samples);

  //Reset accumulators
  sumI = 0;

  return Irms;
}
```

La función *calcVI*(), como se ha comentado anteriormente, es de tipo *void*(), no devuelve ningún valor. Para ello, se ha modificado sutilmente el código de la librería para que imprima cada 100 ms por la línea serie, el nombre del parámetro calculado, su valor con tres decimales y las unidades correspondientes.

```
void EnergyMonitor::serialprint() //imprime por pantalla los valores, nombre,
3 decimales y unidades
{
  Serial.print("Real Power: ");
  Serial.print(realPower, 3);
  Serial.print("W");
  Serial.print("\tApparent Power: ");
  Serial.print(apparentPower, 3);
  Serial.print("VA");
  Serial.print("\tVrms: ");
  Serial.print(Vrms, 3);
  Serial.print("V");
  Serial.print("\tIrms: ");
  Serial.print(Irms, 3);
  Serial.print("A");
  Serial.print("\tPower Factor: ");
  Serial.println(powerFactor, 3);

  delay(100);
}
```

La librería completa de *EmonLib* se puede consultar en:
https://github.com/openenergymonitor/EmonLib

III.4. Códigos completos *Arduino IDE.*

En este apartado se indican las direcciones web a la plataforma GitHub donde se encuentran los códigos completos. URL general: https://github.com/anagarmaes/TFGCode

III.4.1. Síntesis de las ondas de tensión y corriente.

https://github.com/anagarmaes/TFGCode/tree/main/S%C3%ADntesis%20ondas%20de%20tensi%C3%B3n%20y%20corriente

III.4.2. Cálculo de los parámetros deseados con la librería *Emonlib*.

https://github.com/anagarmaes/TFGCode/tree/main/C%C3%A1lculos%20con%20librer%C3%ADa%20Emonlib

III.4.3. Cálculo del consumo eléctrico y envío de datos a *Emoncms*.

https://github.com/anagarmaes/TFGCode/tree/main/C%C3%A1lculo%20y%20env%C3%ADo%20de%20datos%20a%20Emoncms

III.4.4. Simulación del consumo eléctrico diario.

https://github.com/anagarmaes/TFGCode/tree/main/Simulaci%C3%B3n%20consumo%20el%C3%A9ctrico%20diario

III.5. *Scripts* completos *Matlab.*

Directorio donde se encuentran los scripts completos de Matlab para el cálculo de los parámetros necesarios para la obtención del consumo eléctrico imponiendo los valores reales a partir de las potencias normalizas en España.

El código donde se calculan los valores reales imponiendo los distintos valores dependiendo de la potencia normalizada contratada. Se pueden observar tres *scripts* que muestran la generación mediante código de las ondas de tensión y corriente sin armónicos, generadas

introduciendo los dos primeros armónicos y el último combinando los tres primeros armónicos. En cada script se calculan los cuatro casos dependiendo de la potencia contratada: P1 = 1150 W (caso1), P2 = 2300 W (caso 2), P3 = 3450 W (caso 3) y P4 = 4600 W (caso 4).

Url: https://github.com/anagarmaes/TFGCode/tree/main/Scripts%20Matlab

III.6. Código *FreeRTOS*.

Dirección web donde se expone el código implementado para el estudio de las tareas en *FreeRTOS* y la ejecución de éstas en los distintos núcleos disponibles en el microcontrolador ESP32.

En este url: https://github.com/anagarmaes/TFGCode/tree/main/FreeRTOS

IV. Anexo 4: *Open Energy Monitor: Emoncms.*

En este anexo, se explica brevemente las características más importantes que se han tenido en cuenta a la hora de subir los datos deseados a la nube [41]:

- ***Input*:** representan los datos de entrada. En este caso serían la potencia activa, potencia aparente, tensión eficaz, corriente eficaz, el factor de potencia y la energía eléctrica calculadas por la tarjeta ESP32 y la librería *Emonlib*. Se registra el último valor y la hora de entrada. Para registrar datos históricos se debe crear un *feed* a partir de una entrada.
- ***Input Node*:** Es un identificador para una entrada (*input*) o un *feed*.
- ***Feed*:** lugar donde se registran los datos de forma periódica. Se pueden almacenar de dos maneras: 1) *PHPFina* (*Emoncms Fixed Interval TimeSeries*) que almacena los datos durante intervalos fijos de tiempo. El registro de datos puede variar entre 10 segundos y un 1 día. 2) *PHPTimeseries (Emoncms Variable Interval TimeSeries)* la cual sube los datos de forma no periódica a la nube. En este proyecto, se ha utilizado la primera opción para el envío de valores a la nube aprovechando que es periódico.
- ***API Key (Application programming interface)*:** una vez registrado el usuario en *Emoncms, se le* adjudica una clave lectura o de lectura y escritura, llamada *API Key.* Es un identificador único para cada usuario por lo que, a la hora de subir los datos a la nube, la plataforma identifica el proyecto al que está asociada la *API Key*.
  Para el intercambio de información entre la aplicación web y los datos que se deben tratar, se ha utilizado el *API Key* de lectura y escritura ya que deben leer los parámetros calculados por la librería *Emonlib* y escribirlos en la plataforma *Emoncms* para su posterior visualización [42] .
- ***Input API Help*:** *Emoncms* ofrece tres formas de enviar los datos, aunque en este caso solamente se ha utilizado el tipo *input/post* que publica los datos desde un nodo como una estructura de datos *json* [43].
  Por lo tanto, en la dirección web deben aparecer los datos a enviar en formato *json* y el *API Key* de lectura y escritura. A continuación, se presenta un ejemplo para la creación de tres entradas en el nodo *emontx* llamadas *power1*, *power2* y *power3* con sus respectivos valores y en formato *json* (100, 200 y 300 respectivamente). Se observa que no puede faltar la clave identificadora para hacer referencia al proyecto al que se quieren enviar los datos (a946c2b86c727bffee87ce3d4449af3e).

  https://emoncms.org/input/post?node=emontx&fulljson={"power1":100,"power2":200,"power3":300} ***&apikey=a946c2b86c727bffee87ce3d4449af3e***

- **Módulo gráfico:** *Emoncms* permite la creación de gráficos, mostrando los datos enviados a través de las entradas y los *feeds,* creados de forma temporal. Se pueden visualizar los valores durante una hora, 6 horas, un día, una semana o incluso de forma mensual y anual.

V. Anexo 5: Tablas Excel para el cálculo del error.

Los datos recogidos para el cálculo del error se recogen en tablas Excel en el directorio:
https://github.com/anagarmaes/TFGCode/tree/main/Tablas%20Excel

V. Anexo 5: Tablas Excel para el cálculo del error.

VI. Anexo 6: Desglose del coste de los componentes utilizados.

En la Tabla VI. 1, se muestra el precio aproximado de cada componente utilizado para el diseño del prototipo:

Descripción	Identificación	Proveedor	Unidades	Precio unitario (€)	Total (€)
Protoboard	MB102 Protoboard	Amazon	1	4,79 €	4,79 €
Microcontrolador ESP8266	ESP8266 WeMos D1 mini	AliExpress	1	1,89 €	1,89 €
Microcontrolador ESP32	ESP32 Wemos Lolin32	Tiendatec	1	7,95 €	7,95 €
Resistencia	Resistencia	Farnell	4	0,15 €	0,6 €
Condensador	Condensador	Farnell	2	0,718 €	1,436 €
Amplificador operacional	MCP601 – E/P	Farnell	2	0,758 €	1,516 €
Cables	EL2201	Electrio	10	1,50 €	1,5 €
			TOTAL (€)		19,7 €

Tabla VI. 1: Desglose de precios de los componentes utilizados.

Se puede concluir que el prototipo de medición desarrollado es de bajo coste.

VII. Anexo 7: Tiempo invertido en las distintas partes del proyecto.

Este proyecto se inició en el curso académico 2022/2023 dentro del Departamento de Informática e Ingeniería de Sistemas de la Universidad de Zaragoza. La dedicación aproximada durante los primeros meses (desde septiembre a enero) en la realización del trabajo fin de grado han sido 1.5 h diarias (7,5 h semanales) ya que compaginaba dicho proyecto con una beca de formación. Una vez terminada dicha beca se pudo dedicar más tiempo al trabajo, alrededor de 4 horas diarias (20 h semanales). La Tabla VII. 1 expone cuántas horas ha requerido cada uno de los objetivos de este proyecto.

Tarea	**Dedicación (h)**
Elaboración de la propuesta a realizar.	4 h
Investigación de los medidores presentes en el mercado.	5 h
Estudio de los conceptos eléctricos.	5 h
Estudio de los microcontroladores a utilizar.	15 h
Diseño de la placa protoboard.	10 h
Generación de ondas de tensión y corriente.	35 h
Estudio de las distintas librerías en *Arduino IDE*.	30 h
Implementación de código en *Arduino IDE.*	45 h
Implementación de código en *Matlab.*	20 h
Estudio del sistema operativo *FreeRTOS.*	15 h
Cálculo y comprobación de resultados.	65 h
Envío de datos a la nube.	40 h
Redacción de la memoria del proyecto.	20 h
Total	309 h

Tabla VII. 1: Horas dedicadas a cada parte del proyecto.