Universidad Nacional de San Luis
Facultad de Ingeniería y Ciencias Agropecuarias

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA LA ADQUISICIÓN, TRANSMISIÓN Y MONITOREO REMOTO DE DATOS

Enrique Nicolás Martínez

Trabajo final de Ingeniería Electrónica

Director
Walter Adrián Lucero

Codirector
Guillermo Ricardo Catuogno

Asesor
Guillermo Luis Acosta

Villa Mercedes, San Luis
2025

DERECHO DE AUTOR

Agradecimientos

Quiero agradecer principalmente a mi familia, por todo su apoyo durante la carrera, a mis compañeros a lo largo de estos años por su constante ayuda y compañerismo, a los miembros del Laboratorio de Tecnologías Apropiadas por proporcionar un espacio para la experimentación y el aprendizaje y a aquellos profesores de la Facultad de Ingeniería y Ciencias Agropecuarias que con su vocación por la enseñanza me permitieron llegar hasta este lugar.

Resumen

En el presente trabajo se describe el diseño e implementación de un sistema orientado a medir variables eléctricas en instalaciones que hacen uso energías renovables, transmitir esos datos adquiridos a través de largas distancias usando una tecnología de comunicación de baja potencia y facilitar su visualización remota mediante una aplicación para teléfonos celulares. Se comenzó este proyecto diseñando los circuitos de medición de tensión y corriente, adaptando cada uno a los valores de funcionamiento normal del sistema al que se incorporarán. Posteriormente, se desarrolló el programa para adquirir los datos mediante un microcontrolador, que a su vez permitirá enviarlos hacia el módulo transceptor de radiofrecuencia que se usará como transmisor. Luego de configurar los módulos de radiofrecuencia para lograr la comunicación a la distancia requerida con el menor consumo de energía posible, se procedió a desarrollar el programa del receptor, en donde un segundo microcontrolador gestionará la recepción y enviará la información a una base de datos. Además, se llevó a cabo la programación de una aplicación en donde poder visualizar y exportar los datos adquiridos con facilidad. Para comprobar el correcto funcionamiento del sistema de medición se realizó la caracterización del prototipo y, mediante una prueba de distancia máxima, se verificó que el rango en el que es posible mantener una comunicación es adecuado para la aplicación. Finalmente, se procedió con la construcción y soldadura de los prototipos y se fabricaron carcasas para proteger aquellos componentes que se encontrarán en el exterior.

Palabras clave: adquisición de datos, Internet de las Cosas, LoRa, mediciones electrónicas, protocolos de comunicación, sistemas inalámbricos.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO 1: Propuesta

El propósito fundamental de un sistema de comunicaciones electrónico es transmitir información de un lugar a otro. Las comunicaciones electrónicas se pueden resumir como la transmisión, recepción y procesamiento de información entre dos o más ubicaciones utilizando circuitos electrónicos [1].

En tiempos modernos la urbanización avanza a pasos agigantados, generando cambios significativos en nuestra forma de vida. Un tópico en las ciudades, industrias y viviendas es el Internet de las Cosas (más conocido por sus siglas en inglés, IoT). Este concepto hace referencia a una red de dispositivos, máquinas e incluso seres vivos interconectados, que tienen la capacidad de intercambiar información entre ellos sin una interacción directa con el ser humano. La idea anterior da pie a la concepción de un nuevo enfoque para el desarrollo de ciudades inteligentes y sostenibles, en las que se debe abordar los nuevos desafíos que surgirán mediante la integración de tecnologías innovadoras y la optimización de los recursos. De manera similar, las áreas rurales también están siendo impactadas por avances tecnológicos que mejorarán la eficiencia de los procesos agrícolas [2].

LoRa (del acrónimo en inglés *Long Range*) es una tecnología inalámbrica que utiliza una modulación de espectro ensanchado, desarrollada por la empresa Cycleo en Francia. El uso de este tipo de modulación brinda una mejor tolerancia al ruido electromagnético, permitiendo de esta forma alcanzar largas distancias con un consumo muy bajo de energía [3]. Se presenta como una excelente alternativa a las comunicaciones por medio de las ya congestionadas redes WiFi o Bluetooth y en zonas alejadas de las urbes donde la conectividad a redes 3G o similares es inaccesible o resulta ser muy ineficiente.

1.1 Objetivos y alcance del trabajo

Con base en los aspectos tratados anteriormente, se realizó un sistema de comunicaciones electrónico con el objetivo de adaptar y medir de señales eléctricas provenientes de un aerogenerador mediante un microcontrolador, las cuales son tensión, intensidad de corriente y, de forma indirecta, potencia en corriente continua. El aerogenerador se encuentra en el invernadero de la Facultad de Ingeniería y Ciencias Agropecuarias, como se puede ver en la Fig. 1. Mediante tecnología LoRa, los datos adquiridos son transmitidos hasta el Laboratorio de Tecnologías Apropiadas (LabTA) en el edificio principal del campus.

Una vez recibida, la información se almacena en una base de datos que permite visualizarla desde una aplicación para teléfonos celulares. A través de este proyecto se busca brindar una manera de realizar la supervisión en las distintas instalaciones eléctricas en zonas remotas que realiza el LabTA, en las que no existe cobertura de las redes tradicionales, y

Figura N° 1: Vista aérea del invernadero y el aerogenerador en el campus de la FICA

utilizando una tecnología que ofrece ventajas considerables y resulta ser escasamente investigada y utilizada en esta región.

Se establecieron los siguientes objetivos específicos para el trabajo:

- Llevar a cabo una comunicación inalámbrica entre dos microcontroladores por medio de módulos que utilicen tecnología LoRa.
- Comprobar el desempeño de los prototipos en relación a su distancia máxima de transmisión utilizando distintas configuraciones.
- Lograr que la adquisición y medición de las señales no implique un riesgo significativo para el microcontrolador ni para el módulo de transmisión. Verificar la fiabilidad de las mediciones realizadas mediante la caracterización del sistema.
- Conseguir que los prototipos sean modulares y fácilmente modificables para permitir su mejora y/o adaptación según las señales que sean necesarias medir.
- Optimizar el sistema para disminuir el consumo de energía y, de esta forma, prolongar la vida de la batería.
- Crear una aplicación para teléfonos inteligentes que permita ver de forma remota y sencilla los datos obtenidos del lado del receptor, junto a otros parámetros de los prototipos.

Este proyecto está orientado al desarrollo de prototipos experimentales aplicables a entornos educativos o de uso personal. Utiliza un microcontrolador y un módulo transceptor con tecnología LoRa para realizar una medición, con el objetivo de transmitir esta información a una distancia de alrededor de 300 m. En cuanto al desempeño del sistema de medición, no se pretende reemplazar las mediciones que se puedan realizar en sitio, sino disminuir la frecuencia con la que estas se realizan mediante datos que se puedan visualizar de forma remota para un análisis general de las variables eléctricas.

1.2 Marco teórico

1.2.1 Sistemas de comunicaciones electrónicos

Todos los sistemas de comunicaciones electrónicos están compuestos por un transmisor, un medio o canal de comunicación y un receptor. Estos elementos se muestran en un esquema básico en la Fig. 2. El proceso de comunicación comienza con un mensaje o dato (información) que se desea transmitir hacia otro lugar, el primer paso consiste en convertir esa información en una señal eléctrica. Por ejemplo, un micrófono transforma el sonido en una señal de audio electrónica; de la misma manera, una cámara convierte la información lumínica de un plano en una señal de video.

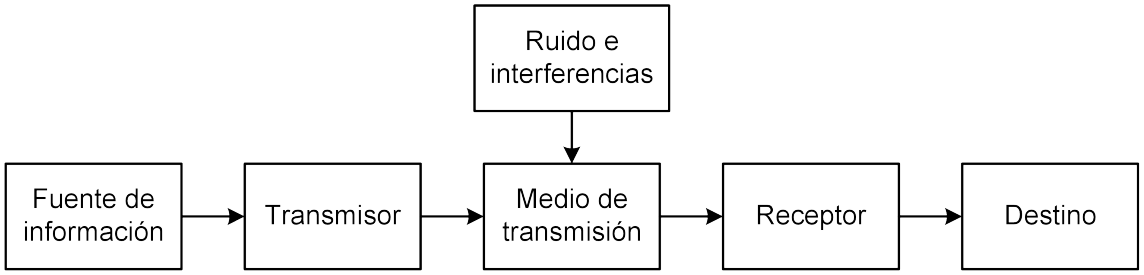


Figura N° 2: Diagrama general de un sistema de comunicaciones electrónico.

El transmisor es un conjunto de circuitos electrónicos diseñado para convertir la señal eléctrica en una señal apropiada para la transmisión, según el medio a usar. Están compuestos por osciladores, amplificadores, moduladores, mezcladores, sintetizadores de frecuencia, filtros y otros tipos de circuitos.

El canal de comunicación es el medio por el que las señales son enviadas de un lugar a otro. Este puede ser, por ejemplo, un par de conductores eléctricos como en un cable coaxial o en un cable trenzado. Otro ejemplo son medios ópticos como los cables de fibra óptica que se usan para transmitir información en forma de ondas de luz. El espacio libre para transmitir señales de manera inalámbrica.

Los receptores son una serie de circuitos que aceptan la información desde el canal y la convierten nuevamente a un mensaje úil que puede ser entendido. La información recibida es mostrada, por ejemplo, a través de un parlante si se trata de una señal de audio o por medio de una pantalla si se recibió una señal de video.

La mayoría de comunicaciones electrónicas son bidireccionales, por lo que ambos partes deben tener tanto un transmisor como un receptor. Estos equipos que incorporan circuitos que permiten transmitir y recibir información son conocidos como transceptores [4].

1.2.1.1 Tipos de comunicaciones electrónicas

Las comunicaciones electrónicas se pueden clasificar, según la capacidad de las partes de transmitir y/o recibir, en simplex o dúplex y, según la naturaleza de la señal a transmitir, en analógicos o digitales.

La forma más sencilla de llevar a cabo una comunicación electrónica es haciéndola de forma unidireccional, lo que normalmente se conoce como comunicación simplex. Es decir, una de las partes solo recibe información y no puede transmitirla, un ejemplo de estos sistemas son la difusión de radio o televisión.

Por otra parte, en los sistemas dúplex ambas partes pueden transmitir y recibir información. Dentro de los sistemas dúplex se distinguen los *half duplex*, en los que solo una de las partes puede transmitir en un momento determinado, y los *full duplex*, en los que ambas partes pueden transmitir información (y por lo tanto recibirla) a la vez [4]. Un ejemplo del primer tipo de sistemas es una radio portátil (o *handie*), mientras que para el segundo tipo de sistemas se puede nombrar una comunicación telefónica. Un diagrama simplificado de los sistemas simplex y dúplex se muestra en la Fig. 3.

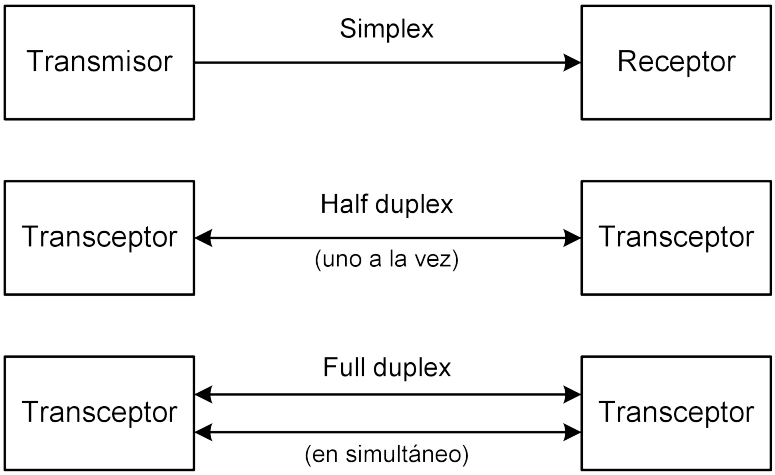


Figura N° 3: Diagrama general de los tipos de comunicaciones electrónicas.

Clasificando las comunicaciones según la naturaleza de la señal que contiene la información, estas se pueden en dividir en analógicas y digitales. En el primer caso, las señales son continuas y pueden tomar cualquier valor dentro de los rangos en los que trabaje el circuito, mientras que en las digitales las señales varían en pasos o incrementos discretos [4].

1.2.1.2 Modulación

Antes de que la información se pueda transmitir, esta debe ser convertida a una señal electrónica compatible con el medio en el que se va a transmitir. Sin importar si la información original se encuentra en una señal analógica o digital, todas se denominan señales de banda

base. En muchos casos, estas señales son incompatibles con el medio de transmisión o resulta poco práctico realizar una comunicación de forma directa con ellas.

A raíz de este problema, la información en banda base se usa para modular una señal de alta frecuencia llamada portadora. Las portadoras de una frecuencia mayor se irradian en el espacio de forma más eficiente que las señales en banda base, estas ondas electromagnéticas, que pueden viajar grandes distancias por el espacio, son conocidas como ondas de radiofrecuencia (RF) o simplemente ondas de radio [4].

La modulación es el proceso en el que una señal en banda base modifica a la portadora de alta frecuencia. Como se ilustra en la Fig. 4, la portadora es comúnmente una onda sinusoidal generadora por un oscilador. Esta portadora es enviada a un circuito llamado modulador junto con la señal en banda base para modificarla de una manera en específico. La portadora modulada es amplificada y se envía a la antena para transmitirla, este proceso es conocido como transmisión en banda ancha.

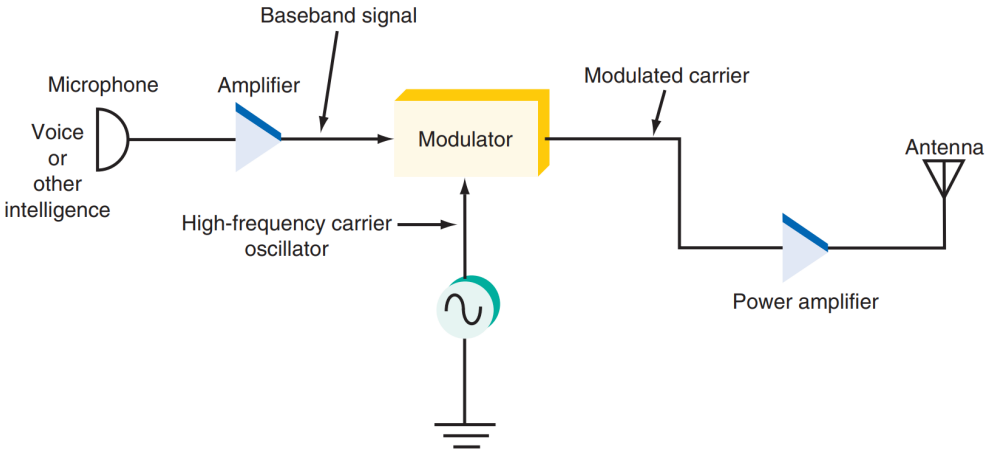


Figura N° 4: Diagrama de la modulación de una portadora. Fuente: [4]

Las tres formas en que la señal en banda base puede modificar a la portadora son variar su amplitud, variar su frecuencia o variar su fase. Los métodos más comunes de modulación son la modulación en amplitud (*amplitude modulation*, AM) y la modulación en frecuencia (*frequency modulation*, FM). La tecnología usada en este trabajo está basada en la FM, por lo que posteriormente se abordará este tema con más detalle.

1.2.1.3 Espectro electromagnético

El espectro electromagnético es el rango que comprende todas las frecuencias a las que pueden oscilar las señales electromagnéticas. Todas las señales eléctricas y electrónicas que se irradian en el espacio libre se consideran parte del espectro electromagnético, a diferencia de las señales que se transmiten en medios guiados y no se incluyen en el espectro [4]. La Fig. 5 muestra el espectro electromagnético y la clasificación de las ondas en función de su frecuencia.

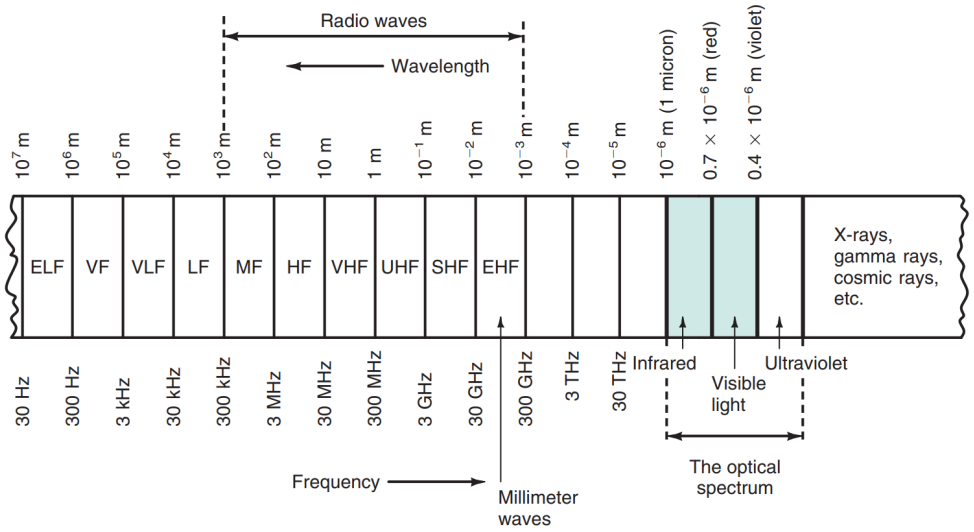


Figura N° 5: Representación del espectro electromagnético. Fuente: [4]

1.2.2 Ancho de banda y capacidad de información

Las dos limitaciones más significativas en el desempeño de un sistema de comunicaciones son el ruido y el ancho de banda. El ancho de banda de una señal es simplemente la diferencia entre la frecuencia más alta y la más baja de la información representada, mientras que el ancho de banda de un canal de comunicaciones es la diferencia entre la frecuencia más alta y la más baja que el canal deja pasar (es decir, su banda de paso). En definitiva, el ancho de banda del canal a usar debe ser mayor o igual que el ancho de banda de la señal de información para que todas las frecuencias significativas no se vean afectadas.

La capacidad de información es una medida de cuanta información puede ser propagada por un sistema de comunicaciones, representa el número de símbolos independientes que se pueden transportar en un sistema en una unidad de tiempo dada. El símbolo digital más básico usado para representar información es el dígito binario o bit. Por lo tanto, resulta conveniente expresar la capacidad de información de un sistema como una tasa de bits en bits por segundo (bit/s) [1].

La ley de Hartley se muestra en la Ec. 1, esta ecuación establece una relación entre el ancho de banda, el tiempo de transmisión y la capacidad de información.

$$I \propto B\, t \tag{1}$$

Donde I es la capacidad de información en bits por segundo, B es el ancho de banda en hertz y t es el tiempo de transmisión en segundos. A partir de la ecuación anterior se puede ver que la capacidad de información es una función lineal del ancho de banda y del tiempo de transmisión y es directamente proporcional a ambos.

Posteriormente, mediante el teorema de Shannon-Hartley, se estableció el límite teórico de la capacidad de información que puede alcanzarse sin errores para un ancho de banda específico y en presencia de ruido blanco aditivo gaussiano (*additive white gaussian noise*, AWGN) [1]. Este teorema se muestra en la Ec. 2.

$$I = B\ log_2\left(1 + \frac{S}{N}\right) \tag{2}$$

Donde I es la capacidad de información en bits por segundo, B es el ancho de banda del canal en hertz y S/N es la relación señal/ruido (*signal-to-noise-ratio*, SNR) sin unidades. De esta expresión se observa que para conseguir aumentar el límite de información teórico es necesario tener un mayor ancho de banda en el canal o bien incrementar la potencia de la señal o disminuir el ruido.

1.2.3 Análisis de ruido

El ruido eléctrico se define como cualquier energía eléctrica indeseada que cae dentro de la banda de paso de una señal. Se puede dividir al ruido en dos categorías generales: correlacionado y no correlacionado. La correlación implica una relación entre la señal y el ruido, por lo tanto, el ruido correlacionado existe solo cuando la señal está presente en el sistema. El ruido correlacionado, por otra parte, está siempre presente en el sistema, sin importar si hay una señal o no.

1.2.3.1 Ruido no correlacionado

El ruido no correlacionado se puede clasificar, a su vez, en ruido externo e interno. El ruido externo es aquel generado fuera del dispositivo o circuito, mientras que el interno es la interferencia eléctrica generada dentro del circuito [1].

Se destacan tres principales fuentes de ruido externo.

- Ruido atmosférico. Conocido también como electricidad estática, se origina en los fenómenos eléctricos de la naturaleza como lo son los rayos.
- Ruido extraterrestre. Consiste en señales que se originan fuera de la atmósfera de la Tierra, se divide en solar y cósmico. El ruido solar se compone de una radiación relativamente constante y perturbaciones esporádicas de alta intensidad. El ruido cósmico proviene de fuentes muy lejanas al sistema solar y su intensidad es relativamente baja.
- Ruido de origen humano. Incluye al ruido causado por circuitos en los que se producen conmutaciones, motores eléctricos, circuitos de ignición, luces fluo-

rescentes, equipos de generación de energía. Comprende un rango amplio de frecuencias que se propagan por el espacio y es más intenso en áreas industriales o densamente pobladas.

Existen varias fuentes ruido interno, sin embargo, se destacan tres de ellas:

- Ruido de disparo. Este tipo de ruido tiene un espectro de potencia con igual energía para cada frecuencia del ancho de banda del sistema. Se debe a las variaciones aleatorias en el flujo de corriente en dispositivos como tubos de vacío, diodos y transistores.
- Ruido de tiempo de tránsito. Muchos dispositivos con junturas producen ruido en frecuencias cercanas a su frecuencia de corte, este ruido de alta frecuencia se produce cuando el tiempo que tardan los portadores de carga en cruzar la juntura es comparable al periodo de la señal. Algunos de los portadores pueden regresar a través de la juntura, produciendo una corriente fluctuante que constituye ruido.
- Ruido térmico. Es producido por el movimiento aleatorio de los electrones en un conductor debido al calor. La densidad de potencia es igual para todas las frecuencias del ancho de banda, por lo que también es llamado ruido blanco. Existe en todos los dispositivos que están a temperaturas por encima del cero absoluto, la única manera de reducirlo es disminuir la temperatura, el ancho de banda del circuito o la cantidad de dispositivos en él.

1.2.3.2 Ruido correlacionado

El ruido correlacionado es una forma de ruido interno que está relacionado con la señal. Es producido por las no linealidades en la amplificación, una característica propia de todos los circuitos. Estas distorsiones no lineales se clasifican en armónica y de intermodulación.

La distorsión armónica ocurre cuando armónicas indeseadas de la señal original se producen en una amplificación o en la mezcla no lineal de señales. La distorsión de intermodulación hace referencia a la aparición de componentes de frecuencia indeseadas, que resultan ser la suma o diferencia de las señales originales involucradas en una mezcla.

1.2.4 LoRa

LoRa es un esquema de modulación propietario de la empresa Semtech (adquirido de Cycleo), se deriva del esquema de ensanchamiento por barrido de frecuencia (*chirp spread spectrum*, CSS). No debe confundirse con LoRaWAN, LoRa es una implementación de la capa

física independiente de las implementaciones de capas superiores. Esta característica permite que coexista y opere con arquitecturas de red existentes. Por otra parte, LoRaWAN es un estándar que se suele representar en la capa de red del modelo de interconexión de sistemas abiertos (*open systems interconnection*, OSI) y define arquitecturas de dispositivos finales, *gateways*, servidores, etc. para transmitir datos a un servidor de forma segura [7]. Esta distinción se puede ver en la representación del modelo OSI de la Fig. 6.

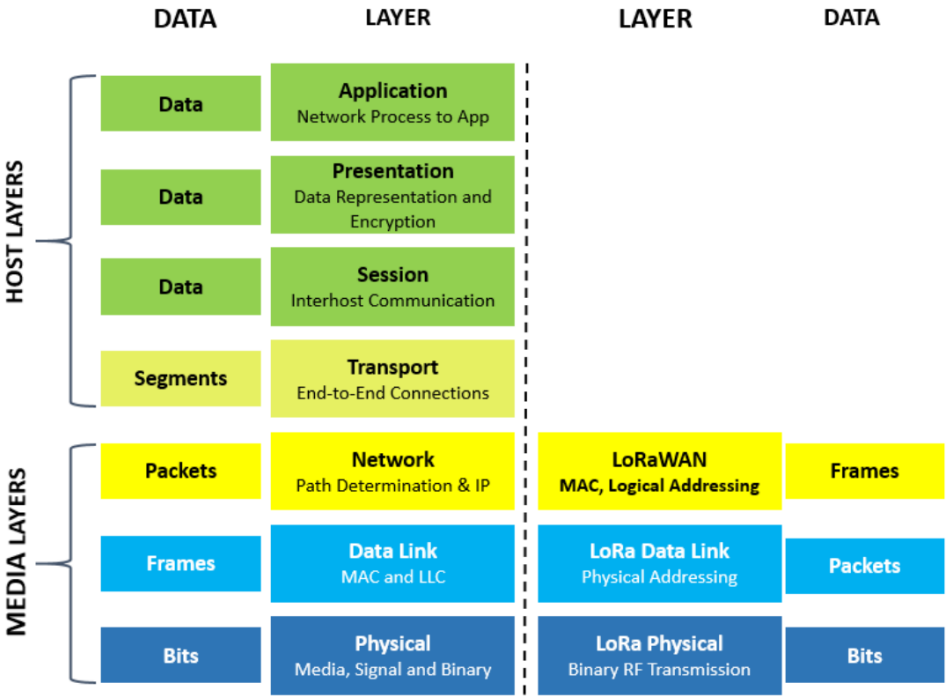


Figura N° 6: Clasificación de LoRa y LoRaWAN en el modelo OSI. Fuente: [7]

1.2.4.1 Internet de las cosas

El término IoT es reciente pero se refiere a uno más antiguo llamado máquina a máquina (*machine-to-machine*, M2M). M2M es un conjunto de tecnologías cableadas o inalámbricas que permiten el intercambio de información automático entre sistemas sin intervención humana. El IoT es simplemente una visión más amplia de M2M, donde los dispositivos no se encuentran solo en la industria, sino también en entornos urbanos, rurales y domésticos [6].

A comparación de otros sistemas electrónicos, en la mayoría de los casos los sistemas embebidos usados en IoT se caracterizan por ser de bajo consumo, poseer una potencia de cómputo reducida y un tamaño y costo menor.

Los protocolos usados en IoT para comunicar dispositivos de forma inalámbrica varían en rango, ancho de banda y potencia requerida. En la Fig. 7 se observa una comparación entre el rango y el ancho de banda de los protocolos de comunicación inalámbricos más usados en IoT. Abajo a la derecha se encuentran los protocolos de bajo ancho de banda y largo alcance, a su vez, resultan ser protocolos de potencia extremadamente baja. Las redes que utilizan

estos protocolos son conocidas como redes de área amplia de baja potencia (*low power wide area networks*, LPWAN), dentro de esta clasificación existen redes que usan protocolos como LoRaWAN, Sigfox, NB-IoT y LTE-M.

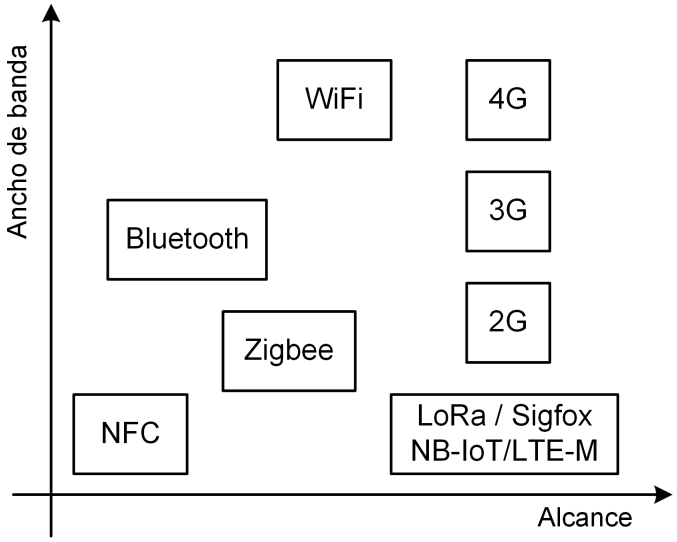


Figura N° 7: Comparación del rango y ancho de banda de protocolos usados en IoT.

1.2.4.2 Modulación en frecuencia

Una portadora sinusoidal puede ser modulada cambiando su amplitud, frecuencia o fase. La expresión básica para una portadora sinusoidal se muestra en la Ec. 3.

$$v = V_c \, sin(2\pi f t \pm \theta) \tag{3}$$

Donde V_c es el valor pico, f es la frecuencia y θ es la fase de la señal. Como su nombre lo indica, en FM la frecuencia de la portadora es modificada por la amplitud de la señal moduladora. A medida que la moduladora varía, la frecuencia de la portadora se desplaza por encima y por debajo de su frecuencia central (frecuencia sin modulación). El cambio producido en la frecuencia de la portadora es conocido como desviación de frecuencia f_d.

Un ejemplo de modulación en frecuencia se observa en la Fig. 8, la portadora sinusoidal se muestra como una señal triangular para facilitar la ilustración. La frecuencia de la portadora se mantiene constante mientras no está presente la señal moduladora. Se puede ver que en los valores pico de la moduladora se producen las desviaciones de frecuencia máximas.

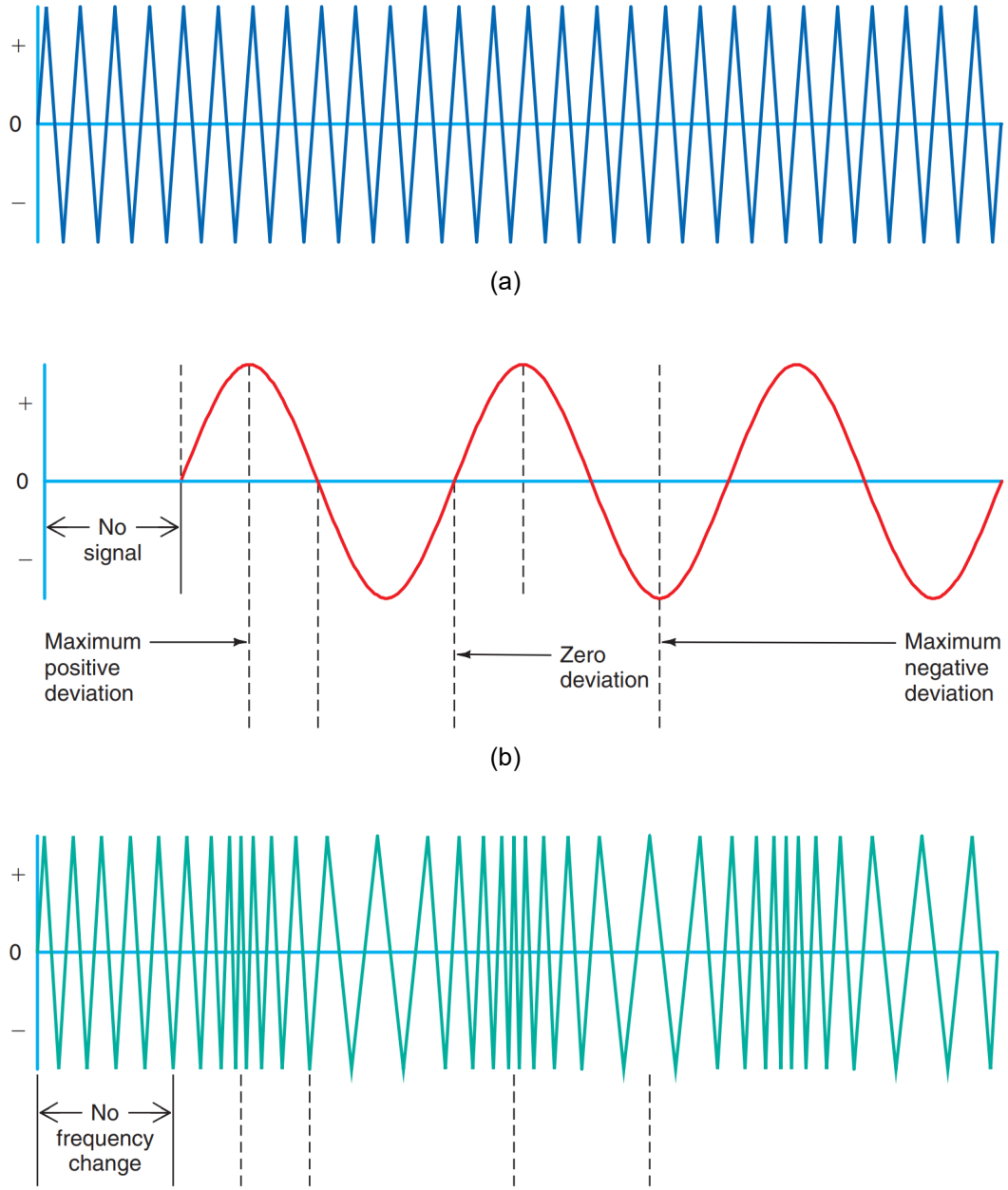


Figura N° 8: Modulación en frecuencia de una señal. (a) Portadora. (b) Moduladora. (c) Señal FM. Fuente: [4]

Otro tipo de modulación muy usada es la modulación en fase (*phase modulation*, PM). Debido a sus similitudes, FM y PM se consideran modulaciones angulares. La señal producida por un sistema en PM resulta ser también una señal FM, pero los circuitos moduladores y demoduladores difieren. Una diferencia clave entre los dos es que en PM la portadora tendrá una desviación de frecuencia solo cuando la señal moduladora cambie de amplitud, a diferencia de lo que sucede en FM donde la desviación de frecuencia de la portadora depende de la amplitud de la moduladora y no de su tasa de cambio [4].

En la actualidad es más frecuente el uso de una señal serial binaria como moduladora, de este modo la señal moduladora resultante solo tiene un valor de desviación de frecuencia.

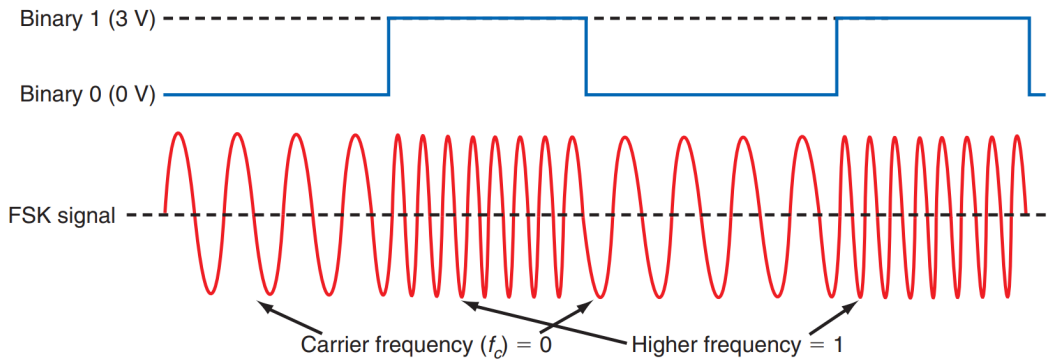


Figura N° 9: Modulación por desplazamiento de frecuencia (FSK). Fuente: [4]

Este tipo modulación es llamado modulación por desplazamiento de frecuencia (*frequency-shift keying*, FSK), un ejemplo se muestra en la Fig. 9.

Algunas de las ventajas de FM respecto a AM son una mayor inmunidad al ruido, resistencia a las interferencias y menor consumo de potencia debido al uso de amplificadores más eficientes. Como contraparte, FM requiere mucho más ancho de banda que AM y los circuitos usados son más complejos [4].

1.2.4.3 Espectro ensanchado

La mayoría de los métodos de modulación están diseñados para transmitir la máxima cantidad de información en el menor ancho de banda posible. Sin embargo, en las modulaciones de espectro ensanchado (*spread spectrum*, SS), la potencia de una señal es deliberadamente distribuida en un ancho de banda mucho mayor al necesario. Una comparación entre una señal en banda estrecha y su versión en SS se muestra en la Fig. 10

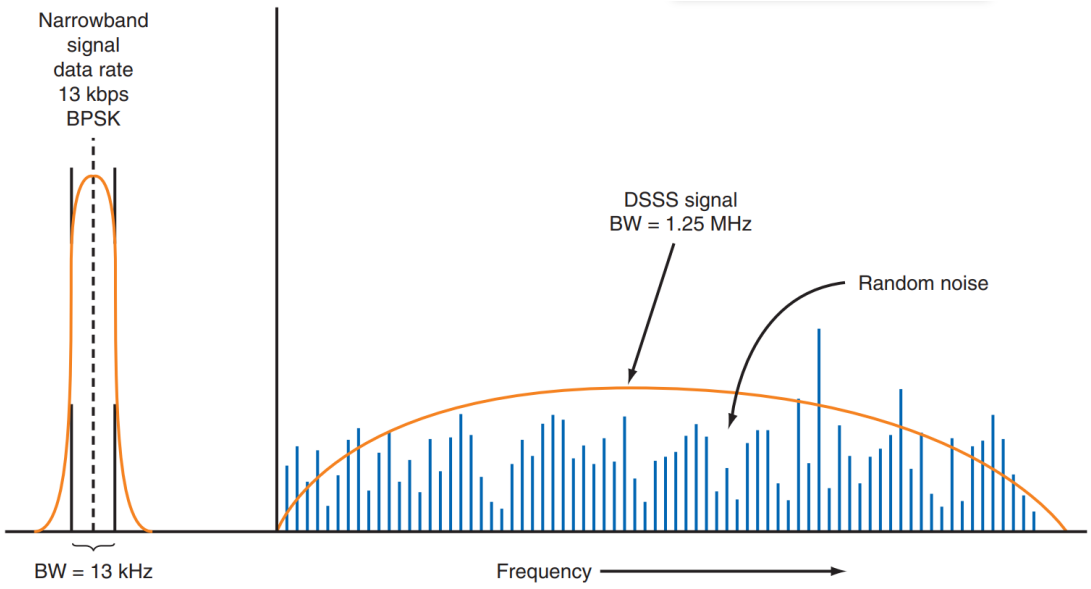


Figura N° 10: Señal en banda estrecha y ensanchamiento de espectro. Fuente: [4]

El objetivo de ensanchar el espectro es proveer ganancia de procesamiento a la señal, esta ganancia está definida por la Ec. 4. Donde BW es el ancho de banda del canal y f_b es la tasa de datos.

$$G = \frac{BW}{f_b} \tag{4}$$

Al usar técnicas de ensanchamiento de espectro, la relación señal ruido presenta valores muy bajos ya que generalmente la señal se encuentra por debajo del piso de ruido. Observando la Ec. 2 y suponiendo que $SNR << 1$, esta se puede reescribir de la manera que se observa en la Ec. 5.

$$C \approx BW \cdot \frac{S}{N} \tag{5}$$

En donde se puede ver que para una SNR fija, para poder transmitir la información sin errores es necesario aumentar el ancho de banda o reducir la cantidad de información.

Los beneficios de implementar esta técnica se listan a continuación.

- Mayor seguridad. Previene recepciones sin autorización, ya que es necesario que el receptor tenga un gran ancho de banda y funcione con exactamente el mismo esquema de modulación.
- Resistencia al *jamming* e interferencias. El jamming (interferencia intencional) está generalmente restringido a una única frecuencia, lo cual no afecta a una señal en SS. De la misma manera, la interferencia de las señales que ocupan la misma banda son prácticamente eliminadas.
- Banda compartida. Varios usuarios pueden compartir la misma banda con poca interferencia.
- Resistencia a los efectos de la propagación por trayectos múltiples. Las técnicas de SS eliminan las variaciones que causa la propagación (reflexión, refracción, difracción) en la intensidad de la señal recibida.

1.2.4.4 Ensanchamiento de espectro por barrido de frecuencia

El ensanchamiento de espectro por barrido de frecuencia o CSS es la base de la modulación LoRa, por lo que también se le suele llamar a esta última modulación CSS. Un *chirp* es una señal sinusoidal cuya frecuencia varía linealmente con el tiempo, su nombre proviene de las siglas en inglés de *compressed high intensity radar pulse* ya que se comenzó a usar en sistemas de radar [8]. Un ejemplo de un chirp ascendente y descendente en el dominio del tiempo se puede observar en la Fig. 10, mientras que en la Fig. 11 se presenta el espectrograma (gráfica de la frecuencia en función del tiempo) de otros chirps.

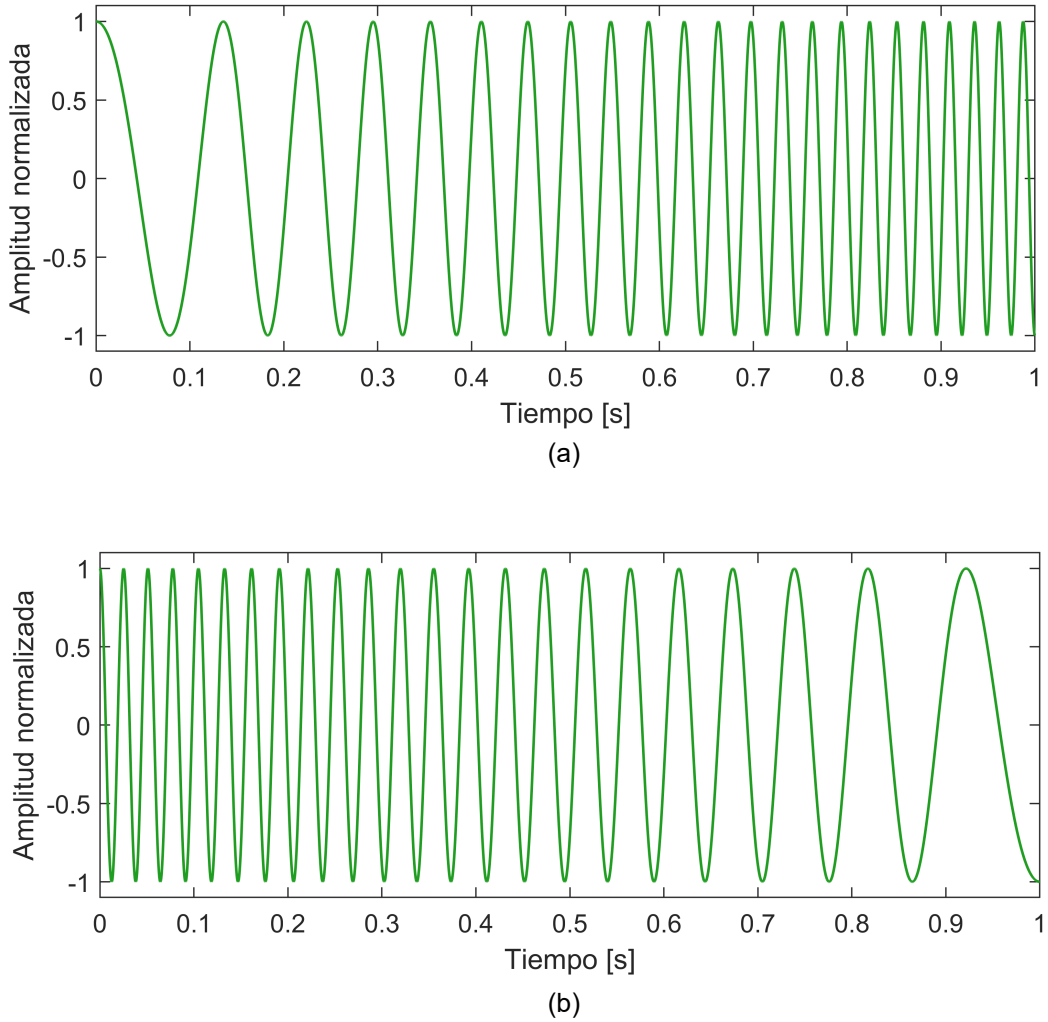
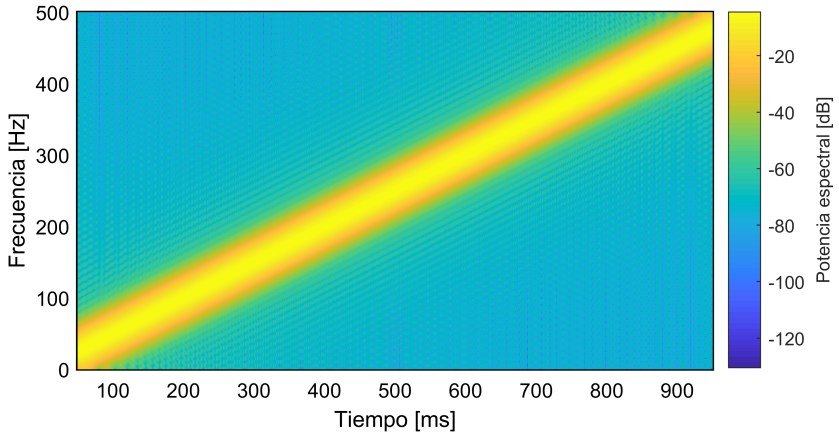


Figura N° 10: Chirps en el dominio del tiempo. (a) Chirp ascendente. (b) Chirp descendente.



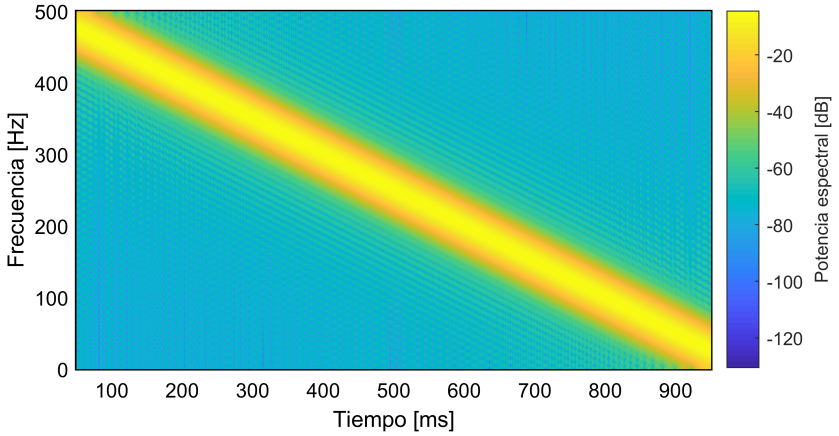


Figura N° 11: Espectrogramas de chirps. (a) Chirp ascendente. (b) Chirp descendente.

1.2.4.5 Modulación LoRa

La modulación LoRa está compuesta por un conjunto de M chirps ascendentes ortogonales, esto quiere decir que las señales son matemáticamente independientes y no interfieren entre si. Cada chirp, que contiene la información de un símbolo, es modulado por una secuencia de datos binarios de longitud igual al factor de ensanchamiento (*spreading factor*, SF), por lo que $M = 2^{SF}$ es el total de chirps o símbolos distintos que se pueden usar. Este proceso se ilustra en la Fig. 12.

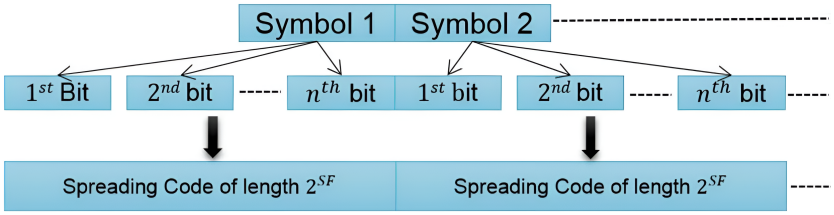


Figura N° 12: Ensanchamiento de espectro en la modulación LoRa. Fuente: [10]

Un símbolo $s(nT_s)$ es un número real que se obtiene usando un vector de datos $\boldsymbol{w}(nT_s)$ con una longitud de SF dígitos binarios, la duración del símbolo o chirp está denotada por T_s. La expresión general de estos símbolos se muestra en la Ec. 6 [9].

$$s(nT_s) = \sum_{h=0}^{SF-1} \boldsymbol{w}(nT_s)_h \cdot 2^h \tag{6}$$

Se puede ver que en la suma se pondera cada bit de $\boldsymbol{w}$ por 2^h, entonces, $s(nT_s)$ tomará el conjunto de valores $\{0, 1, 2, ..., 2^{SF} - 1\}$, como se había mencionado anteriormente. La forma de onda del chirp, de duración T_s, para un cierto $s(nT_s)$ es descrita por la Ec. 7 en tiempo discreto [9].

$$c[nT_s + kT] = \frac{1}{\sqrt{2^{SF}}}\, exp\left[j2\pi \left[(s(nT_s) + k)\ mod\left(2^{SF}\right)\right] kT\frac{BW}{2^{SF}}\right] \tag{7}$$

La amplitud de esta exponencial compleja esta afectada por un factor normalizador, de esta forma la potencia de la señal es independiente de SF. La fase instantánea de una exponencial compleja $e^{j\theta}$ es simplemente su argumento, esta se muestra en la Ec. 8.

$$\phi[k] = 2\pi \left[(s(nT_s) + k)\ mod\left(2^{SF}\right)\right] kT\frac{BW}{2^{SF}} \tag{8}$$

Debido a que en tiempo discreto no es posible hacer una derivada, la frecuencia instantánea de la señal se obtiene a través de una diferencia finita, como se observa en la Ec. 10.

$$f[k] = \frac{1}{2\pi}\left(\phi[k+1] - \phi[k]\right) \tag{9}$$

$$f[k] = \left[(s(nT_s) + 2k + 1)\ mod(2^{SF})\right] T\frac{BW}{2^{SF}} \tag{10}$$

Como se puede ver, la frecuencia del chirp aumenta linealmente con k, el índice de tiempo. Además, el símbolo a transmitir s introduce un desplazamiento en frecuencia, lo cual diferencia a los 2^{SF} chirps. Mediante la operación modulo, una vez que se llegó a la frecuencia máxima, esta comienza de nuevo hasta barrer por completo el ancho de banda.

1.2.4.6 Características de LoRa

Tasa de datos

Los símbolos de longitud SF son ensanchados a una longitud de 2^{SF} dígitos binarios, llamados chips para diferenciarlos de la información útil.

Suponiendo un ancho de banda en la transmisión BW, de forma que se transmite un chip cada $T = 1/BW$, un símbolo será enviado a la entrada del modulador en un tiempo $T_s = 2^{SF} \cdot T$. A partir de esto, se definen la tasa de chips y la tasa de símbolos en la Ec. 11 y la Ec. 12 respectivamente [5].

$$R_c = \frac{1}{T} = BW \quad \left[\frac{chips}{s}\right] \tag{11}$$

$$R_s = \frac{1}{T_s} = \frac{BW}{2^{SF}} \quad \left[\frac{simbolos}{s}\right] \tag{12}$$

Con base en estas expresiones, se dice que en LoRa un chip es enviado por segundo por cada Hz de ancho de banda [5]. Como cada símbolo transporta SF bits de información, la tasa de bits queda definida por la Ec. 13. Los anchos de banda estándar que se usan en LoRa son 125 kHz, 250 kHz y 500 kHz, como se ilustra en la Fig. 13 su espectro bilateral está centrado alrededor de una frecuencia central.

$$R_b = R_s \cdot SF = \frac{BW \cdot SF}{2^{SF}} \quad \left[\frac{bits}{s}\right] \tag{13}$$

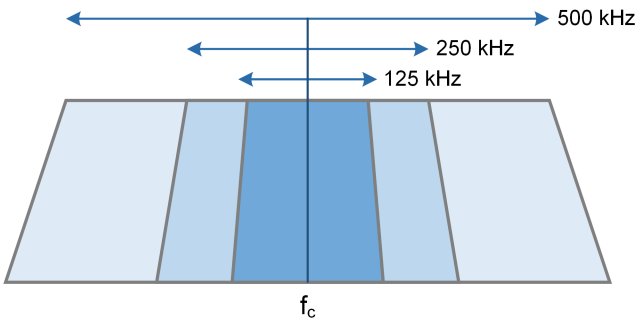


Figura N° 13: Anchos de banda estándar para la modulación LoRa.

Estos cálculos permiten obtener la tasa de datos útiles sin tener en cuenta preámbulos, encabezados, códigos de detección y corrección de errores, etc.

La relación entre el ancho de banda, el factor de ensanchamiento y la tasa de bits obtenida se muestra en la Fig. 14.

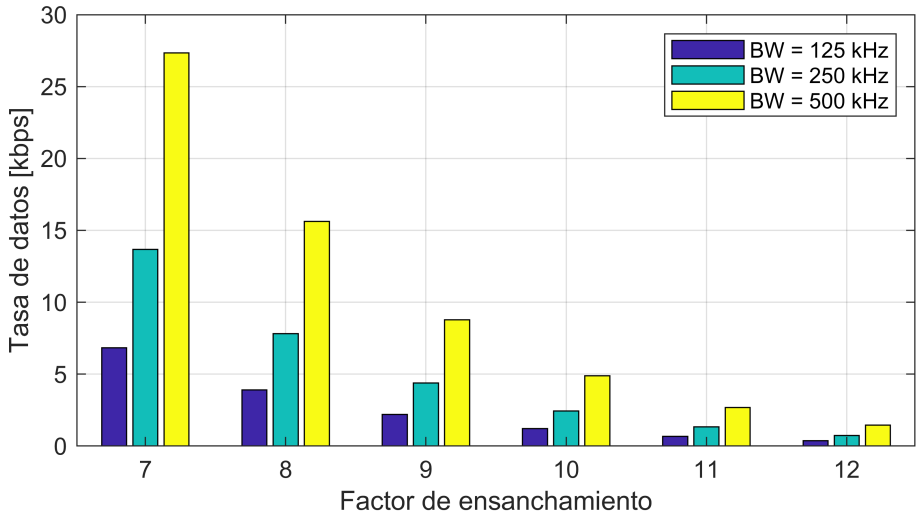


Figura N° 14: Tasas de bits en LoRa para distintos anchos de banda y factores de ensanchamiento.

Se observa que un mayor factor de ensanchamiento disminuye la tasa de bits, recordando la Ec. 5, esto tiene como consecuencia la posibilidad de realizar una comunicación exitosa a mayor distancia. De la misma manera, al aumentar el ancho de banda es posible transmitir más información.

Sensibilidad del receptor

El piso de ruido del receptor es una medida de la mínima potencia que pueden presentar las distintas fuentes de ruido presentes en el sistema, se suele expresar para un ancho de banda de 1Hz a partir de la Ec. 14 y es posible escalar el resultado según el ancho de banda usado.

$$N_0 = 10\,log(kT \cdot BW_{1\,Hz}) + 30 \quad \left[\frac{dBm}{Hz}\right] \tag{14}$$

Donde k es la constante de Boltzmann, T es la temperatura ambiente y BW el ancho de banda. Es común calcularlo considerando una temperatura ambiente de 290 K, lo que da como resultado -174 dBm [4].

La figura de ruido es la relación entre la SNR de la señal recibida y la SNR de la señal obtenida a la salida del receptor, como se puede ver en la Ec. 15. Es una medida del ruido introducido por el amplificador y el resto de circuitos [11].

$$NF = 10\,log\left(\frac{SNR_{in}}{SNR_{out}}\right) \tag{15}$$

La sensibilidad del receptor es el nivel mínimo de potencia de la señal recibida que

el receptor puede detectar de manera confiable. Para este cálculo se tienen en cuenta el piso de ruido (escalado según el ancho de banda), la ganancia de procesamiento, calculada para LoRa en la Ec. 16, y la relación señal/ruido requerida para obtener una determinada tasa de errores, que es el parámetro en función del que se suele calcular. La expresión para obtener la sensibilidad un receptor se muestra en la Ec. 17

$$G_p = 10\,log\left(\frac{BW}{R_b}\right) = 10\,log\left(\frac{2^{SF}}{SF}\right) \tag{16}$$

$$S_{min} = -174\,dBm + 10\,log(BW) + NF + SNR - G_p \quad [dBm] \tag{17}$$

Se puede ver que a mayor ancho de banda se integra más ruido y la sensibilidad empeora, ocurriendo lo mismo con la figura de ruido en el receptor. Como se vio anteriormente, la ganancia de procesamiento es útil para poder recibir adecuadamente señales de menor potencia.

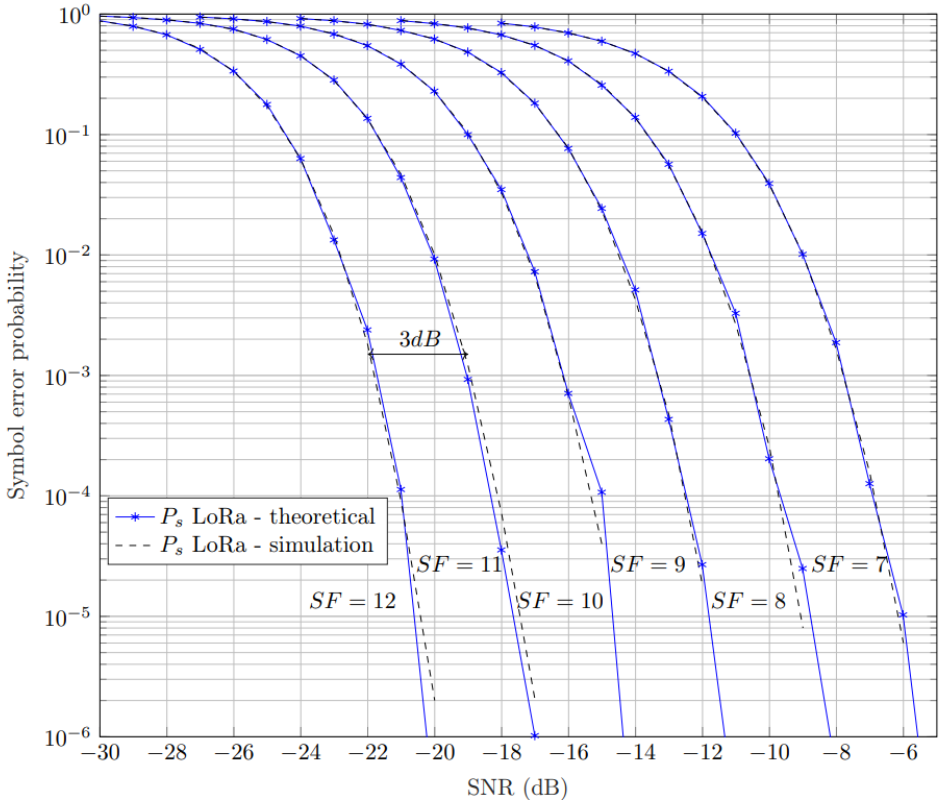


Figura N° 15: Tasa de error de símbolos en función de la SNR para distintos SF. Fuente: [12]

En la Fig. 15 se observa una gráfica de la tasa de error de símbolos (*symbol error rate*, SER) en función de la SNR en una transmisión con LoRa. La SER para una misma SNR disminuye con SF y se puede ver que es posible conseguir valores de SER óptimos (alrededor de $10^{-}6$) incluso para señales cuya potencia está muy por debajo del nivel de ruido.

Preámbulos y sincronización

Cada mensaje en LoRa comienza con un preámbulo que consiste en una serie de 8 símbolos idénticos, los cuales servirán para indicar el comienzo de la transmisión. A continuación, el mensaje contiene 2 chirps descendentes que se usan para sincronizar el inicio de los símbolos y permiten conocer donde comienzan los símbolos de información útil. Una representación del espectrograma de un mensaje en LoRa con su preámbulo, sincronización y chirps de información se muestra en la Fig. 16. Las líneas verticales delimitan cada símbolo para facilitar la visualización.

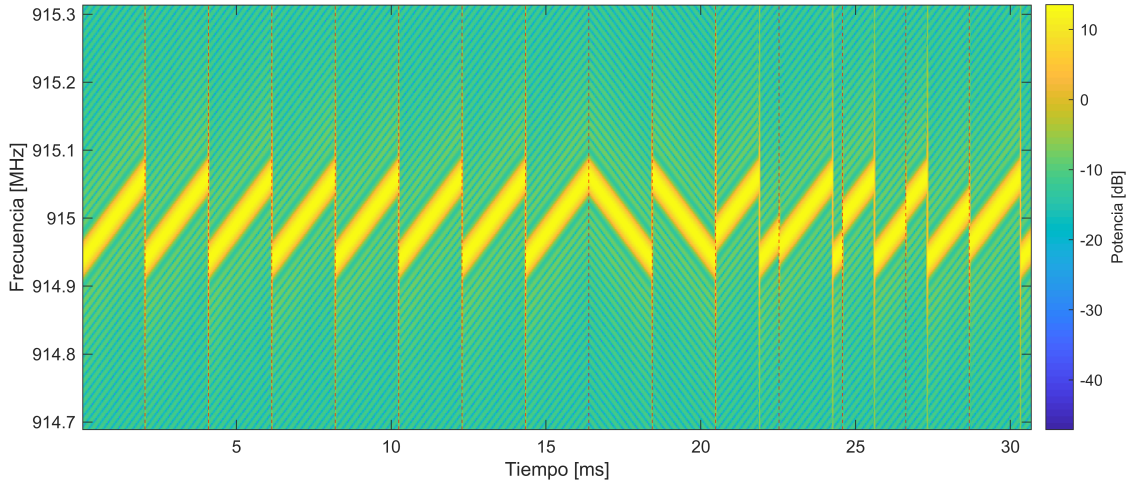


Figura N° 16: Espectrograma de un mensaje en LoRa con preámbulo, símbolos de sincronización y 5 símbolos de información.

Ortogonalidad

Los paquetes de LoRa que tengan diferente factor de ensanchamiento serán ortogonales, de forma que si dos paquetes llegan al receptor en el mismo canal y al mismo tiempo, pero con distinto SF, serán demodulados correctamente [7]. En el caso de que los factores de ensanchamiento sean iguales, habrá una colisión. Semtech asegura que uno de los paquetes sobrevivirá si su potencia es mayor en 6 dB al otro, sin embargo, se demostró que es posible recibir un paquete incluso con menores diferencias de potencia [12].

1.2.4.7 Legislaciones y normativas

LoRa hace uso de las bandas de frecuencia no licenciadas para aplicaciones industriales, científicas y médicas, conocidas como bandas ISM (*industrial, scientific and medical radio bands*), de modo que no se requiere ninguna autorización ni registro para realizar comunicaciones con esta tecnología. En Argentina se utiliza la banda de frecuencias AU915-928, definida por la LoRa Alliance en el documento RP002-1.0.2 [13], la cual va de 915 MHz hasta 928 MHz.

Esta banda coincide en Argentina con una de las bandas de frecuencia de uso compartido sin autorización, estas permiten la operación de dispositivos de radiocomunicaciones sin una autorización que asegure la asignación de una frecuencia o canal específico, pero estableciéndose ciertos requisitos de convivencia. El uso de estas bandas fue aprobado por el ENACOM (Ente Nacional de Comunicaciones) en la Resolución N° 4653/19 y establece los parámetros a cumplir para la banda de 915 a 928 MHz [14].

LoRaWAN fue reconocido como un estándar para las LPWAN en la Recomendación ITU-T Y.4480, “Protocolo de baja potencia para redes de área amplia inalámbricas”.

1.3 Descripción del sistema implementado

El sistema eléctrico actualmente instalado en el invernadero de la facultad cuenta con un aerogenerador que produce en su salida una tensión trifásica de amplitud y frecuencia variable. Esta tensión trifásica se rectifica y, mediante un regulador de carga, se gestiona la carga de un banco de baterías, el cual alimenta al inversor que provee electricidad al lugar. Un diagrama de la instalación del invernadero y del sistema de comunicaciones se muestra en la Fig. 17.

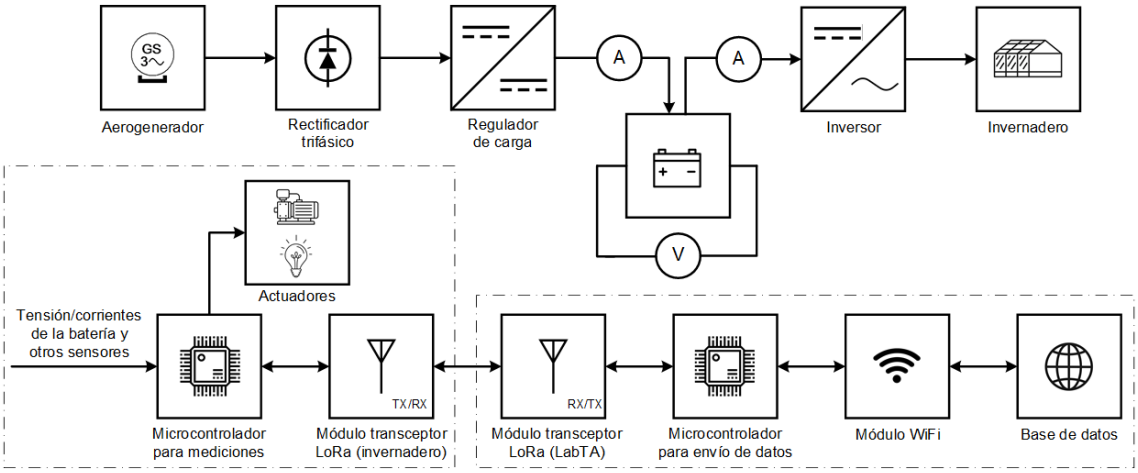


Figura N° 17: Diagrama del sistema de medición y comunicaciones.

Por medio del sistema de mediciones desarrollado, se mide la tensión del banco de baterías, la corriente de carga y la corriente que consume el inversor. Estos datos son adquiridos por un microcontrolador que, a su vez, gestiona la comunicación con el modulo transceptor que estará en otra placa de circuito impreso (PCB). Se pretende que el prototipo sea escalable para permitir la transmisión de información de otros sensores y/o microcontroladores, además de accionar distintos actuadores. Del otro lado, la información transmitida hasta el LabTA es recibida por un segundo transceptor. Finalmente, en este lugar se dispone de conectividad WiFi y los datos se suben a una base de datos, lo que permite visualizarlos desde cualquier lugar mediante una aplicación para teléfonos celulares.

CAPITULO 2: Sistema de medición

Con el fin de adquirir los valores de tensión y corrientes en el sistema, se utilizó el convertidor analógico-digital (*analog-to digital-converter*, ADC) interno de un microcontrolador STM32F103C8T6 integrado en una placa de desarrollo conocida popularmente como "*Bluepill*". Esta placa se muestra en la Fig. 18.

El microcontrolador mencionado cuenta con un procesador basado en un núcleo ARM Cortex-M3 de 32 bits, pudiendo llegar a operar con frecuencias de reloj de hasta 75 MHz. Posee una memoria principal de 20 KB de tipo SRAM (memoria de acceso aleatorio estática) y una memoria secundaria de 128 KB de tipo *flash*. Incorpora más de 30 pines de entrada/salida de uso general (*general purpose input/output*, GPIO), 4 temporizadores, 9 interfaces de comunicación y modos de bajo consumo [16].

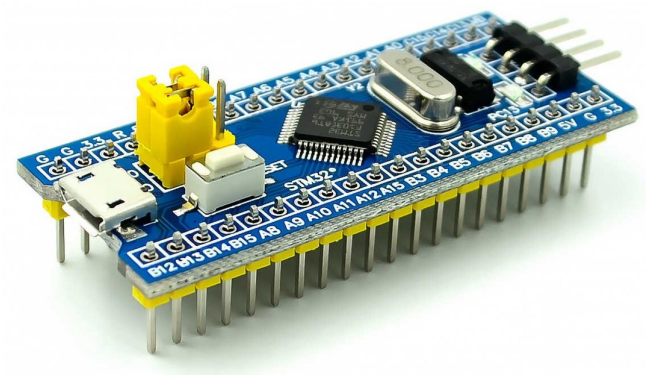


Figura N° 18: Placa de desarrollo Bluepill con STM32F103C8T6.

Por otra parte, el ADC cuenta con un registro de aproximaciones sucesivas (*successive approximation register*, SAR) de 12 bits y posee 16 canales que permiten medir hasta el valor de la tensión de alimentación a una velocidad máxima de 1 Msps. Es posible configurar este periférico en distintos modos y dispararlo desde distintas fuentes, además, soporta operaciones con acceso directo a memoria (*direct memory access*, DMA).

La alimentación para el sistema de medición está provista directamente por una de las baterías de 12 V, esta tensión es reducida hasta 5 V usando un regulador LM7805CV. El circuito usado para este fin se muestra en la Fig. 19. Esta tensión se usó también para la placa de transmisión en donde se encuentra el módulo transceptor usado para ese fin. Se prefirió el uso de reguladores lineales frente a los convertidores conmutados debido a las fluctuaciones en el nivel de tensión y el ruido electromagnético que estos últimos producen, lo cual puede generar problemas en la comunicación.

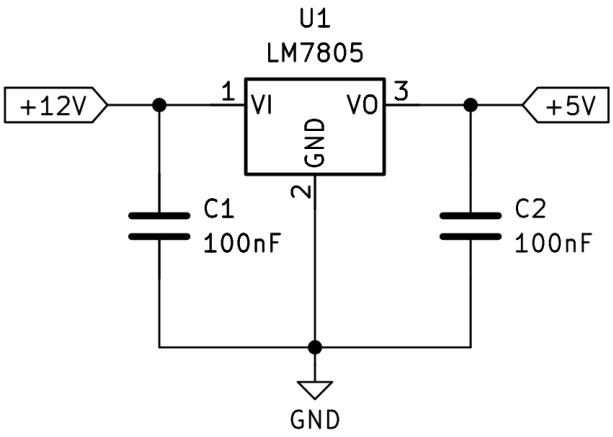


Figura N° 19: Circuito reductor de tensión usado para alimentar el sistema.

2.1 Medición de intensidad de corriente

Para realizar la medición corriente de carga de las baterías y la corriente proporcionada al inversor se utilizaron transformadores de corriente LAH 100-P, los cuales pueden medir intensidades de CC y CA de hasta 100 A con aislamiento galvánico. Estos dispositivos poseen una relación de transformación de 1:2000 [17]. Este dispositivo se muestra en la Fig. 20.

Figura N° 20: Transductor de corriente LAH 100-P. Fuente: [17].

La corriente del secundario atraviesa un resistor, produciendo una caída de tensión que se usa para medir de forma indirecta la intensidad de corriente en el primario. En este caso, teniendo en cuenta las características del sistema existente y los componentes a usar, se consideró que la corriente máxima a medir es de 45 A. Sabiendo que la tensión del banco de baterías es en promedio de 24 V esto equivale a una potencia de aproximadamente 1 kW.

El circuito utilizado se muestra en la Fig. 21, donde la corriente de carga de las baterias (o la que alimenta al inversor) es representada por una fuente de corriente. Con dos resistores de 100 Ω colocados en el secundario se consiguió una resistencia equivalente de 50 Ω; de esta manera, cuando la corriente en el primario sea de 45 A, en el secundario circularan 22,5 mA y la caída de tensión en los resistores será de 1,13 V.

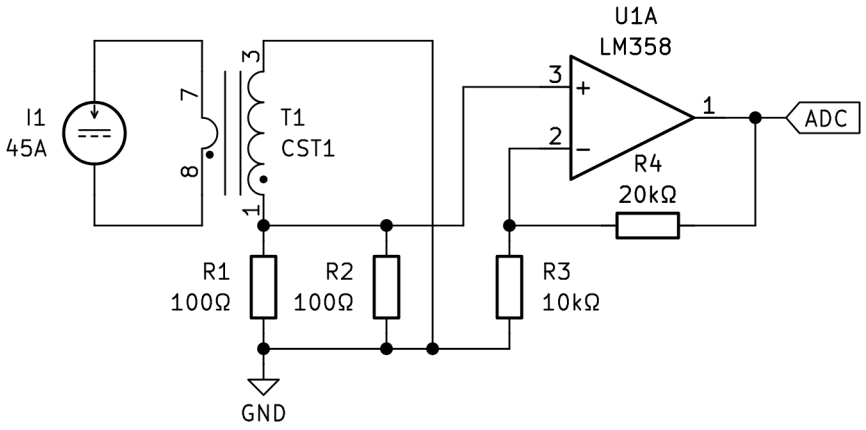


Figura N° 21: Circuito para la medición de corriente continua

La tensión en el divisor resistivo es amplificada por medio de un amplificador no inversor de ganancia 3 con el objetivo de aprovechar el rango de medición del ADC. Con este fin se usaron los dos amplificadores operacionales de un circuito integrado LM358P. La tensión de salida de este circuito está definida por la Ec.18.

$$V_o = V_{in} \cdot \left(1 + \frac{R_4}{R_3}\right) = 1,13\,V \cdot \left(1 + \frac{20\,k\Omega}{10\,k\Omega}\right) = 3,39\,V \tag{18}$$

A raíz de que los transformadores de corriente requieren una alimentación que se encuentre en un rango de entre $\pm$12 V y $\pm$15 V, aún cuando no se está midiendo CA, se tomó como masa del circuito la unión entre el borne positivo de la primera batería y el borne negativo de la segunda. De esta manera, respecto al punto de referencia, el borne positivo de la segunda batería tendrá +12 V y el borne negativo de la primera batería tendrá -12 V.

2.2 Medición de tensión

Como se explicó en el apartado anterior, la referencia de tensión es la conexión entre las baterías, por lo que será necesario medir una tensión negativa y otra positiva respecto a la masa del circuito. La tensión de la batería será tratada como 2 tensiones distintas que luego se sumarán para obtener la tensión total del banco de baterías.

Los circuitos utilizados se muestran en la Fig. 22 y consisten en un divisor resistivo seguido de un amplificador inversor con una ganancia de 1 para medir la tensión negativa y un seguidor de tensión o *buffer* para medir la tensión positiva. El amplificador inversor permite medir la tensión negativa, aportando una ganancia de -1, mientras que el buffer con ganancia de 1 proporciona una alta impedancia de entrada, reduciendo la corriente entregada por el divisor y proporcionando una medición más exacta. Además, asegura la aislación entre las dos partes del circuito y actúa como una protección frente a tensiones de entrada excesivas, pudiendo

obtener en su salida solo una tensión menor a la de la alimentación (5 V). Obviamente, las ventajas del búfer están presentes también en el amplificador inversor.

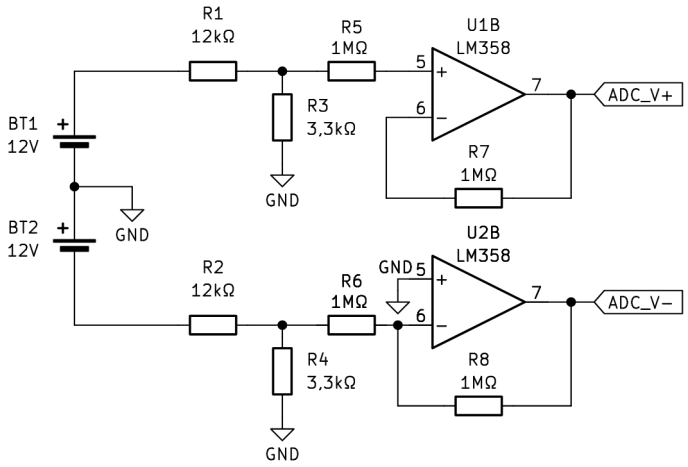


Figura N° 22: Circuito para la medición de tensiones positivas y negativas.

Al igual que en el circuito para la medición de corriente, se utilizó un LM358P. Los valores de resistencia para los divisores se obtuvieron considerando una tensión máxima de entrada de 15 V y, en función de los valores comerciales, se seleccionaron resistores de 12 kΩ y 3,3 kΩ. De esta manera, como se muestra en la Ec.19, se obtiene un valor de tensión máximo de 3,24 V en la entrada del ADC, que resulta cercano a los 3,3 V admisibles.

$$V_o = V_{in} \cdot \frac{R_3}{R_1 + R_3} = 15\,V \cdot \frac{3,3\,k\Omega}{(12 + 3,3)k\Omega} = 3,24\,V \tag{19}$$

Los resistores de 1 MΩ colocados en las entradas inversoras y no inversoras de los amplificadores operacionales tienen el propósito de igualar las corrientes de polarización en las entradas, ayudando a reducir la diferencia entre la tensión de entrada y salida. Además, debido a que el amplificador operacional que funciona como amplificador inversor está alimentado solo con un riel positivo (de 0 V a +5 V), las corrientes inversas producidas por aplicar una tensión negativa en la entrada inversora podrían destruir el dispositivo. El fabricante asegura que mientras esta corriente inversa no sea mayor a 1 mA el amplificador puede operar de forma segura [18], los resistores agregados permiten lograr esta condición.

2.3 Programa implementado para las mediciones de tensión y corriente

Un diagrama de flujo aproximado del programa implementado para realizar las mediciones se muestra en la Fig. 23. El microcontrolador se programó en lenguaje C usando el software STM32CubeIDE.Se comienza inicializando todos los periféricos necesarios, uno de ellos, el temporizador TIM1, está siempre activo y se inicializó con un periodo de 5 minutos.

Cuando se cumple este tiempo, el temporizador dispara una interrupción que invoca un *call-back* en el que se inicializa el temporizador TIM3. Este segundo temporizador es el encargado de generar el evento que dispara el ADC para comenzar con la conversión de la señal en 6 canales, correspondientes a las 2 tensiones y 2 corrientes mencionadas y, de forma adicional, se adquiere una tensión de referencia interna del microcontrolador para mejorar las mediciones y su temperatura interna.

Esta conversión de 6 canales se repite 100 veces con el objetivo de aplicar un filtro promediador sobre las mediciones de tensión y corriente. Dicho filtro es una de las formas más sencillas de atenuar el ruido de alta frecuencia que pudiera estar presente en la señal.

Cuando se obtienen las 100 muestras, se detiene el temporizador TIM3 y se realizan las conversiones necesarias para obtener los valores de tensión y corriente y la temperatura interna del microcontrolador. Finalmente, se preparan los datos para enviarlos al transceptor LoRa encargado de la transmisión inalámbrica.

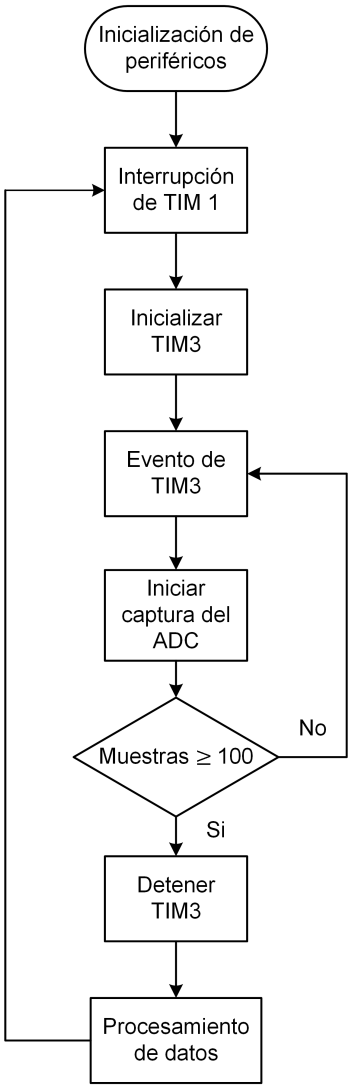
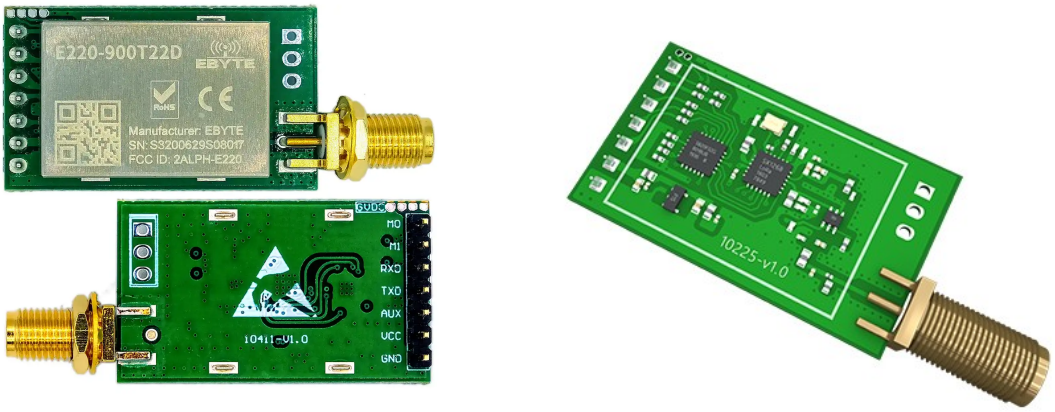


Figura N° 23: Diagrama de flujo del programa usado para las mediciones.

CAPITULO 3: Sistema de comunicaciones

3.1 Transceptor LoRa E220-900T22D

El transceptor LoRa E220-900T22D de Ebyte es un módulo LoRa inalámbrico basado en el circuito integrado LLCC68. Ofrece una interfaz UART (*Universal Asynchronous Receiver - Transmitter*) para facilitar la comunicación con un microcontrolador, opera en un rango de frecuencias desde 850 MHz hasta 930 MHz y puertos IO compatibles con 3,3 V y 5 V [19]. En la Fig. 24 se puede observar una imagen de este módulo con y sin su blindaje.



(a) (b)

Figura N° 24: Módulo transceptor LoRa E220-900T2DD. (a) Vista superior e inferior. Fuente: [19]. (b) Modelo en 3D del módulo sin blindaje. Fuente: [20].

Puede operar en modo *broadcast* transmitiendo a todos los receptores que esten en el mismo canal (misma frecuencia) o en modo de transmisión fija (punto a punto), en donde la transmisión se realiza solo a un receptor que esté en el mismo canal con una dirección especifica. Permite ademas configurar una clave con la que se desencriptaran los datos una vez recibidos para una mayor seguridad, incluye una función LBT (*Listen Before Talk*) para evitar transmitir en el canal cuando otro dispositivo ya lo está haciendo. Posee un modo de bajo consumo del que puede despertar al comenzar a recibir información. Incorpora un conector hembra para una antena SMA con una impedancia de 50 Ω.

Cuenta con 7 pines, de los cuales 2 para la alimentación (VCC y GND), 2 para la comunicación serial (TXD y RXD), 2 para establecer el modo de funcionamiento del módulo (M0 y M1) y un pin auxiliar (AUX). Este último pin está normalmente en estado alto (3,3 V) y cambia a estado bajo (0 V) cuando el módulo está ocupado, ya sea cambiando de modo de operación o recibiendo o transmitiendo datos.

3.1.1 Modos de funcionamiento

El módulo cuenta con 4 modos de operación, que son establecidos según la combinación de valores de tensión (alto o bajo) en los pines M0 y M1. En la Tabla 1 se listan estos modos de funcionamiento y sus características.

Tabla N° 1: Modos de operación del módulo LoRa E220-900T22D

Modo	M0	M1	Descripción	Transmisión	Recepción
0	0	0	Normal	Al transmitir datos hacia el pin RXD el módulo comenzará la transmisión inalámbrica.	El módulo puede recibir información con normalidad; una vez finalizada la recepción, los datos se transmiten a través del puerto serial por el pin TXD.
1	0	1	Transmisión WOR	Se añade un preámbulo al mensaje que sirve para despertar a un transceptor que está en modo de bajo consumo.	Puede recibir datos al igual que en el modo 0.
2	1	0	Recepción WOR	La transmisión inalámbrica está desactivada.	Solo puede recibir información de un transmisor en modo WOR.
3	1	1	*Deep Sleep*	La transmisión inalámbrica está desactivada.	La recepción inalámbrica está desactivada.

En el modo 2, recepción WOR (*Wake On Receive*), el módulo se encuentra en un estado de bajo consumo del que sale al recibir un mensaje de un módulo en modo 1; una vez finalizada la recepción el módulo vuelve al estado de bajo consumo.

El modo 3, deep sleep, inhabilita la transmisión y recepción inalámbrica y pone al módulo en un estado de bajo consumo. Es usado únicamente para configurar los parámetros del módulo a través del puerto serial.

3.1.2 Configuración de los parámetros

Como se aclaró anteriormente, el módulo debe ser configurado en el modo 3. Para poder leer y escribir los registros de 8 bits del módulo, la comunicación por UART debe realizarse a 9600 baudios y con formato 8N1 (sin bit de paridad).

El formato usado para la lectura y escritura de los registros se describe a continuación. Para establecer el valor de un registro se debe transmitir un mensaje en hexadecimal con el formato {0xC0, dirección de inicio, longitud, parámetros a escribir}, esto retornara un mensaje de la forma {0xC1, dirección de inicio, longitud, parámetros escritos}. Por ejemplo, si se quiere escribir el parámetro 0x09 en la dirección 0x04 se debe enviar el mensaje {0xC0, 0x04, 0x01, 0x09} y si la escritura fue exitosa el módulo retornará {0xC1, 0x04, 0x01, 0x09}. En caso de que se produzca un fallo el módulo retornará {0xFF, 0xFF, 0xFF}.

De manera similar, para leer el valor de los registros se debe enviar un mensaje con formato {0xC1, dirección de inicio, longitud} y el módulo retornará el mensaje {0xC1, dirección de inicio, longitud, parámetros}. Existe la posibilidad de establecer el valor de los parámetros de forma temporaria, es decir, los valores se establecen durante un ciclo de operación y se restablecen a sus valores anteriores cuando el módulo se apaga. Los comandos para establecer parámetros temporarios tienen el formato {0xC2, dirección de inicio, longitud, parámetros temporarios}, mientras que la respuesta del módulo tendrá el mismo formato que para los casos anteriores.

Seguidamente, se listan los registros que se pueden modificar y la función que cumplen:

- ADDH y ADDL: 2 registros (16 bits) para definir la dirección del módulo.
- REG0: 3 bits para definir la tasa de datos de la comunicación serial y 2 bits para definir el bit de paridad. Los 3 bits restantes definen la tasa de bits de la comunicación inalámbrica.
- REG1: 2 bits para definir la longitud en bytes de los paquetes en los que se divide el mensaje a transmitir. 1 bit para activar o desactivar la lectura del ruido ambiental. 2 bits para definir la potencia de transmisión. 4 bits reservados.
- REG2: 8 bits para definir el canal (frecuencia) en el que operará el módulo.
- REG3: 1 bit para activar el indicador de la intensidad de señal del mensaje recibido (*Received Signal Strength Indicator*, RSSI). 1 bit para definir el modo de transmisión (fijo o broadcast). 1 bit para activar el modo LBT. 3 bits para definir el ciclo WOR (cada cuanto el módulo verifica si recibió algún mensaje cuando está en el modo 2). 2 bits reservados.
- CRYPT_H y CRYPT_L: 2 registros de solo escritura en los que se almacena la clave usada para encriptar los datos.

Con el fin de establecer los parámetros del módulo, este se conectó a una computadora personal (PC) por medio de un convertidor UART-USB. El convertidor está basado en el CI CP2102, una imagen de este convertidor se muestra en la Fig. 25.

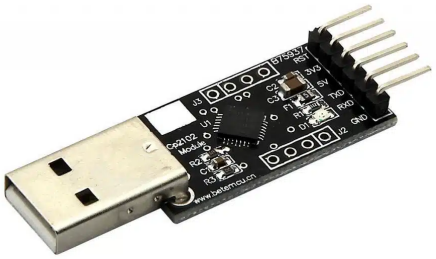


Figura N° 25: Convertidor UART-USB basado en el CP2102. Fuente: [21]

.

El convertidor se conectó a la PC y al módulo siguiendo el diagrama de la Fig. 26. Para enviar los mensajes en hexadecimal al módulo y observar su respuesta se usó el software de código abierto SerialTool. De forma alternativa, es posible realizar esto con cualquier otra interfaz UART, por ejemplo, usando un microcontrolador.

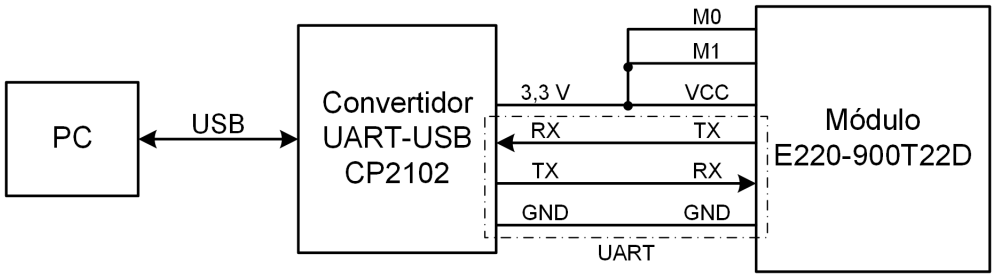


Figura N° 26: Conexiones usadas para configurar el módulo LoRa con una PC.

3.2 Transmisor

El módulo transceptor que cumple la función de transmisor funcionará de manera permanente en el modo 1 (transmisor WOR). Se configuró el módulo de manera que el modo de transmisión sea fijo (punto a punto), en una frecuencia de 915 MHz y con una potencia de 22 dBm (la máxima disponible). El ciclo WOR se definió en 2000 ms y la tasa de bits se estableció en 2,4 kbps. Como se vio en apartados anteriores, en LoRa una baja de tasa de datos permite reducir el ancho de banda y aumentar la distancia para la comunicación.

Dicho módulo está colocado en una PCB, cuyo esquemático se puede ver en la Fig. 27, se comunica con el microcontrolador y recibe alimentación (5 V) por medio de un cable de pares trenzados. Esta PCB está pensada para colocarse en altura, incorpora el módulo transceptor, un conector RJ45 hembra, un LED indicador de encendido y un LED para indicar la transmisión o recepción. El transistor funciona como inversor y enciende el LED cuando el pin AUX está en estado bajo.

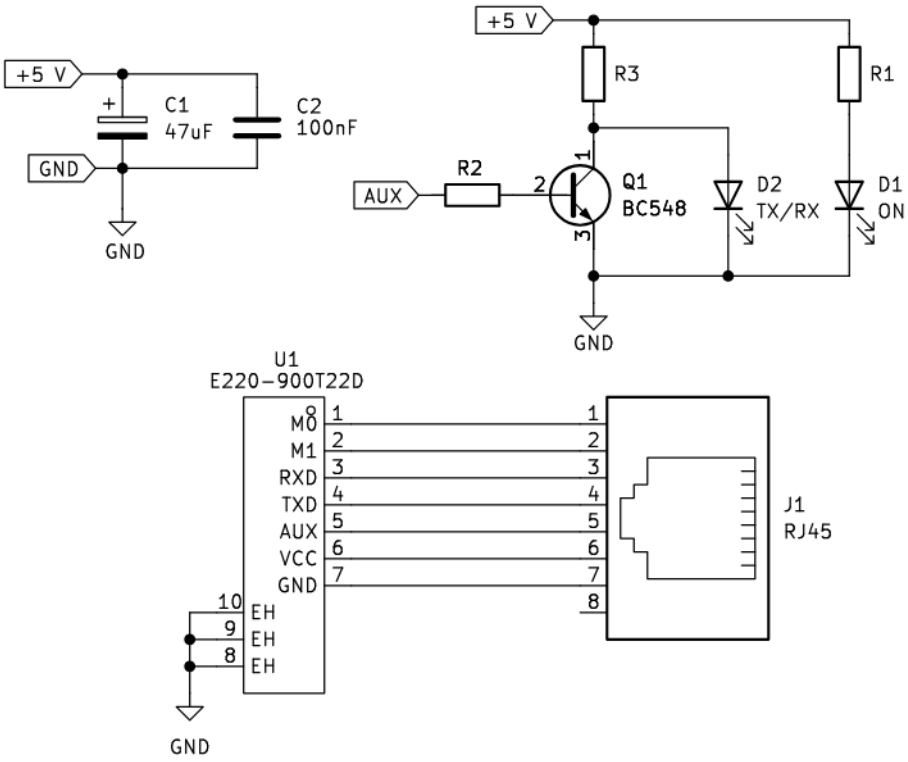


Figura N° 27: Esquemático de la PCB en la que se encuentra el módulo transmisor.

La antena usada tanto en el transmisor como en el receptor son del tipo "*rubber duck*" monopolo omnidireccionales. Tienen conectores SMA macho, una impedancia de 50 Ω, una ganancia de 3 dBi y permiten transmitir una potencia máxima de 10 W.

Este tipo de antenas son eléctricamente cortas, es decir, la longitud de la antena es menor que un cuarto de la longitud de onda de la señal. Es un tipo de antena helicoidal, en la que el helicoide se encuentra sellado con una cubierta de goma o plástico para protegerlo. Estas antenas se muestran en sus versiones para distintas frecuencias en la Fig. 28, donde también se puede observar el helicoide interno.

(a) (b)

Figura N° 28: Antena monopolo omnidireccional con conector SMA macho usada en los transceptores. (a) Antena de 915 MHz. (b) Antenas de distintas frecuencias del mismo fabricante y helicoide interno. Fuente: [22].

Son usadas principalmente en equipos portátiles de frecuencias VHF y UHF debido a que resultan ser más cortas que las antenas de látigo, las cuales tienen una longitud de un cuarto de onda. Sus aplicaciones están en radios y otros transceptores portátiles. Antiguamente se usaban en teléfonos celulares. Su longitud suele ser de entre el 4% y el 15% de la longitud de la onda de la señal, pero como contraparte suelen tener menor ganancia que las antenas de un cuarto de onda debido a las pérdidas que presentan.

3.3 Receptor

El módulo transceptor que se usa como receptor funciona en el modo 2 (receptor WOR), de esta manera, se encuentra en un modo de bajo consumo mientras no recibe información. Los datos recibidos por el transceptor serán procesados en un microcontrolador y subidos a una base de datos.

Debido a que es posible realizar comunicaciones por WiFi sin componentes adicionales, se eligió usar un módulo ESP-WROOM-32 en una placa de desarrollo ESP32 DevKit V1. Este módulo incorpora un microcontrolador ESP32-D0WDQ6, una memoria flash externa, un cristal oscilador, una antena para permitir las comunicaciones por WiFi o Bluetooth y otros componentes auxiliares. Una foto de este módulo en la placa de desarrollo mencionado se muestra en la Fig. 29.

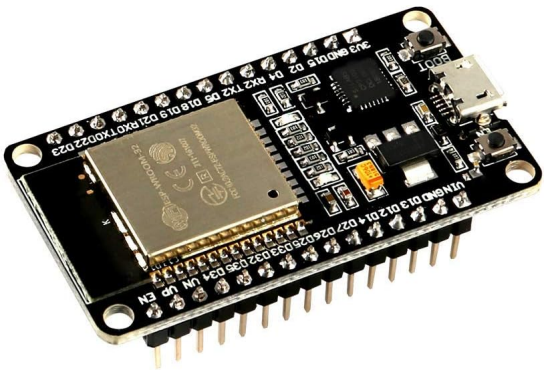


Figura N° 29: Placa de desarrollo ESP32 Devkit V1 con módulo ESP32-WROOM-32. Fuente: [23].

El ESP32-D0WDQ6 cuenta con un procesador basado en una arquitectura Xtensa LX6 de 32 bits, que puede alcanzar frecuencias de reloj de hasta 240 MHz. Dispone de una memoria principal de 520 KB del tipo SRAM y una memoria secundaria de 448 KB, además de soporte para una memoria externa. Integra 32 pines GPIO, 2 ADCs de 12 bits, 1 DAC de 8 bits, 8 temporizadores e interfaces de comunicación UART, SPI, I2C, I2S y CAN. Puede generar señales PWM y cuenta con sensores táctiles capacitivos. Su característica principal es que incorpora un subsistema de radiofrecuencia que permite la comunicación inalámbrica por WiFi o Bluetooth sin componentes externos adicionales. [24].

3.3.1 Alimentación del receptor

La placa del resistor está alimentada por dos celdas de iones de litio 18650 en paralelo, las cuales pueden operar de forma segura en un rango de 4,2 V hasta 2,4 V. El módulo transceptor puede operar con tensiones de alimentación desde 2,3 V hasta 5,5 V, por otra parte, el módulo ESP32-WROOM-32 no puede ser alimentado directamente desde las celdas a raíz de que soporta una tensión de alimentación en el rango de 3 V hasta 3,6 V. La placa ESP32 DevKit V1 incorpora un regulador de tensión lineal AMS1117 de 3,3 V; sin embargo, la tensión de *dropout* típica de estos reguladores para una corriente de 300 mA es de 0,95 V [25], esto resulta en una tensión de salida demasiado baja para alimentar al microcontrolador.

Debido a este motivo, se decidió alimentar al transceptor E220-900T22D directamente desde las celdas, mientras que para alimentar al ESP32-WROOM-32 se usa un regulador lineal AP2112K de 3,3 V. Este regulador presenta mejores características que el AMS1117, puede entregar una corriente de salida de 600 mA y tiene una tensión de dropout de apenas 125 mV para una corriente de 300 mA [26].

Se incorporó a la placa del receptor un módulo de carga para las celdas de litio basado en el CI TP4056, cuya apariencia se observa en la Fig. 30, puede ser alimentado mediante cualquier fuente de 5 V con un conector USB tipo C y entrega una corriente de carga máxima de 1 A. El módulo también incorpora un LED rojo para indicar que las celdas se están cargando y un LED azul para indicar que el proceso de carga finalizó.

Este dispositivo incluye un CI DW01A y 2 MOSFET para proteger las celdas y prolongar su vida útil, en conjunto con el TP4056 todos estos componentes ofrecen las siguientes funcionalidades [27]:

- Gestión de los ciclos de carga de corriente constante y tensión constante necesarios en celdas de litio.
- Prevención de sobre-descargas, desconectando las celdas de la carga cuando su tensión llega a 2,4 V.
- Protección ante sobrecargas, deteniendo el proceso de carga cuando la batería llega a una tensión de 4,2 V.
- Protección ante cortocircuitos o sobrecorrientes (para corrientes mayores a 3 A).
- Limitación de la corriente de encendido de la carga conectada a la batería.
- Carga adecuada para reacondicionar celdas sobre-descargadas.

Un diagrama general de los componentes y conexiones del sistema de alimentación y de carga de las celdas usado en el receptor se puede observar en la Fig. 31.

Figura N° 30: Módulo de carga y protección para baterías de litio basado en el TP4056. Fuente: [27].

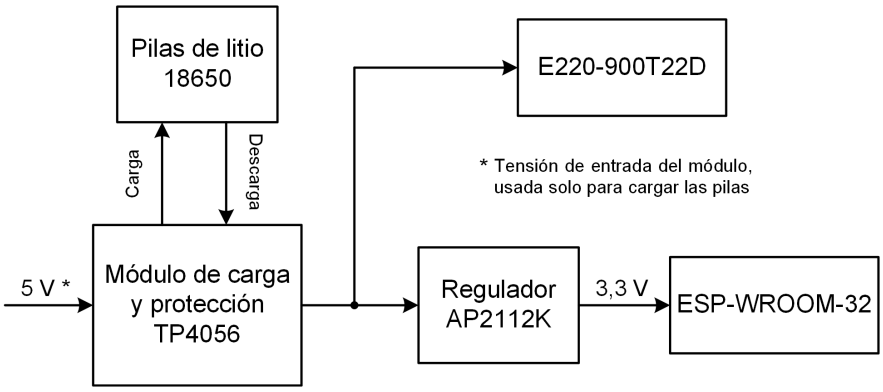


Figura N° 31: Diagrama general del sistema de alimentación usado en el receptor.

3.3.2 Modos de bajo consumo

Tanto el módulo LoRa como el microcontrolador del receptor cuentan como modos de bajo consumo, una característica de suma importancia en esta placa al estar alimentada por una batería (dos celdas). El módulo E220-900T22D entra en el estado de bajo consumo en el modo 2 y 3 (ver Tabla N° 1). Para el modo 3, este es el estado por defecto siempre que el módulo no esté realizando comunicaciones por el puerto serial o configurando algún parámetro. En el modo 2, el módulo se encuentra en el estado de bajo consumo hasta que comienza a recibir el mensaje de un módulo operando en el modo 1, una vez finalizada la recepción del mensaje, el módulo vuelve al estado de bajo consumo. El fabricante asegura que en el modo de bajo consumo la corriente requerida por el módulo es de 5 µA [19].

Los módulos ESP32 soportan 3 modos de bajo consumo. El primero de ellos, *modem sleep* desactiva las funciones inalámbricas del módulo; en el modo *light sleep*, además de desactivar las funciones inalámbricas, también se interrumpe la señal de reloj para los periféricos, el procesador y las memorias principales; en el modo *deep sleep*, que es el elegido para esta aplicación, se apagan el procesador, las memorias y los periféricos. El consumo aproximado del módulo en este modo es de 100 µA. Los únicos bloques que permanecen activos en este módulo son el coprocesador de ultra baja potencia (ULP, *Ultra-Low Power*) y el reloj en tiempo real (RTC, *Real-Time Clock*) y sus periféricos [28].

3.3.3 Duración de la batería del receptor

Los sistemas alimentados por baterías generalmente operan en un modo pasivo, es decir, un modo en el que se hace énfasis en disminuir el consumo, el sistema tiene periodos de inactividad interrumpidos por eventos de alta prioridad que demandan grandes consumos de energía. Por otra parte, en el modo activo los sistemas procesan las distintas tareas de forma continua sin periodos de inactividad [29].

Para calcular la duración aproximada de la batería se divide el consumo total en tres categorías, consumo de secuencias periódica Q_P, consumo de secuencias asincrónicas (con un número previsto de ocurrencias) Q_A y consumo continuo en reposo (en modos de bajo consumo) Q_C. La carga total Q_L entregada por la batería, desde que estaba completamente cargada hasta descargarse, se puede expresar según la Ec.20.

$$Q_L = Q_C + \sum Q_P + \sum Q_A \tag{20}$$

En este caso se asume que no hay eventos asincrónicos, por lo que se omite su definición. La expresión para la carga consumida durante las secuencias periódicas se puede observar a continuación en la Ec.21.

$$Q_P = \frac{t_L}{T_P} \sum_n I_n \Delta t_n \tag{21}$$

Donde t_L es el tiempo de duración total de la batería y T_p es la duración de cada ciclo. Dentro de la sumatoria los productos $I_n \Delta t_n$ son los distintos consumos que ocurren en cada ciclo cuando el sistema está fuera del modo de bajo consumo.

Por otra parte, la carga consumida en reposo se expresa según la Ec.22, donde I_c es la corriente total en el modo de bajo consumo.

$$Q_C = I_C t_L \tag{22}$$

A partir de las expresiones anteriores se puede encontrar la duración aproximada de la batería en el sistema según la Ec.23.

$$t_L = \frac{Q_L}{\frac{1}{T_P} \sum_n I_n \Delta t_n + I_C} \tag{23}$$

Las dos celdas 18650 tienen una carga de 3200 mAh, se suponen ciclos de 15 minutos en los que, durante 14 minutos y 30 segundos, el sistema se encuentra inactivo y el tiempo restante realiza las distintas tareas asignadas por el programa.

En el periodo activo del sistema el módulo ESP32 sale de su modo de bajo consumo y, con el periférico WiFi activo, el consumo promedio es de 250 mA. Para este cálculo se desprecia el consumo de 17 mA del módulo transceptor al recibir la información ya que el tiempo en que ocurre es menor a 10 ms. En el modo de bajo consumo la corriente aproximada requerida por el sistema es de 105 µA. Con esta información se calcula que la duración estimada de la batería del sistema es de 758,93 h (31 dias y 15 horas), como se puede ver en el desarrollo de la Ec.24.

$$t_L = \frac{6400\,mAh}{\frac{1}{0.25\,h}250\,mA \cdot 1/120\,h + 0.105\,mA} = 758,93\,h \tag{24}$$

3.3.4 Programa implementado en el receptor

El programa implementado en el microcontrolador del receptor se explica a grandes rasgos en el diagrama de flujo de la Fig. 32. Uno de los pines GPIO del microcontrolador asociado al RTC (el único bloque activo en el modo deep sleep) se conectó al pin AUX del módulo transceptor con el objetivo de usar la señal auxiliar para despertar al microcontrolador.

El microcontrolador se encuentra en el modo deep sleep hasta que un flanco descendente del pin AUX inicia el programa. Se comienza inicializando los periféricos y pines necesarios, se realiza la conexión a una red WiFi y se sincroniza la hora a la zona horaria UTC-3. Luego, se incializa la conexión con la base de datos en tiempo real (*Realtime Database*, RTDB) y se configuran el certificado y los parámetros necesarios para realizar una conexión segura entre el módulo ESP32 y la base de datos.

Se reciben todos los datos enviados por el transmisor, tensión de las baterías, corriente de carga de las baterías, corriente hacia el inversor, temperatura del microcontrolador del transmisor y RSSI (agregado por el módulo luego de recibir el mensaje). Se lee la temperatura interna del microcontrolador del ESP32, la tensión de las celdas 18650 y el ruido ambiental. Finalmente, se crea una etiqueta de tiempo (*timestamp*), se envía toda la información a la base de datos y el sistema vuelve al modo de bajo consumo.

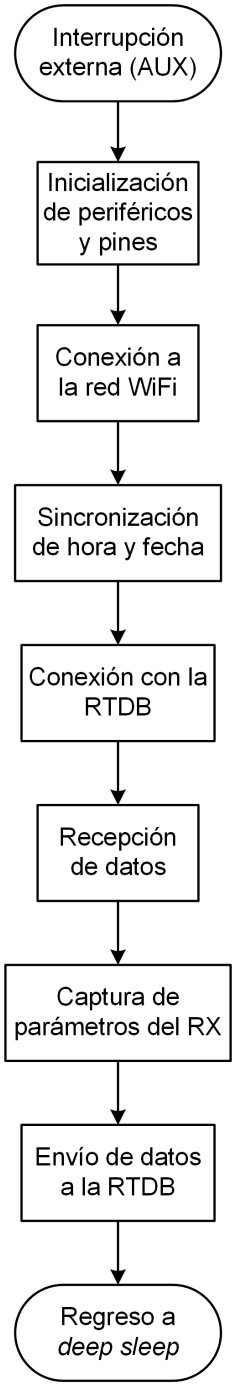


Figura N° 32: Diagrama de flujo del programa implementado en el receptor.

CAPITULO 4: Aplicación para teléfonos celulares

4.1 Firebase

Firebase es una plataforma de desarrollo de aplicaciones adquirida por Google, la cual tiene como objetivo simplificar el proceso de desarrollo de aplicaciones web, móviles e IoT. Firebase ofrece servicios en la nube listos para usar en una aplicación, como pueden ser bases de datos, autenticación de usuarios, almacenamiento de archivos, *hosting*, envío de notificaciones, entre otros [30].

El servicio usado en este trabajo es la RTDB, una base de datos NoSQL alojada en la nube. Los datos se almacenan en un modelo de árbol JSON jerárquico en pares clave-valor al que los clientes pueden acceder para leer, escribir o ambas operaciones de manera simultánea. De forma gratuita se puede obtener 1 GB de almacenamiento, con 100 conexiones simultáneas y un límite de descarga de datos de 10 GB por mes.

El proceso implementado en el microcontrolador para subir los datos a la RTDB consiste en crear un objeto JSON con todos los datos mencionados en el capitulo anterior, más la timestamp y son enviados a la base de datos mediante solicitudes HTTPS asincrónicas. De esta forma, el valor de cada clave JSON enviada será otras sub-claves con valores correspondientes a los enviados desde el microcontrolador y siendo uno de los pares clave-valor la timestamp en formato UNIX (segundos desde el 1 de enero de 1970 UTC). Esta etiqueta permite buscar información según la hora y fecha en la que se recibió, por ejemplo, es posible buscar todos los datos recibidos en un mes o los que se recibieron en un día en específico.

En la Tabla 2 se muestra el ejemplo de un objeto JSON generado en el microcontrolador del receptor con los distintos valores adquiridos.

Tabla N° 2: Ejemplo de un objeto JSON enviado a la base de datos de Firebase.


```
{
    “battCurrent”: 10.24,
    “battVoltage”: 25.43,
    “invCurrent”: 4.22,
    “noise”: 161,
    “rxBattery”: 4.12,
    “temperatureESP”: 48.66,
    “temperatureSTM”: 34.85,
    “RSSI”: 249,
    “timestamp”: 1754249214
}
```

4.2 Desarrollo de la aplicación en Kodular

Con el fin de desarrollar una aplicación para teléfonos celulares con sistema operativo Android se usó la plataforma de desarrollo Kodular. Esta herramienta se deriva del proyecto MIT App Inventor, se basa en un sistema de programación gráfica donde las instrucciones se representan como bloques que se conectan entre si, simplificando la programación. Las interfaces se crean de manera visual arrastrando componentes como botones, texto, etc [31]. Kodular convierte automáticamente el código de la aplicación programada en bloques a un programa en lenguaje Java y lo compila para generar archivos .apk o .aab.

Un ejemplo de código desarrollado en Kodular se puede ver en la Fig. 33, esta es una función que toma dos fechas usadas para indicar un rango de tiempo en el que se obtienen los datos almacenados en la RTDB de Firebase y se verifica que no se elijan fechas erróneas o posteriores a la fecha actual.

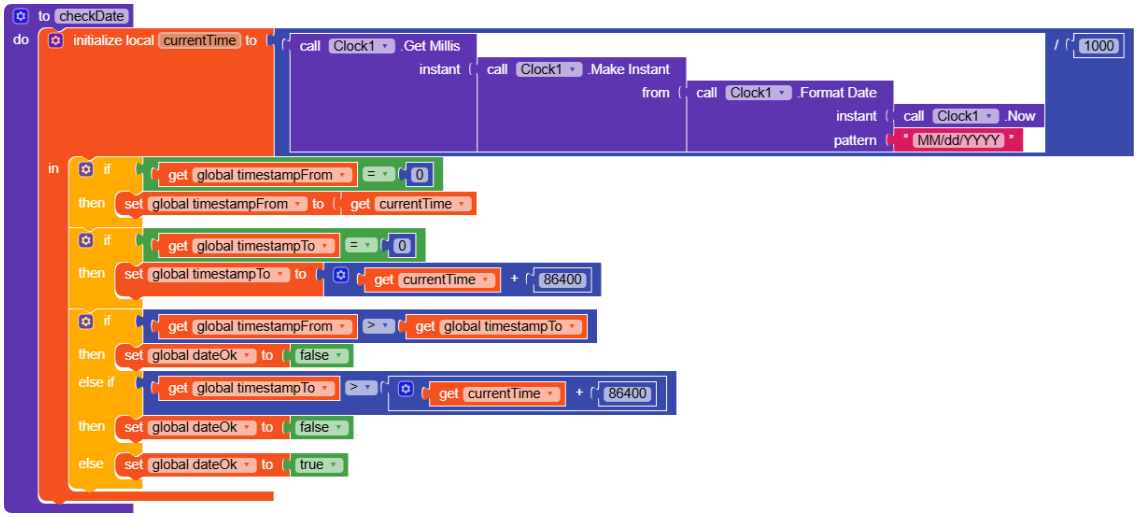
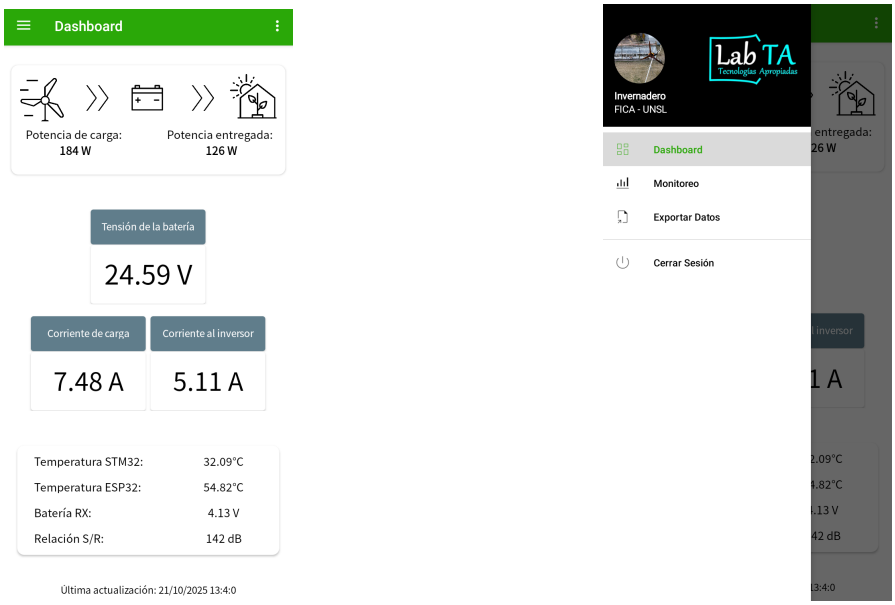


Figura N° 33: Función en Kodular para validar un rango de fechas seleccionado.

La aplicación tiene como finalidad visualizar los datos subidos por el microcontrolador a la RTDB de Firebase. Al ingresar a la aplicación se muestra la pantalla de inicio de sesión de la Fig. 34, en donde se debe ingresar la combinación de usuario y contraseña correcta para poder acceder a la visualización de los datos, también es posible seleccionar la opción de ”Recordarme” para evitar tener que iniciar sesión nuevamente en el dispositivo.

Una vez verificados los datos del usuario, se inicializa la pantalla donde se visualizará la información. Esta pantalla cuenta con tres vistas, siendo la vista ”*Dashboard*” la seleccionada por defecto al abrir la aplicación, en ella se visualizan los datos recibidos más recientes. Una imagen de ejemplo de esta vista se muestra en la Fig.35, junto con el menú desplegable usado para cambiar entre las diferentes vistas.

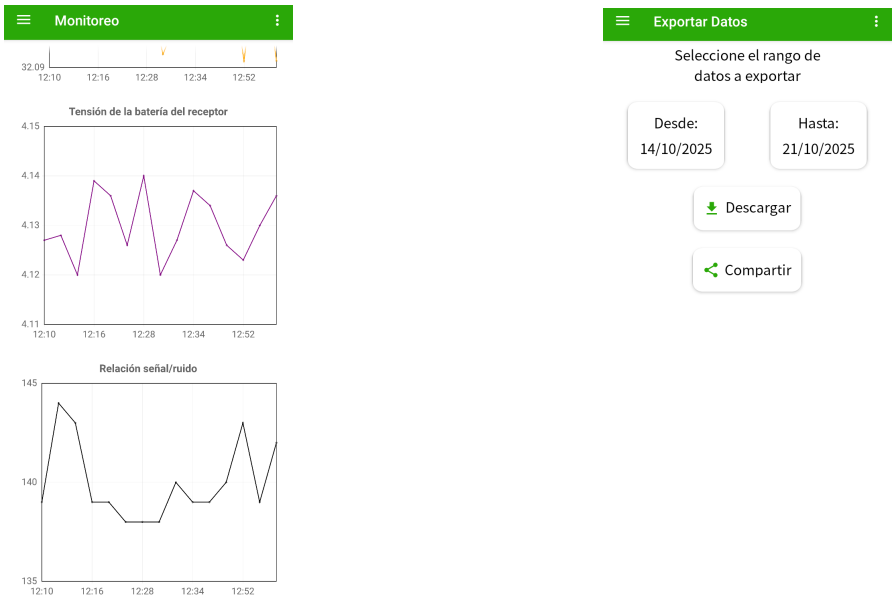
Figura N° 34: Pantalla de inicio de sesión de la aplicación para teléfonos celulares.



(a) (b)

Figura N° 35: Pantalla principal de la aplicación. (a) Vista ”Dashboard”. (b) Menú lateral para desplazarse entre las vistas disponibles.

La vista ”Monitoreo” muestra en gráficas los valores de los parámetros recibidos durante la última hora. En la vista ”Exportar Datos” se puede seleccionar un rango de tiempo y obtener los datos subidos a la RTDB durante ese periodo, siendo posible descargarlos al dispositivo o compartirlos, por ejemplo, a través de correo electrónico usando la funcionalidad nativa de Android. Estas dos vistas se muestran en la Fig. 36. Los datos son exportados en formatos .csv (valores separados por comas), esto permite importarlos y procesarlos fácilmente en cualquier software para cálculos estadística, Excel, MATLAB, etc.

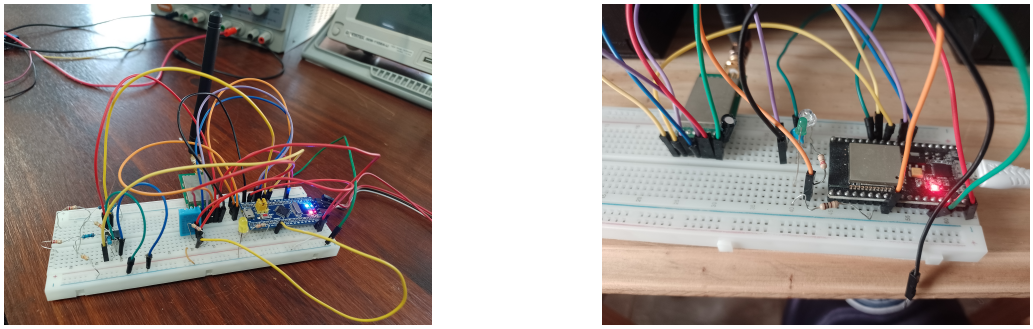


(a) (b)

Figura N° 36: Vistas de la pantalla principal de la aplicación. (a) Vista ”Monitoreo”. (b) Vista ”Exportar”.

CAPITULO 5: Resultados experimentales

Una vez definidos los circuitos de medición y comunicación y los componentes a utilizar, se procedió con las pruebas experimentales en protoboard. En la Fig. 37 se observan los circuitos de medición y transmisión en una protoboard, mientras que el circuito del receptor se colocó en otra protoboard.



(a) (b)

Figura N° 37: Pruebas experimentales en protoboard. (a) Circuito de medición y transmisión de datos. (b) Circuito receptor.

5.1 Caracterización del prototipo

Con el fin de conocer el desempeño del sistema de medición de tensión y corriente se realizó la caracterización del prototipo, una representación del circuito usado para esto se muestra en la Fig. 38. Cómo se explicó en el capitulo 2, las mediciones se realizan de manera indirecta, la tensión de las baterías es obtenida a partir de la suma de dos tensiones y ambas corrientes se miden por medio de la caída de tensión provocada por la corriente del secundario del transformador de corriente en un resistor.

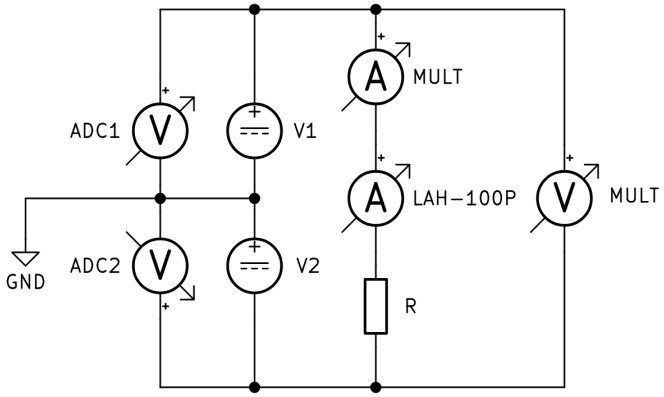


Figura N° 38: Diagrama del circuito usado para caracterizar el sistema de medición.

Para simular el comportamiento de las 2 baterias se utilizó una fuente de alimentación de laboratorio doble, que cuenta con la posibilidad de colocar las fuentes independientes en serie. Se usó un banco de resistencias de laboratorio como carga, de modo que la corriente que circulaba por la carga simulaba la corriente hacia el inversor. En la Fig. 39 se aprecia el banco de resistencias y la configuración usada para llevar a cabo las pruebas mencionadas. La caracterización para el sistema de medición de corriente se realizó con un único transformador y valor de referencia, es decir, no se realizaron mediciones simulando la carga de las baterías; se asume que las diferencias que pueden existir entre el sistema de medición de la corriente hacia el inversor y el de medición de la corriente de carga son despreciables.

Figura N° 39: Banco de resistencias y configuración usada para llevar a cabo la caracterización del prototipo.

Como referencia de valores reales se utilizaron las lecturas de un multímetro UNI-T UT70D para la corriente y las lecturas de un multímetro tipo pinza UNI-T UT210C para la tensión. En la Fig. 40 se pueden ver imágenes de estos instrumentos.

(a) (b)

Figura N° 40: Instrumentos usados para obtener los valores reales en las mediciones. (a) UNI-T UT70D, usado para medir corriente. Fuente: [32]. (b) UNI-T UT210C, usado para medir tensión. Fuente: [33].

El rango de medición del sistema está determinado por los valores del divisor de tensión para la tensión de las baterías, con los valores mencionados en el capitulo 2 la máxima tensión admisible es de 30 V. El rango de la corriente está determinado, en este caso, por la PCB, se definió que la corriente máxima admisible será de 45 A. En la Tabla 3 se definen los rangos del sistema de medición.

Tabla N° 3: Rangos de medición del sistema.

Tensión [V]	Corriente [A]	Potencia [W]
0 V - 30 V	0 A - 45 A	0 W - 1350 W

5.1.1 Exactitud

Con el objetivo de conocer la exactitud del sistema se realizaron 30 lecturas y se registró el valor real y el valor obtenido por el sistema, comenzando con la tensión de la fuente en 1 V y aumentando alrededor de 1 V en cada medición. La comparación entre los valores reales y medidos de tensión y corriente se listan en la Tabla 4. A partir de estos valores se calcula la potencia disipada en el banco de resistencias como el producto entre la tensión y la corriente, estos valores se listan en la Tabla 5.

Tabla N° 4: Comparación entre valores reales y medidos de corriente y tensión.

N°	Corriente real [A]	Corriente medida [A]	Tensión real [V]	Tensión medida [V]
1	0,16	0,21	1,02	0,64
2	0,33	0,31	2,07	1,64
3	0,48	0,46	2,99	2,61
4	0,64	0,64	4,00	3,68
5	0,81	0,84	5,01	4,75
6	0,97	1,01	6,03	5,82
7	1,12	1,17	6,97	6,80
8	1,30	1,36	8,04	7,90
9	1,46	1,52	9,03	8,91
10	1,62	1,68	10,04	9,94
11	1,77	1,83	11,01	10,93
12	1,94	1,98	12,03	11,97
13	2,09	2,10	13,00	12,94
14	2,25	2,22	13,98	13,95
15	2,42	2,37	15,04	15,03
16	2,58	2,53	16,02	16,00
17	2,74	2,69	16,99	16,98
18	2,91	2,88	18,04	18,03
19	3,06	3,05	19,01	19,01

Continúa en la siguiente página

N°	Corriente real [A]	Corriente medida [A]	Tensión real [V]	Tensión medida [V]
20	3,23	3,23	20,02	20,04
21	3,38	3,41	21,00	21,00
22	3,56	3,61	22,09	22,12
23	3,70	3,73	22,95	22,96
24	3,88	3,91	24,03	24,06
25	4,04	4,07	25,02	25,07
26	4,20	4,24	25,92	26,03
27	4,35	4,37	27,01	27,03
28	4,51	4,54	27,92	27,95
29	4,67	4,74	28,95	29,00
30	4,79	4,83	29,74	29,76

Tabla N° 5: Comparación entre valores reales y medidos de potencia.

N°	Potencia real [W])	Potencia medida [W]
1	0,16	0,13
2	0,68	0,51
3	1,44	1,20
4	2,56	2,36
5	4,06	3,99
6	5,85	5,88
7	7,80	7,96
8	10,45	10,74
9	13,19	13,55
10	16,28	16,69
11	19,50	19,99
12	23,36	23,71
13	27,17	27,12
14	31,45	30,99
15	36,39	35,61
16	41,37	40,81
17	46,57	45,66
18	52,54	51,98
19	58,19	58,06
20	64,27	64,33
21	70,98	71,62
22	78,56	79,79
23	85,92	85,57
24	93,29	94,09
25	101,08	101,99
26	108,86	110,31
27	117,09	117,97
28	126,00	127,00
29	135,23	137,46
30	142,49	143,65

Para determinar la exactitud se compararon las mediciones reales contra las obtenidas por el sistema de medición. Para ello, se comenzó calculando el error absoluto para cada medición a partir de la Ec.25.

$$e = |x_{medido} - x_{real}| \tag{25}$$

A raíz de que la potencia se obtiene a partir del producto mencionado, el error absoluto en la medición de potencia se obtiene usando el método de propagación de incertidumbre, como se puede ver en la Ec.26.

$$e_P = \left(\frac{\partial P}{\partial V}e_V + \frac{\partial P}{\partial I}e_I\right) = Ie_V + Ve_i \tag{26}$$

A partir de los errores absolutos se puede encontrar el error relativo porcentual, una medida cuantitativa de la exactitud, según la Ec.27.

$$\varepsilon = \frac{e}{xreal} \cdot 100\% \tag{27}$$

En el caso de la potencia se obtiene que el error relativo porcentual resulta ser la suma de los errores relativos de la tensión y corriente, como se observa en la Ec.28.

$$\varepsilon_P = \frac{Ie_V + Ve_i}{VI} = \frac{e_V}{V} + \frac{e_I}{I} = \varepsilon_V + \varepsilon_I \tag{28}$$

En la Tabla 6 se listan los errores relativos porcentuales de la corriente, tensión y potencia para las 30 mediciones realizadas, mientras que en la Fig. 41 se muestra esta información en un gráfico de barras para facilitar su visualización.

Tabla N° 6: Errores relativos porcentuales (ERP) de tensión, corriente y potencia.

N°	ERP tensión [%]	ERP corriente [%]	ERP potencia [%]
1	37.25	31.25	68.50
2	20.77	6.06	26.83
3	12.70	4.16	16.87
4	8.00	0.00	8.00
5	5.18	3.70	8.89
6	3.48	4.12	7.60
7	2.43	4.46	6.90
8	1.74	4.61	6.35
9	1.32	4.10	5.43

Continúa en la siguiente página

N°	ERP tensión [%]	ERP corriente [%]	ERP potencia [%]
10	0.99	3.70	4.69
11	0.72	3.38	4.11
12	0.49	2.06	2.56
13	0.46	0.47	0.94
14	0.21	1.33	1.54
15	0.06	2.06	2.13
16	0.12	1.93	2.06
17	0.05	1.82	1.88
18	0.05	1.03	1.08
19	0.00	0.32	0.32
20	0.09	0.00	0.09
21	0.00	0.88	0.88
22	0.13	1.40	1.54
23	0.04	0.81	0.85
24	0.12	0.77	0.89
25	0.19	0.74	0.94
26	0.42	0.95	1.37
27	0.07	0.45	0.53
28	0.10	0.66	0.77
29	0.17	1.49	1.67
30	0.06	0.83	0.90

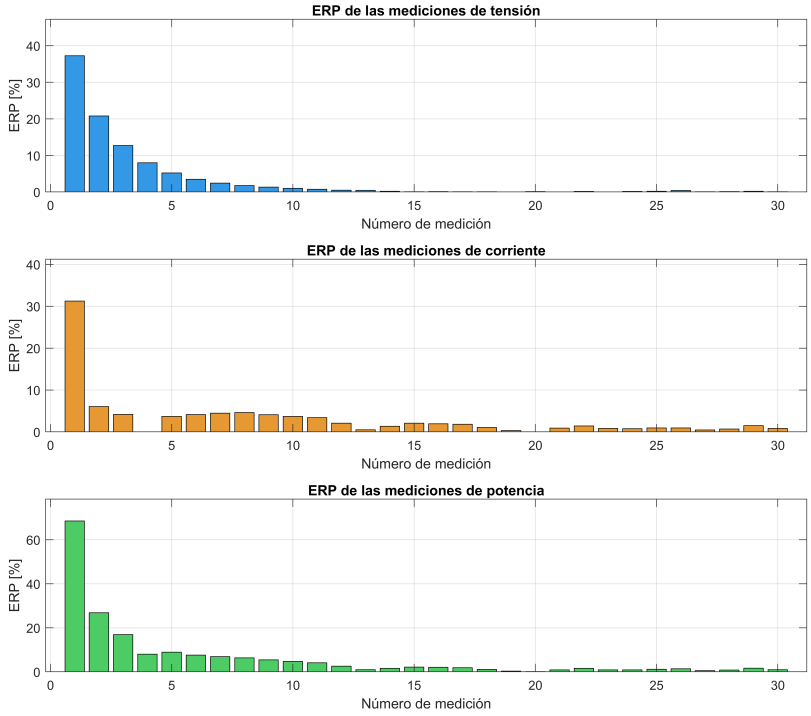


Figura N° 41: Errores relativos porcentuales de tensión, corriente y potencia representados en gráficos de barras.

5.1.2 Linealidad

Utilizando el conjunto de valores de la Tabla 4 y comparando los valores reales y medidos se calculó el coeficiente de correlación lineal de Pearson mediante la Ec.29 (donde $\bar{x}$ y $\bar{y}$ son las medias aritméticas de las variables) y se obtuvo la recta de regresión lineal, a fin de determinar la linealidad del instrumento.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \tag{29}$$

El coeficiente de Pearson para ambas variables resulto ser de 0,99, las gráficas de las rectas de regresión lineal se pueden ver en la Fig. 42

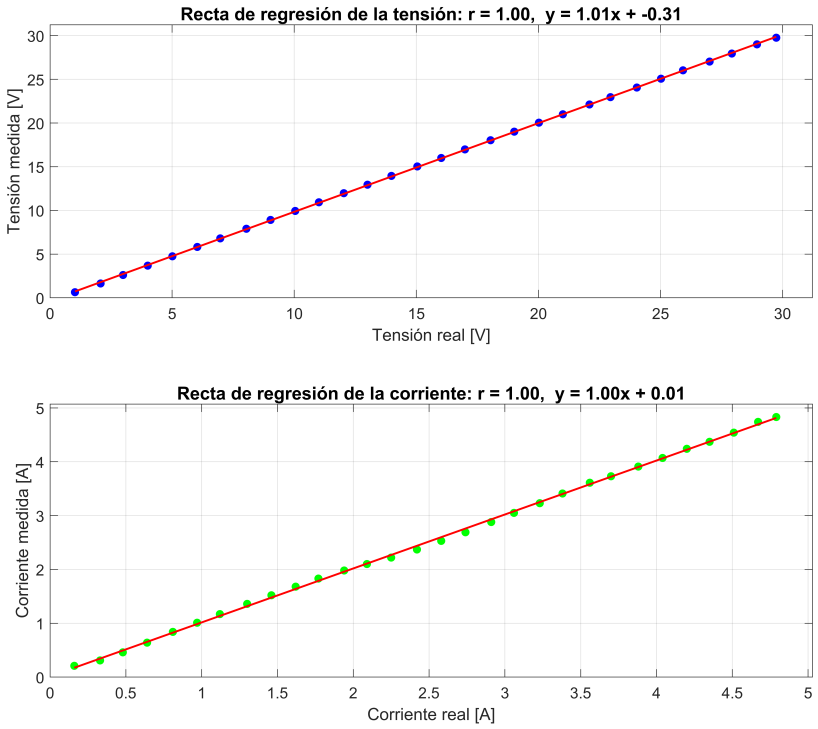


Figura N° 42: Rectas de regresión lineal para la tensión y corriente.

5.1.3 Precisión

Usando el circuito de la Fig. 38 se estableció la tensión de la fuente de alimentación en 25 V (valor real, obtenido con el instrumento de referencia) y la corriente resultante fue de 4,04. Se repitió la medición 10 veces usando el prototipo (los resultados se listan en la Tabla 7) y mediante la Ec.30 se calculó la desviación estándar, una medida estadística de la precisión. En la Tabla 8 se muestran los resultados obtenidos.

$$s = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \tag{30}$$

Tabla N° 7: Valores obtenidos al repetir la medición para una tensión de 25 V.

N°	Tensión [V]	Corriente [A]	Potencia [W]
1	25,03	4,02	100,61
2	24,97	4,06	101,36
3	25,05	4,05	101,51
4	25,00	4,03	100,75
5	25,04	4,01	100,66
6	24,97	4,06	101,64
7	25,00	4,05	101,41
8	24,95	4,02	100,29
9	25,06	4,04	101,42
10	25,01	4,06	101,56

Tabla N° 8: Desviación estándar de las mediciones de tensión, corriente y potencia.

Variable	Desviación estándar (s)
Tensión	0,04
Corriente	0,02
Potencia	0,50

5.1.4 Resolución digital y analógica

La resolución digital del prototipo está dada por el ADC utilizado, debido a que el ADC del STM32F103C8T6 tiene 12 bits, la resolución digital del sistema es de 4096 muestras, como se puede ver en la Ec.31.

$$\Delta x_{digital} = 2^1 2 = 4096 muestras \tag{31}$$

La resolución analógica, por otra parte, es el valor mínimo de tensión que el ADC detecta para cambiar de un valor discreto a otro, como la tensión de referencia del ADC mencionado es de 3,3 V, a partir de la Ec.32 se obtiene que la resolución analógica es de 0,81 mV.

$$\Delta x_{analgica} = \frac{3,3\,V}{2^1 2} = 0,81\,mV \tag{32}$$

En la Ec.33y Ec.34 se obtiene el mínimo valor de tensión y corriente que es posible medir según el acondicionamiento aplicado, a partir de estos valores en la Ec.35 se deriva el valor mínimo de potencia que se puede medir.

$$V_{min} = 0,81\,mV \cdot 0,22 = 0,17\,mV \tag{33}$$

$$I_{min} = \frac{0,81\,mV \cdot 45\,A}{1,1\,V} = 10,71\,mA \tag{34}$$

$$P_{min} == 1,82\,mW \tag{35}$$

5.2 Prueba de distancia

Si bien la distancia entre los dos puntos entre los que se realiza la comunicación es de alrededor de 300 m, con el objetivo de conocer las capacidades del sistema se realizó una prueba de la distancia máxima a la que es posible establecer una comunicación bidireccional.

Para llevar a cabo este proceso se dejó uno de los módulos transceptores LoRa (módulo 1 para distinción) en la rotonda de la Ruta Provincial 55 que se encuentra en las proximidades del peaje de Villa Mercedes, como se observa en la Fig. 43,. El módulo 2 se alejó y cada 200 metros se hizo un registro del indicador de fuerza de la señal recibida (*Received Signal Strength Indicator*, RSSI). El trayecto aproximado que se recorrió durante la prueba se muestra en la Fig. 44.

(a) (b)

Figura N° 43: Prueba de distancia del módulo LoRa. (a) El módulo 1 se colocó en la rotonda de la RP 55. (b) Rotonda de la RP 55, próxima al peaje de Villa Mercedes.

Figura N° 44: Trayecto aproximado recorrido con el módulo 2 durante la prueba de distancia.

En los módulos LoRa E220-900T22D el RSSI es un entero sin signo de 8 bits que aumenta con la potencia de la señal recibida, sin embargo, esta medida es adimensional y no se puede traducir directamente, por ejemplo, a un valor en dBm. En la Tabla 9 se muestran los valores obtenidos, mientras que en la Fig. 45 se muestra esta información de manera gráfica, donde se observa que el RSSI disminuye con cierta linealidad a medida que aumenta la distancia entre los módulos. A una distancia de 2 km se obtuvo la última recepción exitosa, por lo que no se continuó con las pruebas para mayores distancias.

Tabla N° 9: Valores de RSSI obtenidos durante la prueba de distancia.

Distancia	RSSI Módulo 1	RSSI Módulo 2
0,2 km	102	102
0,4 km	102	101
0,6 km	100	100
0,8 km	99	98
1,0 km	97	96
1,2 km	94	93
1,4 km	90	92
1,6 km	87	88
1,8 km	85	83
2,0 km	82	79

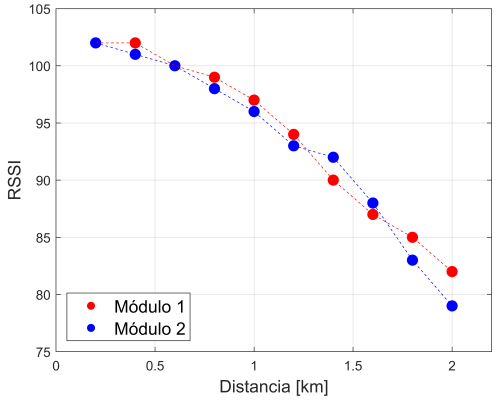
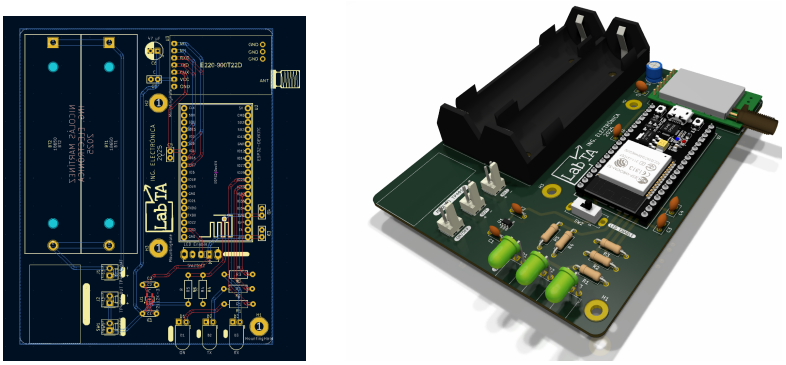


Figura N° 45: Relación entre el RSSI y la distancia entre los módulos.

5.3 Diseño y construcción de las PCB

Luego de realizar todas las pruebas experimentales se trasladaron todos los circuitos a sus respectivas PCB, para diseñar estas placas se hizo uso del software KiCad. En la Fig. 46 se muestra una captura de pantalla del proceso de diseño de la placa del receptor en dicho software y un renderizado en 3D de la PCB.



(a) (b)

Figura N° 46: Diseño de la PCB del receptor en KiCad. (a) Captura de pantalla del diseño. (b) Renderizado en 3D de la PCB.

Estos diseños se enviaron a una empresa manufacturera de PCB, en la Fig. 47 se pueden observar las 3 PCB con sus respectivos soldados, mientras que en la Fig. 48 se muestra el transmisor durante una prueba de funcionamiento. Como se aclaró anteriormente, una placa contiene el circuito de medición, otra el circuito transmisor (que recibe la información de la PCB del circuito de medición) y la restante contiene el circuito receptor.

Figura N° 47: Placas de circuito impreso con sus componentes soldados, placa de medición (abajo a la izquierda), placa de transmisión (arriba) y placa de recepción (abajo a la derecha).

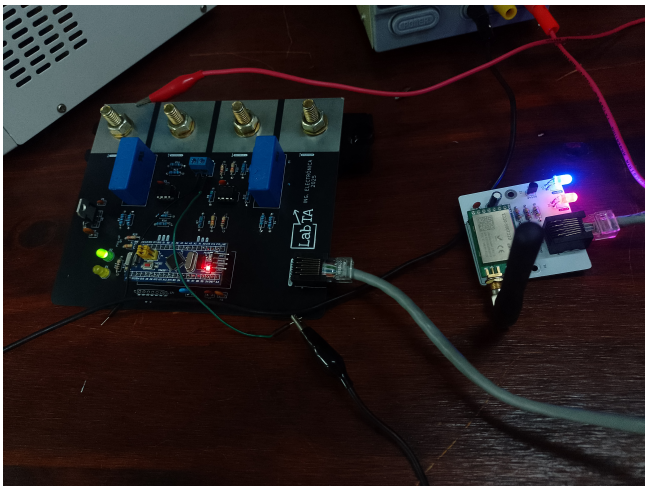


Figura N° 48: Pruebas de funcionamiento en la placa del transmisor.

5.4 Diseño de las carcasas y prueba de funcionamiento final

Las PCB del transmisor y receptor se encontrarán en el exterior, por lo que se decidió fabricar carcasas para protegerlas de los factores ambientales que puedan dañarlas. Las carcasas se diseñaron en el software Autodesk Fusion y se fabricaron con una impresora 3D. En la Fig. 49 se muestra una captura del diseño de la carcasa para la PCB del receptor y una imagen de dicha placa dentro de la carcasa.

(a) (b)

Figura N° 49: Diseño de las carcasas para las PCB. (a) Renderizado en 3D de la carcasa para la PCB del receptor. (b) PCB del receptor dentro de su carcasa.

Con las PCB y sus carcasas finalizadas se procedió a realizar una última prueba de funcionamiento para comprobar la correcta comunicación entre los módulos en las ubicaciones en las que se encontrarán finalmente, en la Fig. 50 se puede ver la placa del receptor en su carcasa durante dicha prueba en el invernadero.

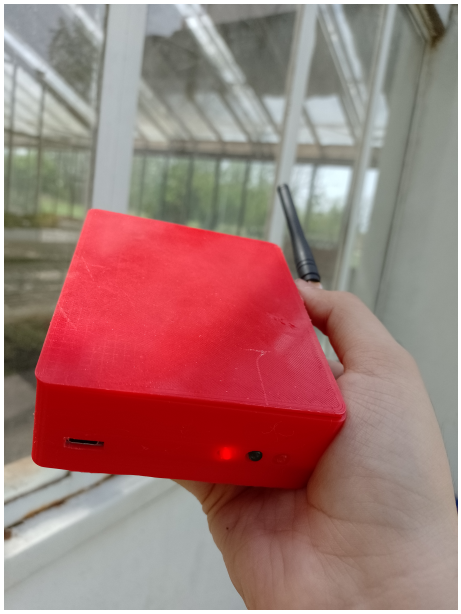


Figura N° 50: Pruebas de funcionamiento final con los prototipos finalizados.

CAPITULO 6: Conclusiones

Este trabajo tuvo como objetivos principales construir un sistema de adquisición de datos para medir variables eléctricas en sistemas de energías renovables, la transmisión de dichos datos por medio de una tecnología de baja potencia y largo alcance como LoRa y la visualización de los datos en una aplicación para teléfonos celulares.

El sistema de mediciones mostró una adecuada linealidad (coeficientes de Pearson iguales a 0,99) y una desviación estándar muy baja para esta aplicación, indicando una buena precisión. El error relativo porcentual resultó ser elevado para las mediciones iniciales, sin embargo, el rango de tensiones y corrientes que se suponen para las condiciones de funcionamiento normal se encuentran por encima de esos valores iniciales, por lo que se concluye que el sistema de mediciones presenta un buen desempeño para esta aplicación.

Los circuitos implementados en el transmisor y receptor del módulo E220-900T22D y la configuración seleccionada (frecuencia de 915 MHz, potencia de transmisión de 22 dBm y baja tasa de datos) proporcionan un alcance muy superior al objetivo inicial de diseño, que era alrededor de 300 m. Durante las pruebas experimentales se logro establecer una comunicación bidireccional en una distancia de hasta 2 km.

La aplicación desarrollada en Kodular, junto con el uso de la RTDB de Firebase, ofrecen una interfaz práctica para la visualización, exportación y consulta de datos, cumpliendo con el requisito de visualización remota y facilidad de uso.

Como trabajos futuros se pueden mencionar, entre muchos otros, la mejora del sistema de medición con componentes como, por ejemplo, un ADC dedicado o amplificadores de instrumentación; implementar técnicas de diseño para aumentar la corriente soportada por la PCB; la reducción en el tamaño de las PCB usando componentes de montaje superficial e incorporando los microcontroladores en la PCB en lugar de usar placas de desarrollo; la realización de pruebas de comunicación con criterios más exhaustivos y asegurando la linea de visión (sin obstaculos entre el transmisor y receptor) y mejoras en la aplicación, como implementar los servicios de autenticación de Firebase para proporcionar un ingreso seguro o la posibilidad de modificar el rango de visualización de datos u otros parámetros en la vista ”Monitoreo” de la pantalla principal.

Glosario

Término	Definición
ADC	Convertidor analógico-digital. Circuito usado para transformar una señal analógica en una digital representada con valores discretos.
AWGN	Ruido blanco gaussiano aditivo. Ruido aleatorio con distribución normal.
CSS	Ensanchamiento de espectro por barrido de frecuencia. Técnica de modulación utilizada en LoRa para mejorar la robustez y el alcance de una comunicación.
ERP	Error relativo porcentual. Indicador de la diferencia entre un valor medido y su valor real, es una medida de exactitud.
FM	Modulación en frecuencia. Técnica mediante la cual una señal con información modifica la frecuencia de una señal portadora.
FSK	Modulación por desplazamiento en frecuencia. Técnica digital en donde la modulación se realiza con cambios discretos en la frecuencia.
IoT	Internet de las cosas. Red de dispositivos físicos interconectados para adquirir y transmitir datos sin intervención humana.
ISM	Bandas industriales, científicas y médicas. Bandas de frecuencia de uso libre, comunes en tecnologías como LoRa.
LoRa	Acrónimo de largo alcance. Tecnología inalámbrica de largo alcance y bajo consumo de energía usada en redes IoT.
LoRaWAN	Protocolo de red que define la comunicación entre dispositivos LoRa y servidores de red.
LPWAN	Red de área amplia de baja potencia. Redes diseñadas para dispositivos que transmiten pequeñas cantidades de datos a largas distancias.
PCB	Placa de circuito impreso. Placa en donde se colocan y conectan todos los componentes de un circuito mediante pistas conductoras.
RF	Radiofrecuencia. Rango de frecuencias del espectro electromagnético en donde se realizan comunicaciones inalámbricas.
SNR	Relación señañl-ruido. Medida de la calidad de una señal en función de su potencia y del ruido presente en un canal.
UART	Receptor-transmisor asíncrono universal. Protocolo de comunicación serial utilizada en comunicaciones digitales de baja velocidad.

Referencias

[1] W. Tomassi, *Electronic Communication Sytems*, 5th ed. Upper Saddle River, NJ: Prentice Hall. 2003.

[2] F. J. Díaz. (Apr. 2019). Soluciones IoT con tecnología LoRaWAN. Presented at XXI Workshop Invest. Cienc. Comput. (WICC), UNSJ, San Juan, Argentina. [Online]. Available: https://sedici.unlp.edu.ar/handle/10915/77230.

[3] wikipedia.org. ”LoRa”. https://en.wikipedia.org/wiki/LoRa (accessed May 2, 2025).

[4] L. E. Frenzel Jr, *Principles of Electronic Communication Systems*, 4th ed. New York, NY: McGraw Hill. 2016.

[5] Semtech Corporation. “AN1200.22 LoRa™ Modulation Basics”.

https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf (accessed May 10, 2025).

[6] S. Montagny, *LoRa - LoRaWAN and Internet of Things*. Haute-Savoie, France: 2022. Accessed: May 11, 2025. [Online]. Available:https://www.univ-smb.fr/lorawan/wp-content/uploads/2022/01/Book-LoRa-LoRaWAN-and-Internet-of-Things.pdf

[7] Semtech Corporation. “LoRa® and LoRaWAN®”. semtech.com. https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf (accessed May 10, 2025).

[8] A. Maleki, H. H. Nguyen, E. Bedeer and R. Barton, ”A Tutorial on Chirp Spread Spectrum Modulation for LoRaWAN: Basics and Key Advances,” in IEEE Open Journal of the Communications Society, vol. 5, pp. 4578-4612, 2024, doi: 10.1109/OJCOMS.2024.3433502.

[9] L. Vangelista, ”Frequency Shift Chirp Modulation: The LoRa Modulation,” in IEEE Signal Processing Letters, vol. 24, no. 12, pp. 1818-1821, Dec. 2017, doi: 10.1109/LSP.2017.2762960.

[10] U. Noreen, A. Bounceur and L. Clavier, ”A study of LoRa low power and wide area network technology,” 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), Fez, Morocco, 2017, pp. 1-6, doi: 10.1109/ATSIP.2017.8075570.

[11] Keysight Technologies. ”Fundamentals of RF and Microwave Noise Figure Measurements”. https://www.keysight.com/us/en/assets/7018-06808/application-notes/5952-8255.pdf (accessed May 23, 2025).

[12] Guillaume Ferré, Audrey Giremus. LoRa Physical Layer Principle and Performance Analysis. ICECS 2018 25th IEEE International Conference on Electronics Circuits and Systems, Dec 2018, Bordeaux, France. ffhal-01977497f

[13] LoRa Alliance. “RP002-1.0.2 LoRaWAN® Regional Parameters”. https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf (accessed May 24, 2025).

[14] ENACOM. “Bandas de uso compartido sin autorización”. www.enacom.gob.ar/bandas-de-uso-compartido-sin-autorizacion_p680 (accessed May 24, 2025).

[15] Pixel Electric. ”STM32F103C8T6 Blue Pill Development Board”. https://www.pixelelectric.com/development-boards/stm32-boards/stm32f103c8t6-blue-pill-development-board/ (accessed Sep 21, 2025).

[16] STMicroelectronics. ”STM32F103x8, Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces”. https://www.st.com/resource/en/datasheet/stm32f103c8.pdf (accessed Sep 16, 2025).

[17] LEM International SA. ”Current Transducer LAH 100-P”. https://www.lem.com/sites/default/files/products_datasheets/lah_100-p.pdf (accessed Sep 20, 2025).

[18] Texas Instruments. ”Application Design Guidelines for LM324 and LM358 Devices”. https://www.ti.com/lit/an/sloa277b/sloa277b.pdf?ts=1733704785952 (accessed Sep 21, 2025).

[19] Ebyte. ”E220-900T22D LoRa Wireless UART Module”. https://www.cdebyte.com/pdf-down.aspx?id=3552 (accessed Oct. 5, 2025).

[20] RogerBit. ”Módulo Lora E22-900T22D hasta 5 Kilometros”. https://rogerbit.com/wprb/2024/02/modulo-e22-900t22d/ (accessed Oct. 5, 2025).

[21] Candy-Ho. ”Conversor USB a serial UART chip CP2102”. https://candy-ho.com/producto/conversor-usb-a-serial-uart-ttl-chip-cp2102-arduino-wifi-gps-2/ (accessed Oct. 11, 2025).

[22] AliExpress. 868MHz 915MHz Antenna 3dbi SMA Male Connector GSM GPRS. https://en.aliexpress.com/item/4001105458127.html (accessed Oct. 12, 2025).

[23] ESPBoards. ”DOIT ESP32 DEVKIT V1 Development Board”. https://www.espboards.dev/esp32/esp32doit-devkit-v1/ (accessed Oct. 13, 2025).

[24] Espressif Systems. ”ESP32-WROOM-32”. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf (accessed Oct. 13, 2025).

[25] Advanced Monolithic Systems. ”AMS1117, 1A Low Dropout Voltage Regulator”. http://www.advanced-monolithic.com/pdf/ds1117.pdf (accessed Oct. 14, 2025).

[26] Diodes Incorporated. ”AP2112, 600 mA CMOS LDO Regulator With Enable”. https://www.mouser.com/datasheet/2/115/AP2112-271550.pdf (accessed Oct. 14, 2025).

[27] JavanElec. ”Lithium Battery Charging & Protection Module Model:TP4056”. https://www.javanelec.com/stfiles/getappdocument/1/true/6e203fe4-c888-446f-a2ec-e0751e0fd167.pdf (accessed Oct. 14, 2025).

[28] Espressif Systems. ”Introduction to Low Power Mode for Systemic Power Management”. https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/low-power-mode/low-power-mode-soc.html (accessed Oct. 18, 2025).

[29] V. Levek, P. Šteffan, “Battery lifespan calculation and principles of design for low power mode”, Journal of Electrical Engineering, vol. 70, pp. 39–45, 2019, doi: 10.2478/jee-2019-0005.

[30] Firebase. ”Firebase | Google’s Mobile and Web App Development Platform”. https://firebase.google.com/ (accessed Oct. 19, 2025).

[31] Kodular. ”Introduction - Kodular Docs”. https://docs.kodular.io/ (accessed Oct. 20, 2025).

[32] Dielectric Argentina. ”Multímetro Digital UNI-T UT70D”. https://www.dielectricargentina.com.ar/de-mano/3270-3270.html (accessed Nov. 1, 2025).

[33] UNI-T. ”UT210 Series Mini Clamp Meters”. https://meters.uni-trend.com/product/ut210-series/ (accessed Nov. 1, 2025).

Anexo 1: Esquemáticos del circuito de medición y del receptor

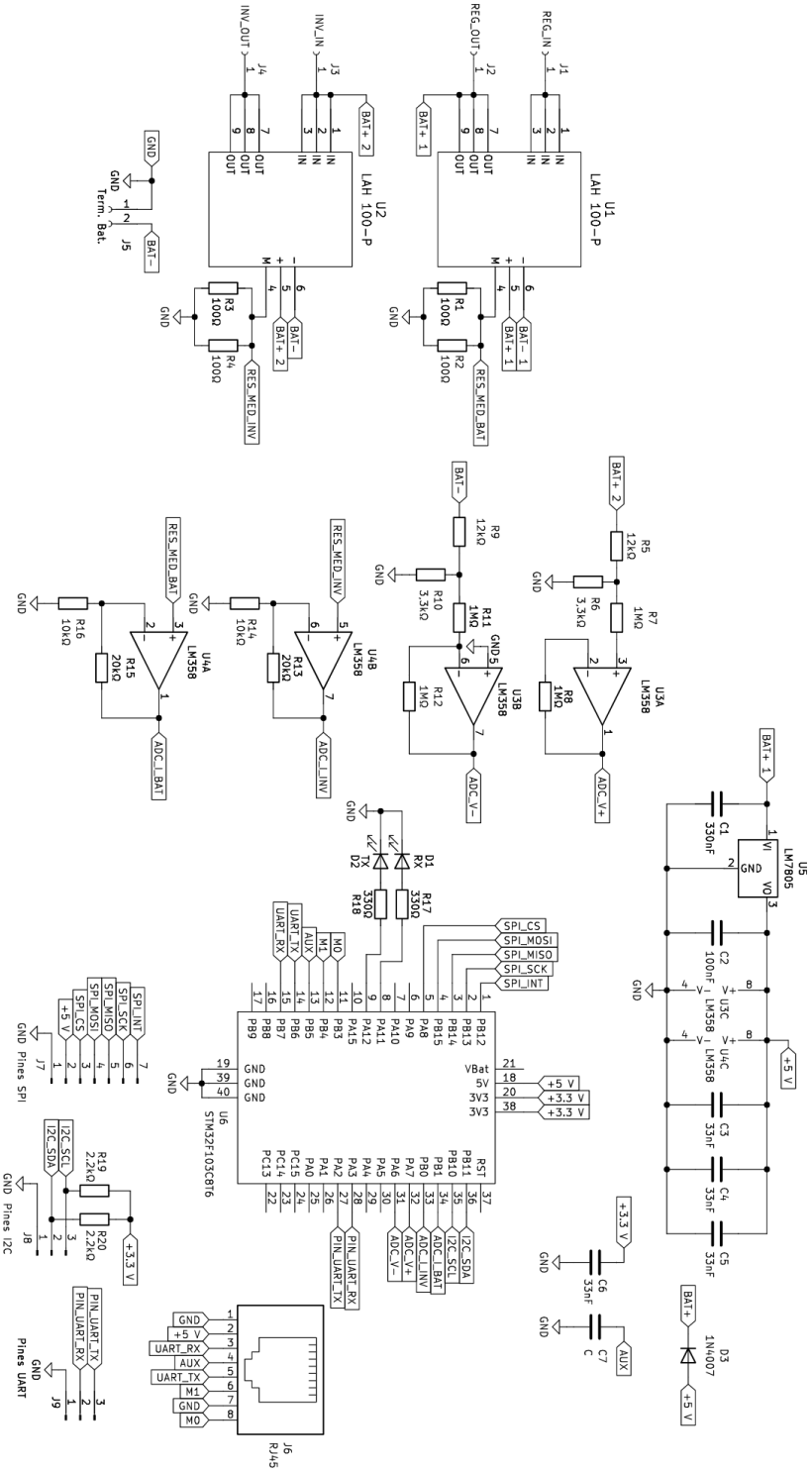


Figura N° 51: Esquemático de la PCB del sistema de mediciones.

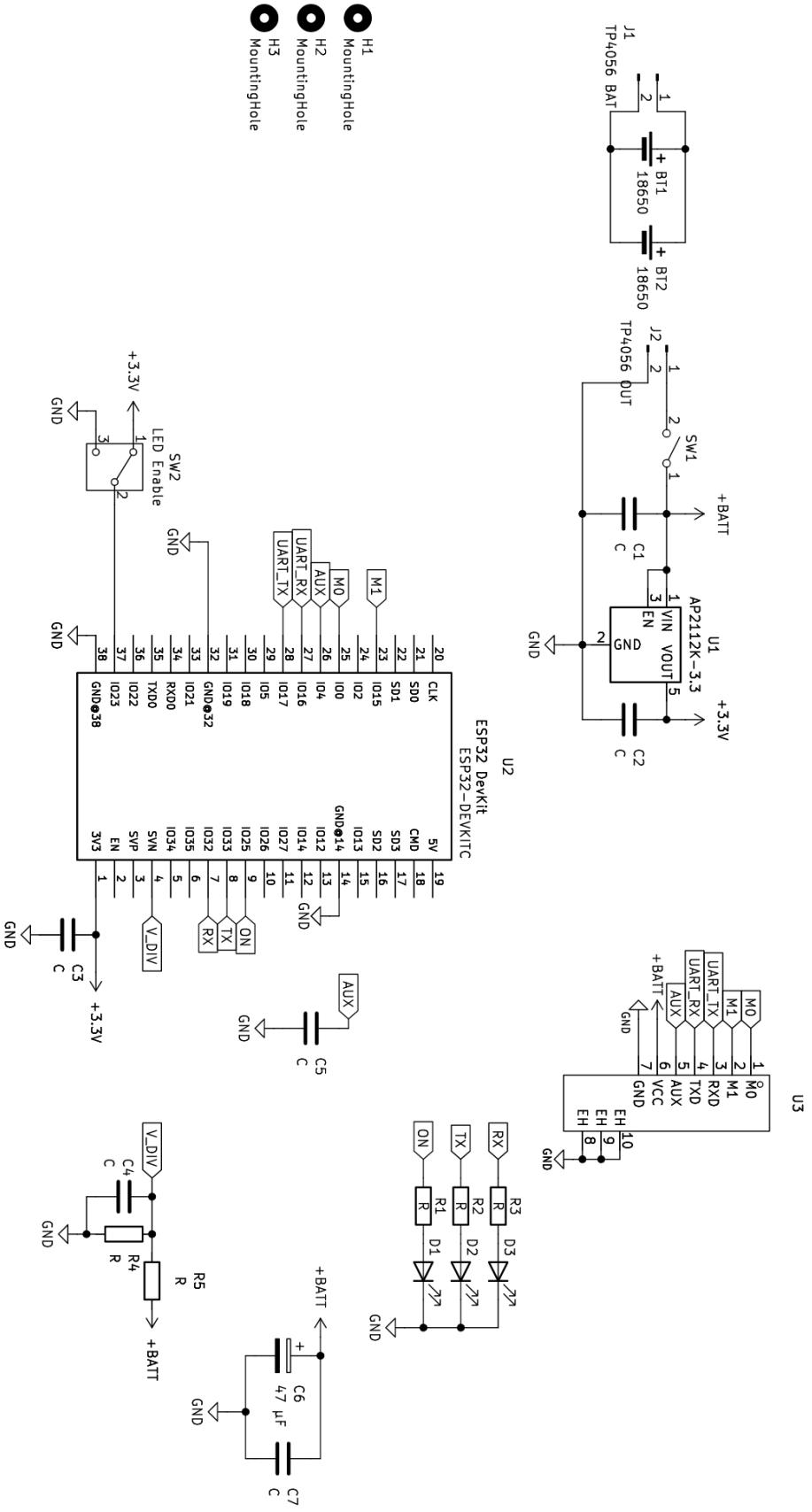


Figura N° 52: Esquemático de la PCB del receptor.

Anexo 2: Código fuente del microcontrolador STM32F103C8T6

```
1  /* USER CODE BEGIN Header */
2
3  /* USER CODE END Header */
4  /* Includes
       ------------------------------------------------------------------*/
5  #include "main.h"
6
7  /* Private includes
       ----------------------------------------------------------*/
8  /* USER CODE BEGIN Includes */
9  #include <stdbool.h>
10 #include <stdint.h>
11 #include <stdlib.h>
12 #include <time.h>
13 #include <string.h>
14 /* USER CODE END Includes */
15
16 /* Private typedef
       -----------------------------------------------------------*/
17 /* USER CODE BEGIN PTD */
18
19 /* USER CODE END PTD */
20
21 /* Private define
       ------------------------------------------------------------*/
22 /* USER CODE BEGIN PD */
23
24 /* USER CODE END PD */
25
26 /* Private macro
       -------------------------------------------------------------*/
27 /* USER CODE BEGIN PM */
28 #define V_REF_INT     1.2F  // Tensión de referencia interna
29 #define AVG_SLOPE     4.3F  // Pendiente que depende de la temperatura
30 #define V_SENSOR_25C  1.43F // Tensión a 25°C
31 /* USER CODE END PM */
32
33 /* Private variables
       ---------------------------------------------------------*/
34 ADC_HandleTypeDef hadc1;
35 DMA_HandleTypeDef hdma_adc1;
36
```

```
37 IWDG_HandleTypeDef hiwdg;
38
39 TIM_HandleTypeDef htim1;
40 TIM_HandleTypeDef htim2;
41 TIM_HandleTypeDef htim3;
42
43 UART_HandleTypeDef huart1;
44 DMA_HandleTypeDef hdma_usart1_rx;
45 DMA_HandleTypeDef hdma_usart1_tx;
46
47 /* USER CODE BEGIN PV */
48 bool fallo = 0;
49 volatile bool flagEmptyTX = false;
50
51 uint8_t bufferTXUART[16], emptyMessage[1] = {0};
52
53 volatile uint8_t checkAuxTX = 0;
54 volatile uint16_t resultadosADC[6], cuenta;
55 volatile uint32_t contador = 0;
56
57 uint32_t sumaTensionPos = 0, sumaTensionNeg = 0,
58 sumaCorrienteBat = 0, sumaCorrienteInv = 0;
59
60 float tensionBateria, tensionBateriaPos, tensionBateriaNeg,
61 corrienteBateria, corrienteInversor, temperaturaMicro, promedio,
62 tensionRef;
63
64 /* USER CODE END PV */
65
66 /* Private function prototypes
      -----------------------------------------------*/
67 void SystemClock_Config(void);
68 static void MX_GPIO_Init(void);
69 static void MX_DMA_Init(void);
70 static void MX_ADC1_Init(void);
71 static void MX_TIM3_Init(void);
72 static void MX_TIM1_Init(void);
73 static void MX_IWDG_Init(void);
74 static void MX_USART1_UART_Init(void);
75 static void MX_TIM2_Init(void);
76 /* USER CODE BEGIN PFP */
77
78 /* USER CODE END PFP */
79
80 /* Private user code
      ---------------------------------------------------------*/
81 /* USER CODE BEGIN 0 */
82
```

```
83  // Callback de los temporizadores
84  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
85    if (htim -> Instance == TIM1) {
86      // Se inicializa el temporizador 3
87      HAL_TIM_Base_Start(&htim3);
88    } else if (htim -> Instance == TIM2) {
89      if ((GPIOB -> IDR >> 5) & 0x01) {
90        HAL_TIM_Base_Stop_IT(&htim2);
91        HAL_GPIO_WritePin(GPIOA, LED_TX_Pin, 0);
92      }
93    }
94  }
95
96  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
97  {
98    cuenta++;
99    sumaTensionPos += resultadosADC[0];
100   sumaTensionNeg += resultadosADC[1];
101   sumaCorrienteBat += resultadosADC[2];
102   sumaCorrienteInv += resultadosADC[3];
103 }
104
105 // Rutina de callback para interrupciones externas
106 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
107 {
108   /* Se verifica que la interrupción vino del pin AUX
109   * y se comprueba que el módulo no está transmitiendo */
110   if (GPIO_Pin == GPIO_PIN_5 && !flagEmptyTX) {
111     HAL_GPIO_WritePin(GPIOA, LED_TX_Pin, 1);
112   }
113 }
114
115 // Rutina de callback transmisión UART completada
116 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
117   /* Se obtiene el estado en el pin de interrupción
118   * (si es 0 el módulo todavía está ocupado transmitiendo) */
119   if ((GPIOA -> IDR >> 2) & 0x01) {
120     /** Si el modulo todavia está recibiendo se
121     * verifica el estado del  pin cada 50 ms */
122     HAL_TIM_Base_Start_IT(&htim2);
123   }
124
125   /* USER CODE END 0 */
126
127   /**
128   * @brief  The application entry point.
129   * @retval int
130   */
```

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU
     Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* Se espera a que se estabilice la tensión antes
  * de configurar el reloj para evitar errores*/
  HAL_Delay(200);
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  MX_TIM1_Init();
  MX_IWDG_Init();
  MX_USART1_UART_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_Delay(10);
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)&resultadosADC, 6);
  HAL_TIM_Base_Start_IT(&htim1);
  // Configuración de M0 y M1
  HAL_GPIO_WritePin(GPIOB, M0_Pin, 1); // WOR Transmitter Mode
  HAL_GPIO_WritePin(GPIOB, M1_Pin, 0);

  // Se apaga el LED indicador de reinicio
  HAL_GPIO_WritePin(GPIOC, LED_Pin, 1);
  /* USER CODE END 2 */

```

```
177     /* Infinite loop */
178     /* USER CODE BEGIN WHILE */
179     while (1)
180     {
181       // Se verifica si ya se tomaron 100 muestras
182       if (cuenta >= 1000) {
183         // Se detiene la base de tiempo para el ADC
184         HAL_TIM_Base_Stop(&htim3);
185
186         // Tensión de alimentacion actual
187         tensionRef = V_REF_INT * 4095 / resultadosADC[4];
188         // Se promedian las mediciones y se hacen las conversiones
189         /* Para la tensión se hace el promedio, se convierte el valor
               leído a tensión
190         *  y se multiplica por los valores del divisor resistivo */
191         tensionBateriaPos = (sumaTensionPos / 1000.0) * (tensionRef /
               4095.0) * 377/81;
192         tensionBateriaNeg = (sumaTensionNeg / 1000.0) * (tensionRef /
               4095.0) * 1333/285;
193         tensionBateria = tensionBateriaPos + tensionBateriaNeg;
194
195         /* Para la corriente se hace el promedio, se convierte el valor
               leído a tensión,
196         * se divide por la ganancia del amplificador, se convierte la
               tensión a corriente
197         * según el resistor de medición elegido y se multiplica por la
               relación de
198         * transformación */
199         promedio = sumaCorrienteBat / 1000;
200         corrienteBateria = ((((sumaCorrienteBat / 1000.0)  * (tensionRef
               / 4095.0)) / 2.99) / 49.7) * 2000.0;
201         // Compensación y consideración de corrientes negativas por el
               ruido
202         /*if (promedio < 5) {
203           corrienteBateria = 0.0;
204         } else if (promedio < 1) {
205           corrienteBateria -= 0.1;
206         } else {
207           corrienteBateria += 0.15;
208         }*/
209         corrienteInversor = 0;
210         // Si por el ruido la medición resulta negativa se hace I = 0
211         corrienteBateria = corrienteBateria < 0 ? 0.0 : corrienteBateria;
212         corrienteInversor = corrienteInversor < 0 ? 0.0 :
               corrienteInversor;
213         // Lectura de temperatura
214         float lecturaTemperatura = resultadosADC[5] * tensionRef / 4095.0;
```

```
215         temperaturaMicro = ((V_SENSOR_25C - lecturaTemperatura) * 1000.0)
                / AVG_SLOPE + 25.0;
216         // Reinicio de variables y contador
217         cuenta = 0;
218         sumaTensionNeg = 0;
219         sumaTensionPos = 0;
220         sumaCorrienteBat = 0;
221         sumaCorrienteInv = 0;
222
223
224         /* Se copian los datos de las variables electricas a un
225         * buffer para la transmisión*/
226
227         memcpy(&bufferTXUART[0], &tensionBateria, sizeof(float));
228         memcpy(&bufferTXUART[4], &corrienteBateria, sizeof(float));
229         memcpy(&bufferTXUART[8], &corrienteInversor, sizeof(float));
230         memcpy(&bufferTXUART[12], &temperaturaMicro, sizeof(float));
231
232         flagEmptyTX = true;
233         HAL_UART_Transmit_DMA(&huart1, emptyMessage,
                sizeof(emptyMessage));
234         flagEmptyTX = false;
235         HAL_Delay(1050);
236
237         HAL_UART_Transmit_DMA(&huart1, bufferTXUART,
                sizeof(bufferTXUART));
238       }
239       HAL_IWDG_Refresh(&hiwdg);
240       /* USER CODE END WHILE */
241
242       /* USER CODE BEGIN 3 */
243     }
244     /* USER CODE END 3 */
245   }
246
247   /**
248   * @brief System Clock Configuration
249   * @retval None
250   */
251   void SystemClock_Config(void)
252   {
253     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
254     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
255     RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
256
257     /** Initializes the RCC Oscillators according to the specified
            parameters
258     * in the RCC_OscInitTypeDef structure.
```

```
259       */
260       RCC_OscInitStruct.OscillatorType =
              RCC_OSCILLATORTYPE_LSI|RCC_OSCILLATORTYPE_HSE;
261       RCC_OscInitStruct.HSEState = RCC_HSE_ON;
262       RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
263       RCC_OscInitStruct.HSIState = RCC_HSI_ON;
264       RCC_OscInitStruct.LSIState = RCC_LSI_ON;
265       RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
266       RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
267       RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
268       if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
269       {
270         Error_Handler();
271       }
272
273       /** Initializes the CPU, AHB and APB buses clocks
274       */
275       RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
276       |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
277       RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
278       RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
279       RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
280       RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
281
282       if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
              HAL_OK)
283       {
284         Error_Handler();
285       }
286       PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
287       PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
288       if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
289       {
290         Error_Handler();
291       }
292     }
293
294     /**
295     * @brief ADC1 Initialization Function
296     * @param None
297     * @retval None
298     */
299     static void MX_ADC1_Init(void)
300     {
301
302       /* USER CODE BEGIN ADC1_Init 0 */
303
304       /* USER CODE END ADC1_Init 0 */
```

```
305
306     ADC_ChannelConfTypeDef sConfig = {0};
307
308     /* USER CODE BEGIN ADC1_Init 1 */
309
310     /* USER CODE END ADC1_Init 1 */
311
312     /** Common config
313     */
314     hadc1.Instance = ADC1;
315     hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
316     hadc1.Init.ContinuousConvMode = DISABLE;
317     hadc1.Init.DiscontinuousConvMode = DISABLE;
318     hadc1.Init.ExternalTrigConv = ADC_EXTERNALTRIGCONV_T3_TRGO;
319     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
320     hadc1.Init.NbrOfConversion = 6;
321     if (HAL_ADC_Init(&hadc1) != HAL_OK)
322     {
323       Error_Handler();
324     }
325
326     /** Configure Regular Channel
327     */
328     sConfig.Channel = ADC_CHANNEL_9;
329     sConfig.Rank = ADC_REGULAR_RANK_1;
330     sConfig.SamplingTime = ADC_SAMPLETIME_7CYCLES_5;
331     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
332     {
333       Error_Handler();
334     }
335
336     /** Configure Regular Channel
337     */
338     sConfig.Channel = ADC_CHANNEL_8;
339     sConfig.Rank = ADC_REGULAR_RANK_2;
340     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
341     {
342       Error_Handler();
343     }
344
345     /** Configure Regular Channel
346     */
347     sConfig.Channel = ADC_CHANNEL_7;
348     sConfig.Rank = ADC_REGULAR_RANK_3;
349     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
350     {
351       Error_Handler();
352     }
```

```
353
354       /** Configure Regular Channel
355       */
356       sConfig.Channel = ADC_CHANNEL_6;
357       sConfig.Rank = ADC_REGULAR_RANK_4;
358       if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
359       {
360         Error_Handler();
361       }
362
363       /** Configure Regular Channel
364       */
365       sConfig.Channel = ADC_CHANNEL_VREFINT;
366       sConfig.Rank = ADC_REGULAR_RANK_5;
367       sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
368       if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
369       {
370         Error_Handler();
371       }
372
373       /** Configure Regular Channel
374       */
375       sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
376       sConfig.Rank = ADC_REGULAR_RANK_6;
377       if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
378       {
379         Error_Handler();
380       }
381       /* USER CODE BEGIN ADC1_Init 2 */
382
383       /* USER CODE END ADC1_Init 2 */
384
385     }
386
387     /**
388     * @brief IWDG Initialization Function
389     * @param None
390     * @retval None
391     */
392     static void MX_IWDG_Init(void)
393     {
394
395       /* USER CODE BEGIN IWDG_Init 0 */
396
397       /* USER CODE END IWDG_Init 0 */
398
399       /* USER CODE BEGIN IWDG_Init 1 */
400
```

```
401     /* USER CODE END IWDG_Init 1 */
402     hiwdg.Instance = IWDG;
403     hiwdg.Init.Prescaler = IWDG_PRESCALER_32;
404     hiwdg.Init.Reload = 3000;
405     if (HAL_IWDG_Init(&hiwdg) != HAL_OK)
406     {
407       Error_Handler();
408     }
409     /* USER CODE BEGIN IWDG_Init 2 */
410
411     /* USER CODE END IWDG_Init 2 */
412
413   }
414
415   /**
416   * @brief TIM1 Initialization Function
417   * @param None
418   * @retval None
419   */
420   static void MX_TIM1_Init(void)
421   {
422
423     /* USER CODE BEGIN TIM1_Init 0 */
424
425     /* USER CODE END TIM1_Init 0 */
426
427     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
428     TIM_MasterConfigTypeDef sMasterConfig = {0};
429
430     /* USER CODE BEGIN TIM1_Init 1 */
431
432     /* USER CODE END TIM1_Init 1 */
433     htim1.Instance = TIM1;
434     htim1.Init.Prescaler = 4895;
435     htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
436     htim1.Init.Period = 65358;
437     htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
438     htim1.Init.RepetitionCounter = 0;
439     htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
440     if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
441     {
442       Error_Handler();
443     }
444     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
445     if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
446     {
447       Error_Handler();
448     }
```

```
449     sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
450     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
451     if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
            HAL_OK)
452     {
453       Error_Handler();
454     }
455     /* USER CODE BEGIN TIM1_Init 2 */
456
457     /* USER CODE END TIM1_Init 2 */
458
459   }
460
461   /**
462   * @brief TIM2 Initialization Function
463   * @param None
464   * @retval None
465   */
466   static void MX_TIM2_Init(void)
467   {
468
469     /* USER CODE BEGIN TIM2_Init 0 */
470
471     /* USER CODE END TIM2_Init 0 */
472
473     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
474     TIM_MasterConfigTypeDef sMasterConfig = {0};
475
476     /* USER CODE BEGIN TIM2_Init 1 */
477
478     /* USER CODE END TIM2_Init 1 */
479     htim2.Instance = TIM2;
480     htim2.Init.Prescaler = 50;
481     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
482     htim2.Init.Period = 49999;
483     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
484     htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
485     if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
486     {
487       Error_Handler();
488     }
489     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
490     if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
491     {
492       Error_Handler();
493     }
494     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
495     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
496     if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
            HAL_OK)
497     {
498       Error_Handler();
499     }
500     /* USER CODE BEGIN TIM2_Init 2 */
501
502     /* USER CODE END TIM2_Init 2 */
503
504   }
505
506   /**
507   * @brief TIM3 Initialization Function
508   * @param None
509   * @retval None
510   */
511   static void MX_TIM3_Init(void)
512   {
513
514     /* USER CODE BEGIN TIM3_Init 0 */
515
516     /* USER CODE END TIM3_Init 0 */
517
518     TIM_ClockConfigTypeDef sClockSourceConfig = {0};
519     TIM_MasterConfigTypeDef sMasterConfig = {0};
520
521     /* USER CODE BEGIN TIM3_Init 1 */
522
523     /* USER CODE END TIM3_Init 1 */
524     htim3.Instance = TIM3;
525     htim3.Init.Prescaler = 0;
526     htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
527     htim3.Init.Period = 399;
528     htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
529     htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
530     if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
531     {
532       Error_Handler();
533     }
534     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
535     if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
536     {
537       Error_Handler();
538     }
539     sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
540     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
541     if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) !=
            HAL_OK)
```

```
542     {
543       Error_Handler();
544     }
545     /* USER CODE BEGIN TIM3_Init 2 */
546
547     /* USER CODE END TIM3_Init 2 */
548
549   }
550
551   /**
552   * @brief USART1 Initialization Function
553   * @param None
554   * @retval None
555   */
556   static void MX_USART1_UART_Init(void)
557   {
558
559     /* USER CODE BEGIN USART1_Init 0 */
560
561     /* USER CODE END USART1_Init 0 */
562
563     /* USER CODE BEGIN USART1_Init 1 */
564
565     /* USER CODE END USART1_Init 1 */
566     huart1.Instance = USART1;
567     huart1.Init.BaudRate = 9600;
568     huart1.Init.WordLength = UART_WORDLENGTH_8B;
569     huart1.Init.StopBits = UART_STOPBITS_1;
570     huart1.Init.Parity = UART_PARITY_NONE;
571     huart1.Init.Mode = UART_MODE_TX_RX;
572     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
573     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
574     if (HAL_UART_Init(&huart1) != HAL_OK)
575     {
576       Error_Handler();
577     }
578     /* USER CODE BEGIN USART1_Init 2 */
579
580     /* USER CODE END USART1_Init 2 */
581
582   }
583
584   /**
585   * Enable DMA controller clock
586   */
587   static void MX_DMA_Init(void)
588   {
589
```

```
590     /* DMA controller clock enable */
591     __HAL_RCC_DMA1_CLK_ENABLE();
592
593     /* DMA interrupt init */
594     /* DMA1_Channel1_IRQn interrupt configuration */
595     HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 4, 0);
596     HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
597     /* DMA1_Channel4_IRQn interrupt configuration */
598     HAL_NVIC_SetPriority(DMA1_Channel4_IRQn, 0, 0);
599     HAL_NVIC_EnableIRQ(DMA1_Channel4_IRQn);
600     /* DMA1_Channel5_IRQn interrupt configuration */
601     HAL_NVIC_SetPriority(DMA1_Channel5_IRQn, 0, 0);
602     HAL_NVIC_EnableIRQ(DMA1_Channel5_IRQn);
603
604   }
605
606   /**
607   * @brief GPIO Initialization Function
608   * @param None
609   * @retval None
610   */
611   static void MX_GPIO_Init(void)
612   {
613     GPIO_InitTypeDef GPIO_InitStruct = {0};
614     /* USER CODE BEGIN MX_GPIO_Init_1 */
615     /* USER CODE END MX_GPIO_Init_1 */
616
617     /* GPIO Ports Clock Enable */
618     __HAL_RCC_GPIOC_CLK_ENABLE();
619     __HAL_RCC_GPIOD_CLK_ENABLE();
620     __HAL_RCC_GPIOA_CLK_ENABLE();
621     __HAL_RCC_GPIOB_CLK_ENABLE();
622
623     /*Configure GPIO pin Output Level */
624     HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);
625
626     /*Configure GPIO pin Output Level */
627     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14|M0_Pin|M1_Pin, GPIO_PIN_RESET);
628
629     /*Configure GPIO pin Output Level */
630     HAL_GPIO_WritePin(GPIOA, LED_RX_Pin|LED_TX_Pin, GPIO_PIN_RESET);
631
632     /*Configure GPIO pin : LED_Pin */
633     GPIO_InitStruct.Pin = LED_Pin;
634     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
635     GPIO_InitStruct.Pull = GPIO_NOPULL;
636     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
637     HAL_GPIO_Init(LED_GPIO_Port, &GPIO_InitStruct);
```

```
638
639     /*Configure GPIO pins : PB14 M0_Pin M1_Pin */
640     GPIO_InitStruct.Pin = GPIO_PIN_14|M0_Pin|M1_Pin;
641     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
642     GPIO_InitStruct.Pull = GPIO_NOPULL;
643     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
644     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
645
646     /*Configure GPIO pins : LED_RX_Pin LED_TX_Pin */
647     GPIO_InitStruct.Pin = LED_RX_Pin|LED_TX_Pin;
648     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
649     GPIO_InitStruct.Pull = GPIO_NOPULL;
650     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
651     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
652
653     /*Configure GPIO pin : AUX_Pin */
654     GPIO_InitStruct.Pin = AUX_Pin;
655     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
656     GPIO_InitStruct.Pull = GPIO_NOPULL;
657     HAL_GPIO_Init(AUX_GPIO_Port, &GPIO_InitStruct);
658
659     /* EXTI interrupt init*/
660     HAL_NVIC_SetPriority(EXTI9_5_IRQn, 1, 0);
661     HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
662
663     /* USER CODE BEGIN MX_GPIO_Init_2 */
664     /* USER CODE END MX_GPIO_Init_2 */
665   }
666
667   /* USER CODE BEGIN 4 */
668
669   /* USER CODE END 4 */
670
671   /**
672   * @brief  This function is executed in case of error occurrence.
673   * @retval None
674   */
675   void Error_Handler(void)
676   {
677     /* USER CODE BEGIN Error_Handler_Debug */
678     /* User can add his own implementation to report the HAL error return
          state */
679     __disable_irq();
680     NVIC_SystemReset();
681     /* USER CODE END Error_Handler_Debug */
682   }
683
684   #ifdef  USE_FULL_ASSERT
```

```
685   /**
686   * @brief  Reports the name of the source file and the source line number
687   *         where the assert_param error has occurred.
688   * @param  file: pointer to the source file name
689   * @param  line: assert_param error line source number
690   * @retval None
691   */
692   void assert_failed(uint8_t *file, uint32_t line)
693   {
694     /* USER CODE BEGIN 6 */
695     /* User can add his own implementation to report the file name and
          line number,
696     ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
          line) */
697     /* USER CODE END 6 */
698   }
699   #endif /* USE_FULL_ASSERT */
```

Anexo 3: Código fuente del microcontrolador ESP32-D0WDQ6

```
#define ENABLE_DATABASE
#define ENABLE_USER_AUTH

#include <esp32-hal.h>
#include "driver/rtc_io.h"
#include "firebase_root_ca.h"
#include <time.h>
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <FirebaseClient.h>

// Prototipos de funciones
void pinConfig(void);
void wakeUpManagement(void);
void connectToWiFi(const char* ssid, const char* password, uint32_t
    timeout = 10000);
void syncTime(const char* timezone, uint32_t timeout = 10000);
void initFirebaseRTDB(const char* url, uint32_t timeout = 10000, uint32_t
    hs_timeout = 5);
void processData(AsyncResult& aresult);
uint8_t measureAmbientNoise(uint32_t timeout = 5000);
float measureaBatteries(void);
void readReceivedData(float& battV, float& battI, float& invI, float&
    tempSTM, uint8_t& rssi);
void sendDataToFirebase(uint32_t &timestamp, float &battV, float &battI,
    float &invI, float &tempSTM, float &tempESP, float& rxBatt, uint8_t
    noise, uint8_t &rssi);

#define WIFI_SSID "WiFi Network"
#define WIFI_PASSWORD "WiFi Password"
#define FIREBASE_URL "RTDB URL"
#define FIREBASE_API_KEY "Api Key"
#define USER_EMAIL "x"
#define USER_PASSWORD "x"

#define LED_POWER 25
#define LED_RX 32
#define LED_TX 33
#define SWITCH 23
#define BATT_MEAS 39
#define M0 GPIO_NUM_0
```

```
38 #define M1 GPIO_NUM_15
39 #define AUX GPIO_NUM_4
40 #define TXD 17
41 #define RXD 16
42
43 bool switchState;
44 const uint8_t ambientNoiseCommand[6] = {0xC0, 0xC1, 0xC2, 0xC3, 0x00,
       0x01};
45 uint8_t RSSIESP;
46 uint32_t timestamp;
47 RTC_DATA_ATTR int wakeCounter = 0;
48 float batteryVoltage, batteryCurrent, inverterCurrent,
49 rxBattery = 0, temperatureSTM, temperatureESP = 0;
50 const char* timezone = "<-03>3";  // Zona horaria Argentina
51 struct tm timeinfo; // Estructura con la información del tiempo actual
52
53 HardwareSerial LoRaModule(2);  // Alias puerto serial 2
54
55 UserAuth user_auth(FIREBASE_API_KEY, USER_EMAIL, USER_PASSWORD); // Clase
       que gestiona las credenciales
56 FirebaseApp app; // Instancia que gestiona todas las tareas
57 WiFiClientSecure ssl_client; // Instancia de un cliente SSL
58 using AsyncClient = AsyncClientClass; // Alias para la clase
       AsyncClientClass
59 AsyncClient async_client(ssl_client); // Instancia de cliente para
       operaciones asincrónicas
60 AsyncResult dbResult;
61 RealtimeDatabase Database; // Instancia de la clase RTDB para gestionar
       las DB
62
63 void setup() {
64   Serial.begin(115200);
65   LoRaModule.begin(9600, SERIAL_8N1, RXD, TXD);
66
67   pinConfig(); // Configuracion de GPIOs
68   switchState = digitalRead(SWITCH);
69   digitalWrite(LED_POWER, switchState);
70
71   esp_sleep_enable_ext0_wakeup(AUX, 0); // Despierta con flanco
       descendente
72   wakeUpManagement(); // Comportamiento al salir del modo de bajo consumo
73
74   connectToWiFi(WIFI_SSID, WIFI_PASSWORD); // Conexión a la red WiFi
75   syncTime(timezone); // Obtiene la hora actual
76
77   initFirebaseRTDB(FIREBASE_URL); // Conexión con la RTDB y configuración
       SSL
78 }
```

```
79
80  void loop() {
81    uint32_t timeoutLoop = millis();
82    Serial.println("Waiting for serial data...");
83    app.loop();
84    while(millis() - timeoutLoop < 5000) {
85      app.loop();
86      if(LoRaModule.available()){
87        // Recepción de los datos enviados por LoRa
88        readReceivedData(batteryVoltage, batteryCurrent, inverterCurrent,
89        temperatureSTM, RSSIESP);
90        uint8_t ambientNoise = measureAmbientNoise();
91        temperatureESP = temperatureRead();
92        rxBattery = measureaBatteries();
93
94        /*************************************************************************/
95        Serial.print("Battery voltage: "); Serial.println(batteryVoltage,
              2);
96        Serial.print("Battery current: "); Serial.println(batteryCurrent,
              2);
97        Serial.print("Inverter current: "); Serial.println(inverterCurrent,
              2);
98        Serial.print("STM32 temperature: "); Serial.println(temperatureSTM,
              2);
99        Serial.print("ESP32 temperature: "); Serial.println(temperatureESP,
              2);
100       Serial.print("RX battery voltage: "); Serial.println(rxBattery, 2);
101       Serial.print("Receiver ambient noise: ");
              Serial.println(ambientNoise);
102       Serial.print("RSSI ESP32: "); Serial.println(RSSIESP);
103       /*************************************************************************/
104
105       // Envio de datos a Firebase
106       sendDataToFirebase(timestamp, batteryVoltage, batteryCurrent,
              inverterCurrent,
107       temperatureSTM, temperatureESP, rxBattery, ambientNoise, RSSIESP);
108
109       // Fin del programa, se entra a modo de bajo consumo
110       Serial.println(); Serial.println("Entering deep sleep...");
111       esp_deep_sleep_start();
112     }
113   }
114   Serial.println("ERROR: Failed to get data trough serial port");
115   Serial.println("Entering deep sleep...");
116   esp_deep_sleep_start();
117 }
118
119 void pinConfig() {
```

```
120   /* Se usan pines GPIO del módulo RTC para establecer el modo de
          funcionamiento porque son los únicos activos
121   durante el modo deep sleep */
122   rtc_gpio_init(M0);
123   rtc_gpio_init(M1);
124   rtc_gpio_set_direction(M0, RTC_GPIO_MODE_OUTPUT_ONLY);
125   rtc_gpio_set_direction(M1, RTC_GPIO_MODE_OUTPUT_ONLY);
126   rtc_gpio_set_level(M0, 0);
127   rtc_gpio_set_level(M1, 1);
128   rtc_gpio_hold_en(M0);
129   rtc_gpio_hold_en(M1);
130
131   pinMode(AUX, INPUT);
132   pinMode(SWITCH, INPUT);
133   pinMode(LED_TX, OUTPUT);
134   pinMode(LED_RX, OUTPUT);
135   pinMode(LED_POWER, OUTPUT);
136 };
137
138 void wakeUpManagement() {
139   /* Si el  C despertó por una causa distinta a un flanco
140   descendente este vuelve a dormir */
141   esp_sleep_wakeup_cause_t wakeup_reason = esp_sleep_get_wakeup_cause();
142   Serial.println();
143   if (wakeup_reason == ESP_SLEEP_WAKEUP_EXT0) {
144     wakeCounter++;
145     Serial.println("Wakeup reason: EXTI0 - AUX");
146     Serial.print("Wakeup counter: ");
147     Serial.println(wakeCounter);
148   } else {
149     Serial.println("Wakeup reason: another than EXTI0");
150     Serial.println("Entering deep sleep...");
151     esp_deep_sleep_start();
152   }
153 };
154
155 void connectToWiFi(const char* ssid, const char* password, uint32_t
       timeout) {
156   WiFi.mode(WIFI_STA); // El ESP32 actua como cliente
157   WiFi.begin(ssid, password);
158   //WiFi.begin(ssid, password); Redes con contraseña
159   Serial.println(); Serial.printf("Connecting to WiFi network
          \"%s\"...\n", ssid);
160
161   uint32_t startTime = millis();
162   while (WiFi.status() != WL_CONNECTED && millis() - startTime < timeout)
          {
163     delay(500);
```

```
164       Serial.print(".");
165     }
166
167     if (WiFi.status() == WL_CONNECTED) {
168       Serial.println();
169       Serial.print("Connected with IP: ");
170       Serial.println(WiFi.localIP());
171     } else {
172       Serial.println();
173       Serial.println("ERROR: Wi-Fi connection timeout");
174       Serial.println("Entering deep sleep...");
175       esp_deep_sleep_start();
176     }
177   };
178
179   void syncTime(const char* timezone, uint32_t timeout) {
180     // Diferentes opciones de servidores para consultar la hora
181     configTzTime(timezone,"ar.pool.ntp.org", "south-america.pool.ntp.org",
          "pool.ntp.org");
182     Serial.print("Waiting for NTP sync...");
183
184     uint32_t startTime = millis();
185     while (!getLocalTime(&timeinfo) && millis() - startTime < timeout) {
186       Serial.print(".");
187       delay(500);
188     }
189
190     if (getLocalTime(&timeinfo)) {
191       char timeString[64];
192       strftime(timeString, sizeof(timeString), "%Y-%m-%d %H:%M:%S",
            &timeinfo);
193       Serial.println(); Serial.print("Date and time (UTC-3): ");
194       Serial.println(timeString);
195       // Timestamp para clave en la RTDB
196       timestamp = time(nullptr);
197     } else {
198       Serial.println();
199       Serial.println("ERROR: NTP time sync timeout");
200       Serial.println("Entering deep sleep...");
201       esp_deep_sleep_start();
202     }
203   };
204
205   uint8_t measureAmbientNoise(uint32_t timeout) {
206     // Cambio de modo para poder transmitir al módulo
207     rtc_gpio_hold_dis(M0);
208     rtc_gpio_hold_dis(M1);
209     rtc_gpio_set_level(M0, 0);
```

```
210   rtc_gpio_set_level(M1, 0);
211   digitalWrite(LED_TX, switchState);
212
213   delay(50);
214   LoRaModule.write(ambientNoiseCommand, sizeof(ambientNoiseCommand));
215   Serial.println("Measuring ambient noise...");
216
217   uint32_t startTime = millis(), endTime = 0;
218   while(!LoRaModule.available() && endTime < 5000){
219     endTime = millis() - startTime;
220   }
221
222   // Cambio de función del módulo para la siguiente recepción
223   digitalWrite(LED_TX, 0);
224   rtc_gpio_set_level(M0, 0);
225   rtc_gpio_set_level(M1, 1);
226   rtc_gpio_hold_en(M0);
227   rtc_gpio_hold_en(M1);
228
229   if (endTime < 5000) {
230     digitalWrite(LED_RX, switchState);
231     uint8_t ambientNoiseESP[4];
232     LoRaModule.readBytes(ambientNoiseESP, sizeof(ambientNoiseESP));
233     return ambientNoiseESP[3];
234   } else {
235     Serial.println("ERROR: Failed to measure receiver ambient noise");
236     return 0;
237   }
238 };
239
240 float measureaBatteries() {
241   analogReadResolution(12);
242   analogSetPinAttenuation((gpio_num_t)BATT_MEAS, ADC_11db);
243
244   uint32_t sum = 0;
245   for (int i = 0; i < 100; i++) {
246     sum += analogRead(BATT_MEAS);
247   }
248
249   float batteriesVoltage = ((sum / 100.0f) / 4095) * 3.3 * 0.9717 *
        ((1.01 + 0.329) / 1.01);
250   return batteriesVoltage;
251 }
252
253 void readReceivedData(float &battV, float &battI, float &invI,
254 float &tempSTM, uint8_t &rssi) {
255   uint8_t bufferRX[17];  // Buffer para almacenar los 17 bytes recibidos
256   digitalWrite(LED_RX, switchState);
```

```
257   Serial.println("Receiving data...");
258   LoRaModule.readBytes(bufferRX, sizeof(bufferRX));
259
260   memcpy(&battV, bufferRX, sizeof(float));
261   memcpy(&battI, &bufferRX[4], sizeof(float));
262   memcpy(&invI, &bufferRX[8], sizeof(float));
263   memcpy(&tempSTM, &bufferRX[12], sizeof(float));
264   memcpy(&rssi, &bufferRX[16], sizeof(uint8_t));
265   digitalWrite(LED_RX, 0);
266 };
267
268 // Callback tareas finalizadas
269 void processData(AsyncResult &aResult)
270 {
271   if (!aResult.isResult())
272   return;
273
274   if (aResult.isError())
275   {
276     Serial.printf("[ERROR] Task: %s, Code: %d, Message: %s\n",
277     aResult.uid().c_str(),
278     aResult.error().code(),
279     aResult.error().message().c_str());
280   }
281   else if (aResult.available())
282   {
283     Serial.printf("[SUCCESS] Task: %s\n",
284     aResult.uid().c_str());
285   }
286 }
287
288 void initFirebaseRTDB(const char* url, uint32_t timeout, uint32_t
      hs_timeout) {
289   ssl_client.setCACert(FIREBASE_ROOT_CA); // Verificación del certificado
        SSL
290   ssl_client.setTimeout(timeout);
291   ssl_client.setHandshakeTimeout(hs_timeout);
292
293   Serial.println("Initializing Firebase RTDB..."); Serial.println();
294   initializeApp(async_client, app, getAuth(user_auth), nullptr, "Firebase
        Authentication");
295   app.getApp<RealtimeDatabase>(Database);
296   Database.url(FIREBASE_URL);
297 };
298
299 void sendDataToFirebase(uint32_t &timestamp, float &battV, float &battI,
      float &invI, float &tempSTM,
300 float &tempESP, float &rxBatt, uint8_t noise, uint8_t &rssi) {
```

```
301
302   // Se crea un objeto JSON con toda la información
303   JsonWriter writer;
304   object_t ts, bV, bC, iC, tS, tE, rxB, R, aN, dataJson;
305
306   writer.create(ts, "timestamp", timestamp);
307   writer.create(bV, "battVoltage", battV);
308   writer.create(bC, "battCurrent", battI);
309   writer.create(iC, "invCurrent", invI);
310   writer.create(tS, "temperatureSTM", tempSTM);
311   writer.create(tE, "temperatureESP", tempESP);
312   writer.create(rxB, "rxBattery", rxBatt);
313   writer.create(R, "RSSI", rssi);
314   writer.create(aN, "noise", noise);
315   writer.join(dataJson, 9, ts, bV, bC, iC, tS, tE, rxB, R, aN);
316
317   Serial.println(); Serial.println("Sending data to Firebase...");
318   app.loop();  //  Procesa eventos antes del envío
319   uint32_t startTime = millis(), endTime = 0;
320   while(!app.ready() && endTime < 5000){
321     app.loop();
322     endTime = millis() - startTime;
323   }
324
325   if (app.ready()) {
326     Database.push<object_t>(async_client, "data", dataJson, processData,
          "Firebase Data Upload");
327     Database.set<uint32_t>(async_client, "latestData", timestamp,
          processData, "Firebase Latest Data Update");
328     // Esperar a que se ejecute el callback
329     uint32_t t0 = millis();
330     while (millis() - t0 < 1000) app.loop();
331   } else {
332     Serial.println("ERROR: Firebase not ready, could not send data");
333   }
334 };
```