Universidad Zaragoza
1542

Trabajo Fin de Grado

Dispositivo IoT para la monitorización de parámetros fisiológicos

IoT device for monitoring physiological parameters

Autor

Elioenay Pérez López

Director

Dr. David Asiain Ansorena

Escuela Universitaria Politécnica La Almunia

Junio 2022

Página intencionadamente en blanco.

Escuela Universitaria Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

MEMORIA

Dispositivo IoT para la monitorización de parámetros fisiológicos

IoT device for monitoring physiological parameters

424.21.19

Autor: Elioenay Pérez López

Director: Dr. David Asiain Ansorena

Fecha: 06 2022

Página intencionadamente en blanco.

INDICE DE CONTENIDO BREVE

INDICE DE CONTENIDO

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1. RESUMEN

El Internet de las cosas (IoT) cobra cada día un papel más importante en nuestra sociedad. El número de dispositivos conectados a Internet crece exponencialmente, mientras que se desarrollan tecnologías que permiten esta escalabilidad. Al mismo tiempo, el tamaño y consumo energético de los dispositivos portátiles o ponibles (wearables) se han visto reducidos.

En este documento se crea un prototipo de dispositivo wearable que hace uso de tecnologías IoT para la transmisión de parámetros fisiológicos del cuerpo humano. Concretamente se hace uso de la tecnología LoRaWAN (Long Range Wide Area Network), desarrollando un prototipo capaz de medir la temperatura corporal y transmitirla mediante la red LoRaWAN hasta visualizarla en la interfaz del cliente final. Se desarrollan todos los niveles de la arquitectura LoRaWAN, adaptándolos a las necesidades del prototipo y permitiendo el acceso, a través de internet, a los datos de temperatura medidos.

Se emplea un enfoque con un nuevo tipo de arquitectura de dispositivo LoRaWAN, formado por un único microcontrolador de la familia STM32WL, estos son los primeros microcontroladores en contar con un transceptor LoRa integrado. También se emplean las herramientas de desarrollo del entorno STM32Cube. Todo esto permite simplificar considerablemente el diseño del hardware y software y ayuda a obtener mejores resultados de consumo.

Los resultados de consumo se han ajustado correctamente a los resultados obtenidos mediante las simulaciones, sobre todo en el modo de bajo consumo del microcontrolador. Se ha visto que este es el modo que más afecta al consumo en una aplicación real al ser el que más tiempo consume.

1.1. PALABRAS CLAVE

LoRa y LoRaWAN, STM32WL, The Things Network, dispositivo ponible, Internet de las cosas (IoT).

2. ABSTRACT

The Internet of Things (IoT) plays a more important role in our society every day. The number of devices connected to the Internet is growing exponentially, while technologies that allow this scalability are being developed. Inversely, the size and energy consumption of portable or wearable devices are in constant reduction.

In this document, a wearable device prototype is created that makes use of IoT technologies for the transmission of physiological parameters of the human body. The specific implementation of a LoRaWAN (Long Range Wide Area Network) allows the prototype to measure body temperature and transmit this via the LoRaWAN network, to the client interface. All levels of the LoRaWAN architecture are developed and adapted to the needs of the prototype, the measured temperature data being made accessible via the internet.

An approach is used based on a new type of LoRaWAN device architecture formed by a single microcontroller from the STM32WL family, these are the first microcontrollers to have an integrated LoRa transceiver. Development tools from the STM32Cube environment are also utilised. This in turn considerably simplifies the hardware and software design and helps to obtain better energy consumption results.

The consumption results accurately align with those obtained via simulations, especially in the low consumption mode of the microcontroller. Observations indicate that this is the mode that most affects consumption during live usage, as it is the one that consumes the most time.

2.1. KEY WORDS

LoRa and LoRaWAN, STM32WL, The Things Network, wearable device, Internet of things (IoT).

3. INTRODUCCIÓN

El Internet de las cosas (IoT) se ha convertido en una de las tecnologías más prometedoras y con un gran crecimiento en los últimos años. En 2018 se estimaba que había alrededor de 22 mil millones de dispositivos IoT conectados a internet, mientras que para el 2025 se estiman unos 38.6 mil millones de dispositivos conectados y para el 2030 se pronostican unos 50 mil millones [1]. Gracias a esta tecnología los usuarios pueden disponer fácilmente de los datos ambientales o fisiológicos, registrados por distintos dispositivos, en cualquier momento y lugar a través de internet.

El desarrollo de tecnologías que permitan la escalabilidad de los dispositivos IoT es necesario para poder hacer frente al aumento de los dispositivos conectados. Las redes amplias de baja potencia (LPWAN), como LoRaWAN (Long Range Wide Area Network), ofrecen una buena solución para dispositivos IoT que se alimenten mediante batería y que requieran de un largo alcance de comunicación. Su reducido ancho de banda no supone un problema para la mayoría de los sensores, debido a que la cantidad de datos transmitida es muy reducida, por lo que son más importantes otros factores como el gran número de dispositivos que se pueden conectar a una sola puerta de enlace, el bajo consumo energético o el largo alcance.

Es por ello por lo que se desarrolla del prototipo de un dispositivo ponible (wearable) IoT que emplee redes LPWAN, concretamente la red LoRaWAN, con el objetivo de crear un dispositivo de bajo consumo capaz de enviar la temperatura corporal a través de toda la arquitectura de red hasta el cliente final.

Para ello se deberán cumplir los siguientes objetivos o características:

- Desarrollar un prototipo funcional capaz de conectarse a una puerta de enlace mediante el protocolo LoRaWAN.
- La implementación y configuración de los distintos niveles de la arquitectura LoRaWAN, incluyendo la puerta de enlace o Gateway, el servidor de red y de aplicación y el cliente final en el que se visualizarán los datos.
- El cliente final debe mostrar los datos de temperatura medidos por el prototipo de forma adecuada y en tiempo real, junto con otros parámetros de la conexión como el SNR y el RSSI.
- El consumo energético del prototipo debe ajustarse a los consumos obtenidos mediante la herramienta de simulación ofrecida por la compañía del microprocesador empleado.

3.1. ANTECEDENTES

En los siguientes apartados se realizará un estudio de los proyectos y diseños actuales de dispositivos IoT ponibles (wearable), enfocados en la monitorización de parámetros fisiológicos y/o transmisión de los datos a internet.

El objetivo de este estudio es conocer los diseños, tecnologías y metodologías empleados en el desarrollo de otros proyectos, para así obtener una visión más amplia del sector y llegar a conclusiones generales que ayuden diseño adecuado del dispositivo.

3.1.1. Casco inteligente para bomberos forestales[2]

Se trata de un prototipo de casco inteligente diseñado para medir datos vitales de bomberos forestales y enviarlos a su supervisor. Los principales datos que recopila el casco son de temperatura, de frecuencia cardiaca y de aceleración. Estos datos son empleados para alertar al supervisor sobre posibles problemas de salud relacionados con el calor, posibles caídas o ritmos cardiacos anormales en los bomberos durante los turnos de trabajo [2]. Esto resulta de vital importancia debido a las condiciones extremas a las que se ven expuestos los bomberos durante sus misiones, como en los incendios forestales.

Este prototipo se basa en un anterior diseño de casco inteligente, el cual también busca alertar al supervisor del estado de los bomberos y prevenir problemas como la deshidratación u otras enfermedades relacionadas con el calor o el esfuerzo excesivo. Pero este diseño plantea un problema en la comunicación con el supervisor, puesto que depende de la cobertura de móvil para transmitir datos, la cual no siempre es confiable o no está disponible en las ubicaciones típicas de trabajo de un bombero forestal.

Para solventar este problema se desarrolla el prototipo actual, en el cual se elimina la dependencia de la cobertura móvil mediante el uso de LoRa (protocolo inalámbrico para comunicación de largo alcance y baja potencia). Con esto se consigue aumentar la autonomía de los dispositivos, para poder funcionar durante los turnos de hasta 16 horas de los bomberos forestales [2], y solventar los problemas de disponibilidad de cobertura móvil.

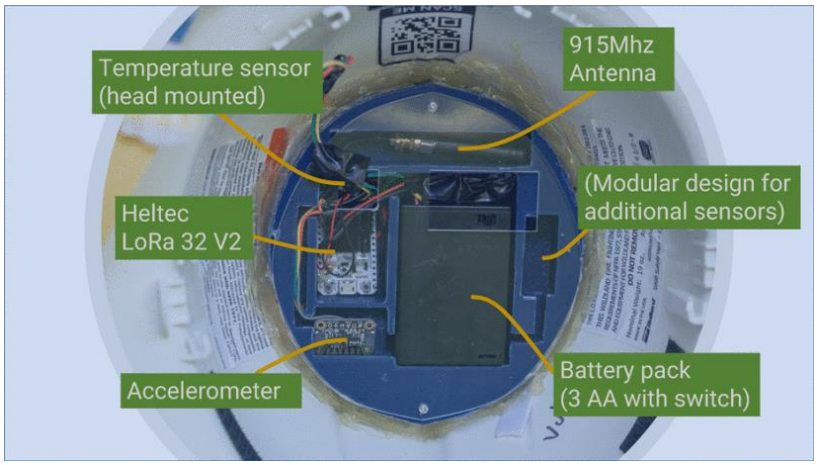

Figura 1. Prototipo casco de bombero forestal. [2]

Para el prototipo han empleado el microprocesador Heltec Wifi LoRa 32 (V2), el cual se basa en un ESP32 con capacidad Bluetooth y un módulo recepción y transmisión LoRa. Consta de tres sensores:

- El sensor Polar OH1+, un sensor de frecuencia cardíaca colocado en la sien del bombero o en algún lugar del brazo (conectado con el casco mediante Bluetooth).
- El sensor GY-906MLX90614ESF, el cual es un sensor de temperatura por infrarrojos situado en la frente del bombero.
- El sensor Adafruit LIS3DH, un acelerómetro colocando dentro del casco.

Los datos se transmiten entre los bomberos mediante LoRa en una red de malla y el casco del supervisor recibe los datos transmitidos. El casco del supervisor está conectado a una aplicación a través de bluetooth, de esta forma puede visualizar el estado de los bomberos en tiempo real. Cuando se detecta una situación de emergencia, se transmite una señal de alerta para que el supervisor pueda tomar las decisiones oportunas.

Con este prototipo se realizaron dos pruebas en Olympia, Washington, con bomberos forestales del Departamento de Recursos Naturales (DNR). En la prueba de distancia de transmisión LoRa, se mostró una alta perdida de paquetes a unos 800 metros entre el casco del supervisor y un casco de bombero, lo cual es significativamente superior a la distancia promedio entre bomberos en el campo, alrededor de 100m. También es superior al requisito de distancia del DNR, el cual es de 500m. [2]

Esta caída en la recepción de paquetes a partir de los 800m puede deberse a que a esa distancia los bomberos llegaron a una región del valle en las colinas, en la que las rocas pudieron interferir en la señal, junto con la alta densidad de árboles.

3.1.2. Sistema de emergencia discreto de largo alcance [3]

Se propone un sistema de emergencia de bajo costo integrado en un zapato, el cual es capaz de reconocer gestos con los pies y transmitir un aviso a larga distancia en caso de emergencia. De esta forma el usuario será capaz de notificar a sus contactos de emergencia de manera discreta mediante el empleo de gestos específicos con los pies, mientras se realizan otras actividades como caminar.

El prototipo cuenta con dos rasgos principales, ser un dispositivo discreto y de bajo costo que pueda operar de forma independiente en ciudades inteligentes, y la capacidad para distinguir entre varias actividades y gestos con los pies mediante una IA integrada en dispositivos IoT. Por lo que el sistema se puede dividir en dos partes, la interfaz de los gestos basada en Machine Learning (ML), y la transmisión de la señal mediante LoRa.

Para la interfaz de gestos se emplean dos sensores de fuerza de bajo costo, colocados debajo de la suela del zapato, uno en la punta y otro en el talón. El modelo ML está diseñado para ejecutarse con los recursos de la MCU, para ello se desarrolla un clasificador basado en una red neuronal (Neural Network) ligera, capaz de diferenciar 3 actividades (caminar, trotar, estar de pie) y dos gestos (doble toque con el talón o con la punta del pie). Este modelo de red neuronal liviano se implementa y se ajusta en Python para posteriormente transferirse a Python.

Al implementar el modelo de ML directamente en la MCU del dispositivo final, se evita el envió excesivo de información de todos los datos capturados por los sensores, puesto que estos datos son procesados directamente en el dispositivo. Una vez procesados con el modelo de ML, se obtiene únicamente una de las 3 actividades o 2 gestos, con una precisión general del 97,5% [3]. Gracias a esto es posible emplear LoRa para la transmisión de la señal de emergencia, puesto que LoRa se limita a la transmisión de pequeñas cantidades de información.

Figura 2. Prototipo de zapato capaz de reconocer gestos de emergencia.[3]

Este dispositivo LPWAN (Low Power Wide Area Networks) está formado por los dos sensores de fuerza mencionados anteriormente, un ESP32, el cual será la MCU principal que contendrá el modelo ML, y el módulo RFM95 LoRa para la transmisión y recepción de la señal sin necesidad de llevar un teléfono móvil ni depender de puertas de enlace de corto alcance.

3.1.3. Dispositivo de seguimiento de personas con demencia [4]

Este dispositivo busca monitorear la ubicación de personas con demencia, las cuales se pueden salir de su residencia y desorientarse con facilidad. El prototipo propuesto puede mostrarles el camino de vuelta a casa cuando abandonen una geovalla y alertar a su cuidador o familiares, también les recordará sus tareas diarias o medicamentos.[4]

Para alertar al cuidador o familiares se emplea la tecnología de comunicación LoRa con una puerta de enlace en el centro de residencia del paciente, de esta forma se evita el empleo de dispositivos móviles o la dependencia de una red de cobertura móvil. También se consigue un consumo de energía muy bajo, algo importante para facilitar el uso del dispositivo y la seguridad del paciente al disminuir el riesgo de que se quede sin batería.

Para determinar la posición de los pacientes se hace uso de GPS. Puesto que uno de los objetivos principales es reducir el consumo energético del dispositivo, el GPS se actualizará con menor frecuencia cuando el paciente se encuentra dentro de la geovalla y aumentará la frecuencia de actualización cuando el paciente la abandone, al mismo tiempo que se enviaran las coordenadas y el aviso de abandono mediante LoRa.

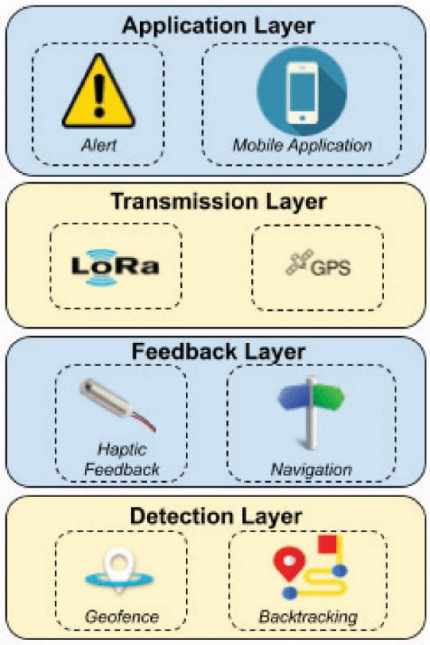

Figura 3. Diagrama de arquitectura del dispositivo portátil. [4]

El sistema se divide en un dispositivo portátil, una puerta de enlace y una plataforma en la nube. El dispositivo portátil está formado por tres elementos principales, un módulo GPS (u-blox NEO-6), un motor de vibración para obtener una retroalimentación háptica y el módulo Heltec LoRa 32, el mismo empleado en el casco de bomberos forestales visto anteriormente [2]. Este módulo contiene un microcontrolador ESP32, el receptor y transmisor de LoRa SX1276 y una pantalla OLED que le mostrará datos al paciente.

3.1.4. Reloj de pulsera para monitorización de SpO_2 y frecuencia cardiaca [5]

Se trata de un dispositivo con forma de reloj de pulsera que busca monitorizar de forma continua la saturación de oxígeno en sangre (SpO_2) y la frecuencia cardiaca mediante un sensor de fotopletismografía óptica (PPG). El dispositivo se comunica inalámbricamente empleando las bandas de 868 MHz (sub-GHz).

A diferencia de los otros prototipos mencionados anteriormente, este busca un diseño completo y compacto en el que también se desarrolla una antena de 868 MHz altamente integrada, impresa directamente en la superficie de la carcasa del reloj. También se estudia con detalle los beneficios de usar las bandas de 868 MHz frente a la de 2,45 GHz. En dicho estudio llega a la conclusión de que la banda 868 MHz experimenta una menor atenuación de la señal transmitida, se logra un rango de comunicación mayor y un consumo de energía mucho menor al requerir menor potencia para lograr un rango de comunicación similar al de la banda 2,45 GHz [5].

Otra diferencia con los prototipos antes mencionados es que se emplea el protocolo de comunicación MiWi en lugar de LoRa, aunque ambos emplean la banda de radio Sub-GHz. MiWi es un protocolo inalámbrico de la compañía Microchip, que utiliza una tasa de datos baja y radio de baja potencia [5], [6]. Por lo que emplea el microcontrolador de Microchip ATSAMR30E18A, el cual cuenta con un transceptor integrado de banda Sub-GHz de ultra baja potencia que puede emplear el protocolo MiWi.

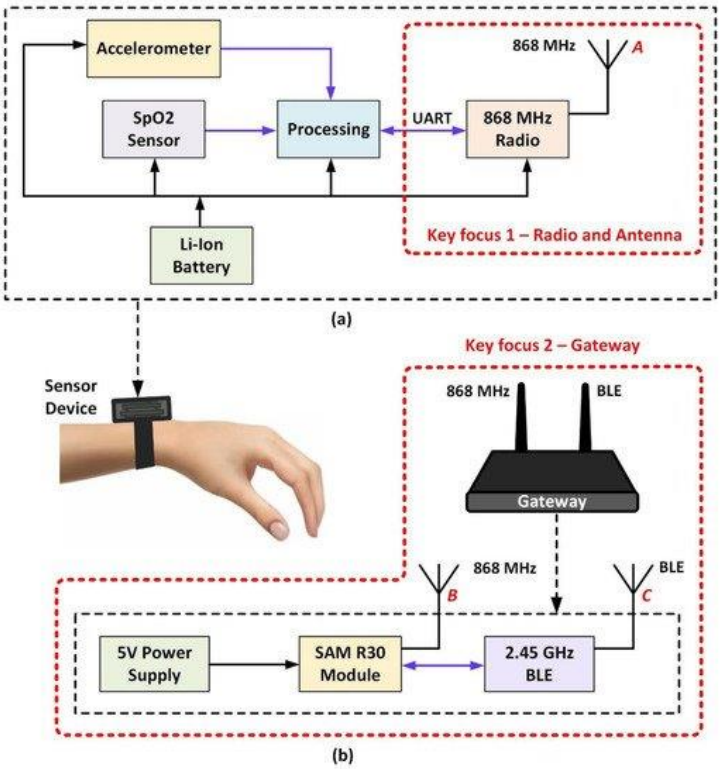

Figura 4. Esquema del hardware del reloj de pulsera [5].

El reloj de pulsera que contiene el sensor SpO_2, un acelerómetro, una batería de iones de litio, un procesador y la unidad de radio de 868 MHz con la antena, como se muestra en Figura 4. La unidad de radio utiliza el microcontrolador (MCU) ATSAMR30E18A, al igual que en el Gateway. El empleo de este microcontrolador simplifica y optimiza el diseño al tener incorporado el módulo de radio.

3.1.5. Dispositivos comerciales

3.1.5.1. CareBand 4 [7]

CareBand 4 es un dispositivo portátil estilo reloj de pulsera desarrollado por la compañía CareBand. Este dispositivo cuenta con comunicación inalámbrica mediante la red LoRaWAN.

El dispositivo está diseñado para el seguimiento de personas distintos entornos. El dispositivo y sus servicios se puede personalizar en

función del uso a través del portal de CareBand y la APIs/GraphQL [7]. Los usos propuestos por la compañía van desde el tratamiento de adicciones hasta el seguimiento de la ubicación de niños, persona con demencia o Alzheimer o el de pacientes hospitalizados.

Para cumplir con estas aplicaciones el dispositivo cuenta con Bluetooth, carga inalámbrica, sensores para determinar si la persona lleva puesta la pulsera, acelerómetro, un led frontal, respuesta háptica y un botón que puede configurarse para distintos usos como el envió de alertas.

Figura 5. Dispositivo CareBand 4 [7]

3.1.5.2. Botón de pánico LW004-PB [8]

LW004-PB es un dispositivo IoT de la compañía Moko Smart, es compacto, fácil de usar mediante su botón y se comunica inalámbricamente mediante la red LoRaWAN. Se emplea como botón de pánico, el cual enviará un aviso de alerta y la ubicación actual al servidor cuando se presione el botón en caso de emergencia.

A diferencia del dispositivo anterior, la CareBand 4 [7], este botón no envía la información de la ubicación automáticamente al salir de una geovalla, si no únicamente cuando se pulsa el botón, por lo que no está diseñado para personas dependientes.

El dispositivo cuenta con GPS y Bluetooth, además de un motor de vibración para una respuesta háptica y un acelerómetro para la detección de movimiento.

Figura 6. Dispositivo LW004-PB [8]

3.1.5.3. Wearloc-2 [9]

Se trata de un reloj desarrollado por ProEsys Technologies, el cual permite un posicionamiento de alta precisión mediante GNSS y monitorear las actividades biológicas de la persona mediante un biosensor [9]. Estos datos pueden ser visualizados en la pantalla del dispositivo y se envían en tiempo real mediante la red LoRaWAN.

El reloj también cuenta con un sensor de "hombre caído", para detectar si el usuario ha sufrido algún tipo de caída, un sensor de temperatura, un sensor de presión atmosférica y una brújula.

Es un dispositivo ideado para rastrear a personas y sus constantes en lugares al aire libre donde no existe cobertura de otros sistemas de comunicación, por lo que el empleo de la tecnología LoRa permite establecer esta comunicación de largo alcance en sitios de difícil acceso de una forma más fácil.

Figura 7. Wearloc-2 [9]

4. DESARROLLO

4.1. MARCO TEÓRICO

4.1.1. LPWAN (Low Power Wide Area Networks)

El término LPWAN (Low Power Wide Area Networks), cuyas siglas se pueden traducir como redes de baja potencia y área amplia, hace referencia a una categoría redes inalámbricas que han sido diseñadas para transferir pequeñas cantidades de datos a largas distancias manteniendo un muy bajo consumo energético [10], [11].

Las redes LPWAN poseen tres características principales:

- Bajo consumo energético. Esto permite que los dispositivos IoT, normalmente sensores, lleguen a funcionar hasta un máximo de 10 años con una batería tipo botón o AAA [12].
- Largo alcance. Permite cubrir territorios extensos al poder transmitir datos con varios kilómetros de separación entre los nodos y la antena de recepción [10].
- Bajo ancho de banda. La principal desventaja de las redes LPWAN es el bajo volumen de datos que pueden transmitir, pero es lo que permite mantener un bajo consumo energético con un largo alcance [12].

Actualmente no existe una tecnología que permita tener un bajo consumo energético al mismo tiempo que se mantiene un largo alcance y un amplio ancho de banda. Por ejemplo, el bluetooth tiene un bajo consumo y un ancho de banda medio, pero su alcance es muy bajo. Otro ejemplo sería el 4G, el cual tiene un buen alcance y un gran ancho de banda, pero un consumo mayor. [13]

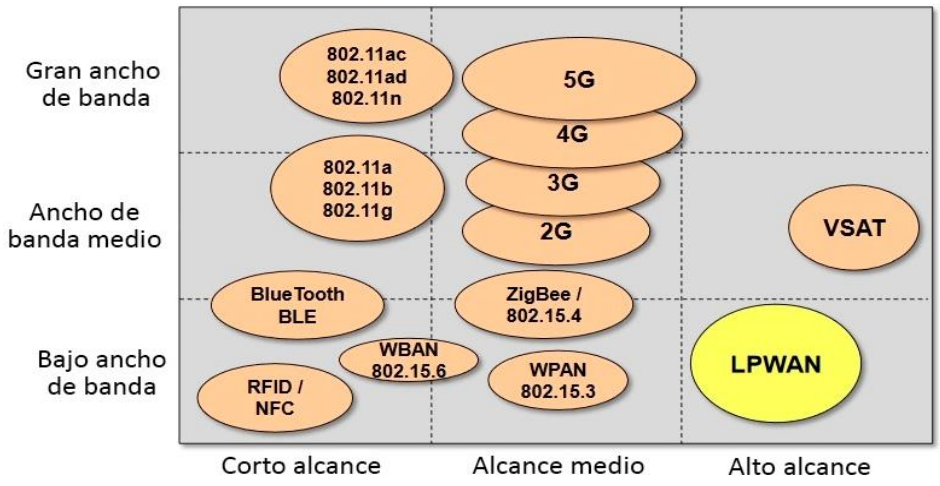

Figura 8. Comparación de tecnologías de transmisión de datos en función del ancho de banda (BW) y alcance. [12]

Estas características hacen que las redes LPWAN sean perfectas para IoT, puesto que la mayoría de los dispositivos IoT, especialmente los sensores, no necesitan un gran ancho de banda, sino un bajo consumo energético que permita aumentar su autonomía.

A la hora de elegir una tecnología LPWAN nos encontramos con numerosas opciones. Alguna de las principales son LoRaWAN, Sigfox, NB-IoT (Narrowband-IoT) y LTE-M (Long Term Evolution for Machines). Las dos últimas son sistemas de espectro licenciado y uso exclusivo, y permiten una mayor velocidad de datos, especialmente LTE-M, la cual permite enviar voz con una velocidad de hasta 1Mbps y está preparada para aplicaciones de movilidad, mientras que NB-IoT está pensada para aplicaciones estáticas que requieran una velocidad de datos menor, también tiene la ventaja de que el coste de sus módulos es menor [14], [15].

Por otro lado, Sigfox y LoRaWAN son de espectro no licenciado, aunque para utilizar la infraestructura de Sigfox los dispositivos deben suscribirse a su servicio de pago. [10]

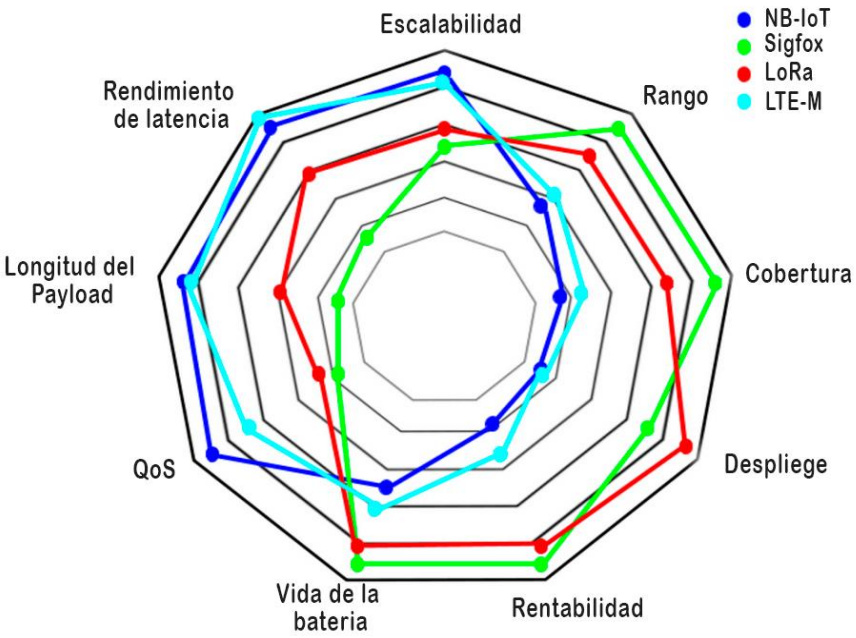

Figura 9. Comparación de tecnologías LPWAN. [16].

4.1.2. LoRa (Long Range)

Como se ha dicho en el apartado anterior, la red LoRaWAN se encuentra dentro de la categoría LPWAN, esta red es posible gracias a la modulación LoRa.

LoRa es una tecnología de modulación de radio frecuencia (RF) creada y patentada por Semtech Corporation, la cual opera típicamente en la banda de los 868 MHz en Europa. El nombre LoRa proviene de la abreviación de Long Range (largo alcance), por el largo alcance de sus

transmisiones. LoRa hace referencia únicamente a la implementación de la capa física (PHY) que se encarga de realizar la técnica de modulación patentada basada en la tecnología Chirp Spread Spectrum (CSS). [17]

Puesto que se trata de una tecnología LPWAN, posee sus características de largo alcance, bajo consumo energético y bajo ancho de banda. Mediante LoRa se pueden realizar comunicaciones de hasta 5 km en áreas urbanas y hasta 15 km o más en áreas rurales [17]. En cuanto al consumo de energía, una batería tipo botón podría alimentar a un sensor durante varios años (típicamente entre 3 y 5), aunque esto dependerá de los parámetros de la comunicación, el hardware y la configuración del dispositivo y otros factores [18]. Otra de sus ventajas es que una sola puerta de enlace puede admitir a unos 10.000 dispositivos, por lo que se requieren menos puertas de enlace, reduciendo los costes de la red [17].

4.1.2.1. Modulación LoRa

LoRa se basa en una variante de la modulación de espectro ensanchado (Spread Spectrum), llamada Chirp Spread Spectrum (CSS). Este nombre se debe a que emplea "pitidos" (Chirps) en lugar de "códigos" para modular la señal, como en el caso de Spread Spectrum [19]. Emplea factores de propagación (SF) ortogonales, esto permite realizar variaciones en el alcance y la velocidad de los datos de un nodo final para optimizar la duración de la batería en función de la distancia hasta la puerta de enlace [17]. Cuanto mayor es el SF, mayor es la distancia a la que se puede transmitir sin errores, pero el consumo de batería aumenta y la tasa de bits se reduce.

El factor de propagación (Spreading Factor) puede tomar seis valores del 7 al 12 (SF7 - SF12), aumentar el SF reduce la tasa de bits puesto que esta es proporcional al ancho de banda (BW) e inversamente proporcional a 2^{SF}, siguiendo la ecuación:

$$Bit\ Rate = SF \cdot \frac{BW}{2^{SF}} \qquad \textit{Ecuación 1}$$

Donde el ancho de banda (BW) es fijo y puede tomar valores de 125 kHz, 250 kHz o 500 kHz [20].

Tabla 1. comparación de distintos factores de propagación. [17]

Factor de propagación (SF)	Bite Rate	Tiempo en el aire para un Payload de 10 bytes
SF12	293 bps	1483 ms
SF11	537 bps	741 ms
SF10	977 bps	371 ms
SF9	1758 bps	205 ms
SF8	3125 bps	103 ms
SF7	5469 bps	56 ms

La tasa real de bits es más baja si tenemos en cuenta el Coding Rate (CR), el cual incrementa el número de bits transmitidos para llevar a cabo la detección y corrección de errores. La tasa de bit que realmente contiene información será 4/(4+CR) donde CR puede tomar valores entre 1 y 4. Por lo que, para un CR de 4, el número de bits con información real es la mitad del total.

Los paquetes que se envíen en el mismo canal con SF distintos son invisibles entre sí, puesto que los factores de propagación son ortogonales. Por lo que dos paquetes que lleguen al mismo tiempo en el mismo canal no interferirán. Sin embargo, si tienen el mismo SF se podría provocar una colisión, aunque esto dependerá de la diferencia de dB entre los paquetes. [17]

Los "Chirps" empleados son esencialmente una señal sinusoidal que aumenta o disminuye su frecuencia recorriendo el ancho de banda del canal. La forma de onda de un Chirp se puede ver representada en la Figura 10.

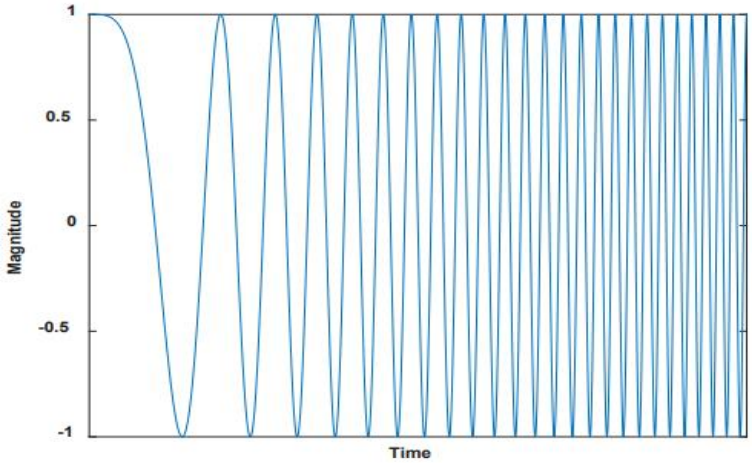

Figura 10. Simulación en Matlab de la forma de onda de un Chirp. [19]

Un solo Chirp representan un número de bits igual al SF, por ejemplo, un Chirp en una transmisión con un factor de propagación de 9 representa un conjunto de 9 bits. Cada Chirp se diferencia únicamente en la frecuencia en la que empieza, lo que provoca distintas formas para cada Chirp. Estas formas o frecuencias de inicio determinan todo el conjunto de bits, por lo que tendrá 2^{SF} formas posibles. Continuando con el ejemplo anterior, para un SF9, el Chirp podrá empezar en 512 frecuencias distintas.

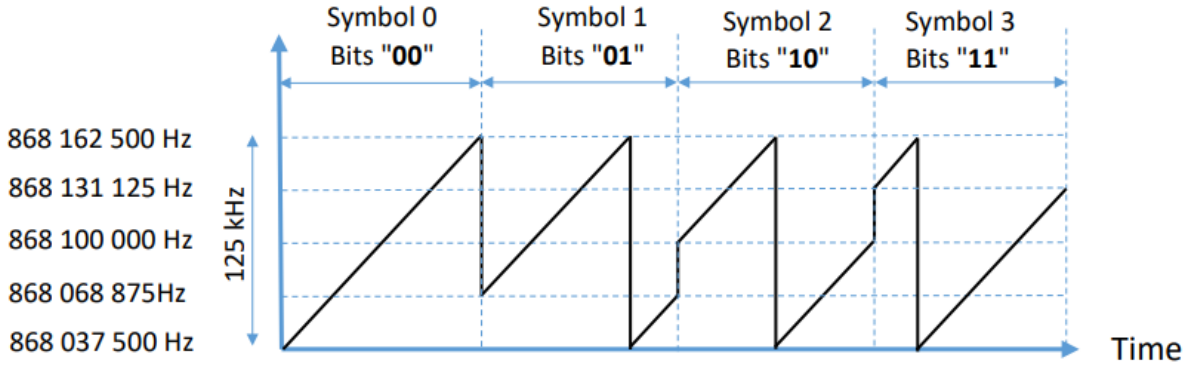

Figura 11. Ejemplo de forma de los Chirps para una transmisión con un SF2 (conjuntos de 2 bits), un ancho de banda de 125 kHz y una frecuencia de 868,1 MHz. [19]

En una transmisión LoRa, la información que queremos transmitir, también llamada carga útil (Payload), se transmite junto con otra información necesaria para realizar la conexión y verificar su integridad. Por lo que señal transmitida empieza con un preámbulo que permite sincronizar la recepción y una cabecera, seguido de la carga física útil (PHY Payload) y por último el CRC, encargado de verificar la integridad del mensaje. [19]

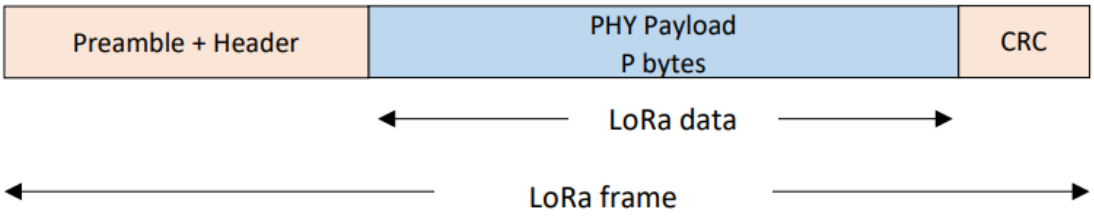

Figura 12. información transmitida en una transmisión LoRa. [19]

4.1.2.2. Magnitudes y conceptos de las transmisiones de radio.

Para determinar la calidad de una conexión entre un transmisor y un receptor se emplean distintos parámetros, como el RSSI, la sensibilidad, SNR o el balance de enlace (Link Budget).

Para explicar estos conceptos, debemos saber que cuando un transmisor transmite una señal a una determinada potencia (P_T), el receptor solo recibe una parte de la potencia transmitida (P_R) junto con ruido del entorno (P_N). [19]

Figura 13. Transmisión de radio frecuencia.[19]

El RSSI (Received Signal Strength Indicator), o indicador de fuerza de la señal recibida, es la potencia que recibe el receptor (P_R) del transmisor, sin contar el ruido del entorno. Se expresa en dBm, una unidad de medida de potencia en dB con relación a 1 milivatio (mW), siguiendo la expresión:

$$Power\ (dBm) = 10 \cdot \log_{10}\left(\frac{Power(W)}{0{,}001}\right) \qquad \textit{Ecuación 2}$$

La sensibilidad es la mínima potencia recibida (P_R), o RSSI, que debe llegar al receptor para que pueda leer la señal.

El SNR (Signal over Noise Ratio), es la relación señal-ruido, se refiere a la proporción entre la potencia de la señal recibida y la potencia del ruido que interfiere.

El balance de enlace o Link Budget es la diferencia entre la potencia de transmisión (P_T) y la sensibilidad del receptor. Por lo que cuando las pérdidas de potencia son mayores al Link Budget, la conexión se perderá. Para determinar si la conexión es posible, se debe comparar la sensibilidad del receptor con la sensibilidad que llega según la ecuación básica:

$$P_R\ (dBm) = P_T\ (dBm) + \ Ganancias\ (dB) - \ Perdidas\ (dB) \qquad \textit{Ecuación 3}$$

Si la potencia recibida (P_R) es menor a la sensibilidad, la conexión no será posible. Si se desarrolla la ecuación teniendo en cuenta las distintas pérdidas y ganancias típicas en un sistema de radio, tenemos la ecuación:

$$P_R = P_{TX} + G_{TX} + G_{RX} - L_{TX} - L_{FS} - L_P - L_{RX} \qquad \textit{Ecuación 4}$$

Donde:

P_R = Potencia recibida (dBm)

P_T = Potencia de salida del transmisor (dBm)

G_{TX} = Ganancia de la antena del transmisor (dBi)

G_{RX} = Ganancia de la antena del receptor (dBi)

L_{TX} = Pérdidas del transmisor (alimentador, conector, etc.) (dB)

L_{FS} = Pérdidas en el espacio libre o en el trayecto (dB)

L_P = Pérdidas misceláneas de propagación de la señal (margen de desvanecimiento, pérdida de cuerpo, desajuste de polarización y otras perdidas) (dB)

L_{RX} = Pérdidas del receptor (alimentador, conectores, etc.) (dB)

En cuanto al Link Budget de LoRa suele estar entorno a los 157 dB, superando a otras tecnologías como el 4G, con un Link Budget de unos 130 dB. [19]

4.1.3. LoRaWAN

LoRaWAN es un estándar de red de área amplia y baja potencia (LPWAN) desarrollada y mantenida por LoRa Alliance, una asociación abierta sin ánimo de lucro creada en 2015. [21]

Es importante diferenciar entre LoRa y LoRaWAN. LoRa, como se vio en el apartado anterior, se refiere a la capa física encargada de realizar la modulación que permite la conexión entre los dispositivos. Mientras que LoRaWAN se refiere a toda la arquitectura de red, desde los nodos finales hasta los servidores o aplicaciones.

LoRaWAN es un protocolo diseñado para reducir el consumo energético de los dispositivos, proporcionar comunicaciones bidireccionales seguras de extremo a extremo mediante parámetros y especificaciones estandarizados. [17], [22]

4.1.3.1. Arquitectura de la red LoRaWAN

La arquitectura de LoRaWAN utiliza una topología de estrella, en la que los distintos dispositivos finales transmiten a una o más puertas de enlace (Gateways), y las distintas puertas de enlace se conectan a los servidores como se muestra en la Figura 1. Todos los elementos pueden comunicarse bidireccionalmente. También hay soporte para realizar una multidifusión y realizar tareas como actualizaciones de firmware por aire (FOTA), esta configuración aumenta el rango de red, pero la sobrecarga, añade complejidad y aumenta el consumo de energía de los nodos. [22]

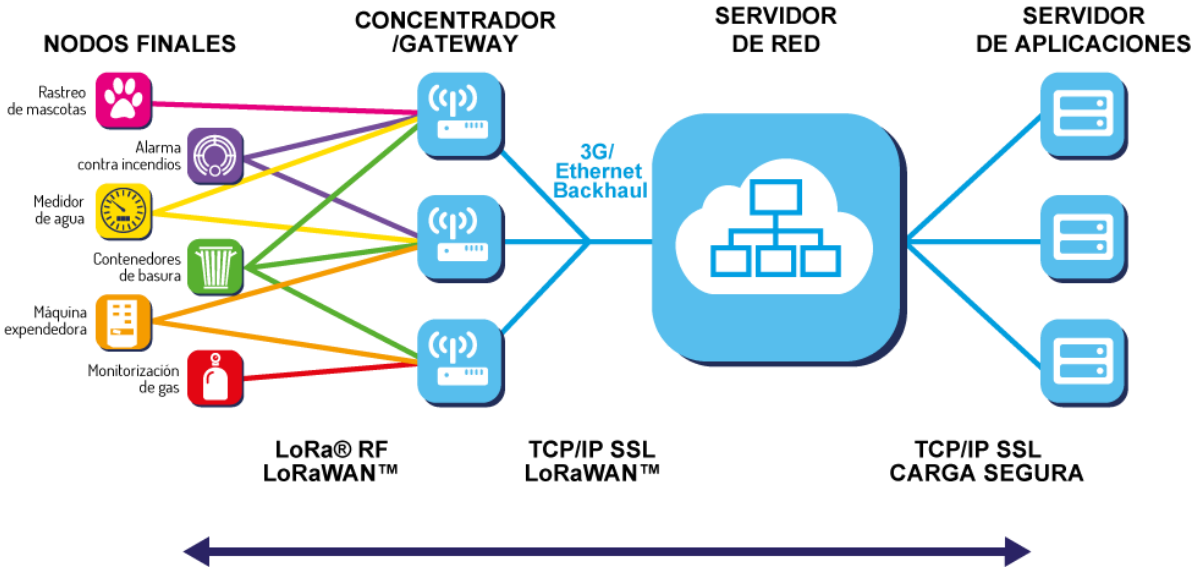

Figura 14. Arquitectura de red LoRaWAN. [23]

Está formada principalmente por cuatro elementos: los dispositivos finales, las puertas de enlace, los servidores de red y los servidores de aplicación.

- Dispositivos finales (nodos finales):

  Estos son los distintos sensores o dispositivos que recopilan información. Estos dispositivos envían y/o reciben información de los Gateways. No están asociados a una única puerta de enlace, por lo que la transmisión puede ser recibida por varias puertas [23]. El modo y el momento en el que transmiten o reciben la información depende de la clase del dispositivo, la cual puede ser A, B o C. La diferencia entre cada una será explicada en el apartado 4.1.3.3.

- Las puertas de enlace (Gateway):

  Actúan como puentes bidireccionales entre los nodos finales y el servidor de red, para ello convierte los paquetes RF en paquetes IP y viceversa [22]. Pueden escuchar en todos los canales de LoRa y en todos los factores de propagación (SF) al mismo tiempo. El Gateway puede conectarse a internet mediante Wifi, 3G, 4G, 5G, Ethernet…. Cada Gateway tiene un identificador único (64 bits EUI) que se utiliza para registrar y activar el Gateway en el servidor de red. [19]

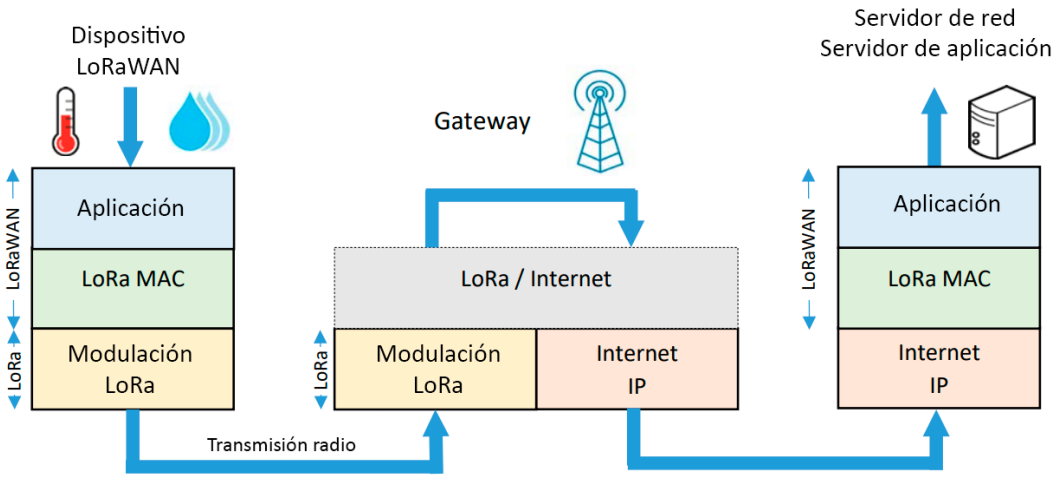

Figura 15. Esquema de la información que recibe y envía un Gateway. [19]

- El servidor de red:

  El servidor de red recibe los mensajes de las puertas de enlace y elimina los mensajes duplicados (mensajes de los nodos finales que han llegado a más de un Gateway). El servidor de red es el encargado de garantizar la autenticidad de los dispositivos de la red, para ello emplea una clave AES de 128 bits, esta clave tiene el nombre de NwkSKey (Network Session Key), se emplea únicamente para la autentificación, no para encriptar [19]. El servidor de red no es capaz de ver los datos que se están transmitiendo.

  También realiza otras funciones de administración, controlando dinámicamente los parámetros de la red adaptándola a las distintas condiciones. Decide que Gateway debe transmitir un mensaje al nodo final, también gestiona el nodo final configurando las velocidades de transmisión de datos (mediante el factor de propagación) para así maximizar la capacidad de la red y reducir el consumo energético de los dispositivos finales. [20]

- Servidores de aplicación:

  Los servidores de aplicación gestionan e interpretan los datos de los dispositivos finales, también pueden generar y enviar información a los dispositivos. Los mensajes que recibe o envía están encriptados, para poder desencriptar y encriptar los mensajes se emplea una clave AES de 128 bits llamada AppSKey (Application Session Key). Este proceso se explicará con detalle en siguiente apartado 4.1.3.2.

4.1.3.2. Seguridad LoRaWAN

LoRaWAN utiliza algoritmos AES con dos claves de 128 bits. Una es la clave de sesión de red, llamada NwkSKey y compartida entre el dispositivo final y el servidor de red, que proporciona la autentificación e integridad de los paquetes al servidor de red. La segunda es la clave de sesión de aplicación (AppSKey), compartida de extremo a extremo entre el dispositivo y el servidor de aplicación, y se encarga del cifrado de los paquetes impidiendo que el servidor de red u otros elementos tengan acceso al contenido de los mensajes. [22]

La clave de aplicación (AppSKey), se usa para encriptar la información entre el dispositivo final y el servidor de aplicación. Se trata de un método de encriptación simétrico por lo que la AppSKey debe de ser igual en el dispositivo final y en el servidor de aplicación.

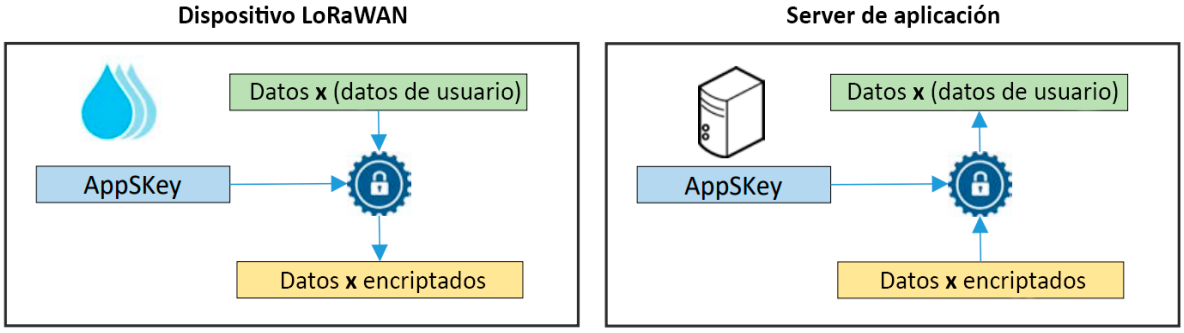

Figura 16. Proceso de encriptado y desencriptado. [19]

Lo mismo ocurre con la NwkSKey, aunque esta clave se emplea en el servidor de red para autentificar los mensajes y los dispositivos. Para esto se añade al paquete un código de integridad del mensaje (message integrity code, MIC). El MIC del paquete debe de coincidir con el MIC generado en el servidor de red a partir de la información encriptada recibida y la NwkSKey. [19]

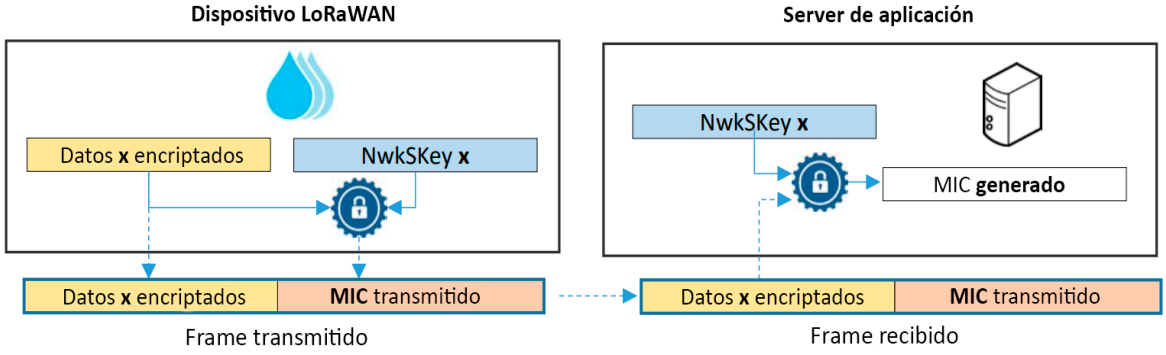

Figura 17. Autentificación del dispositivo en el servidor de red. [19]

Las claves se pueden activar mediante dos métodos. El primero es mediante activación mediante personalización (ABP), donde los dispositivos finales son programados en fábrica con la información de autentificación de una red LoRaWAN determinada. El segundo es

mediante activación por aire (OTAA), este es el método más empleado puesto que las claves no son predeterminadas y se pueden volver a generar. [20]

En el método de activación OTAA, la DevAddr (Device Addres) empleada para identificar el dispositivo, la AppSKey y la NwkSKey se generan durante el procedimiento de unión del dispositivo a la red. Este procedimiento de unión comienza con una petición de unión (Join-Request) y si es exitoso termina la aceptación de la unión (Join-Accept). Para poder generar esas claves y direcciones es necesario que tanto el dispositivo final como el servidor de red conozcan otros tres parámetros:

- El DevEUI: un identificador único e invariable para cada dispositivo LoRaWAN.
- AppEUI/JoinEUI: un identificador de aplicación único de 8 bytes. Para versiones LoRaWAN 1.0.3 se emplea la AppEUI, mientras que para versiones superiores a la 1.0.4. se emplea la JoinEUI en un servidor de unión (Join Server).

  Esto se debe a que a partir de la versión 1.0.4, el proceso de unión se lleva a cabo en un servidor específico llamado Join Server, de esta forma se evita que el servidor de red conozca la clave AppSKey empleada para encriptar los mensajes, aumentando la seguridad.
- AppKey: una clave AES de 128 bits empleada para autentificar la petición de unión y encriptar las claves generadas durante el proceso de unión.

Mediante estos tres parámetros almacenados tanto en el servidor como en dispositivo final, se generan la DevAddr, la AppSKey y la NwkSKey necesarias para la transmisión de cada mensaje. Los parámetros generados cambian cada vez que se realiza una nueva Join-Request.

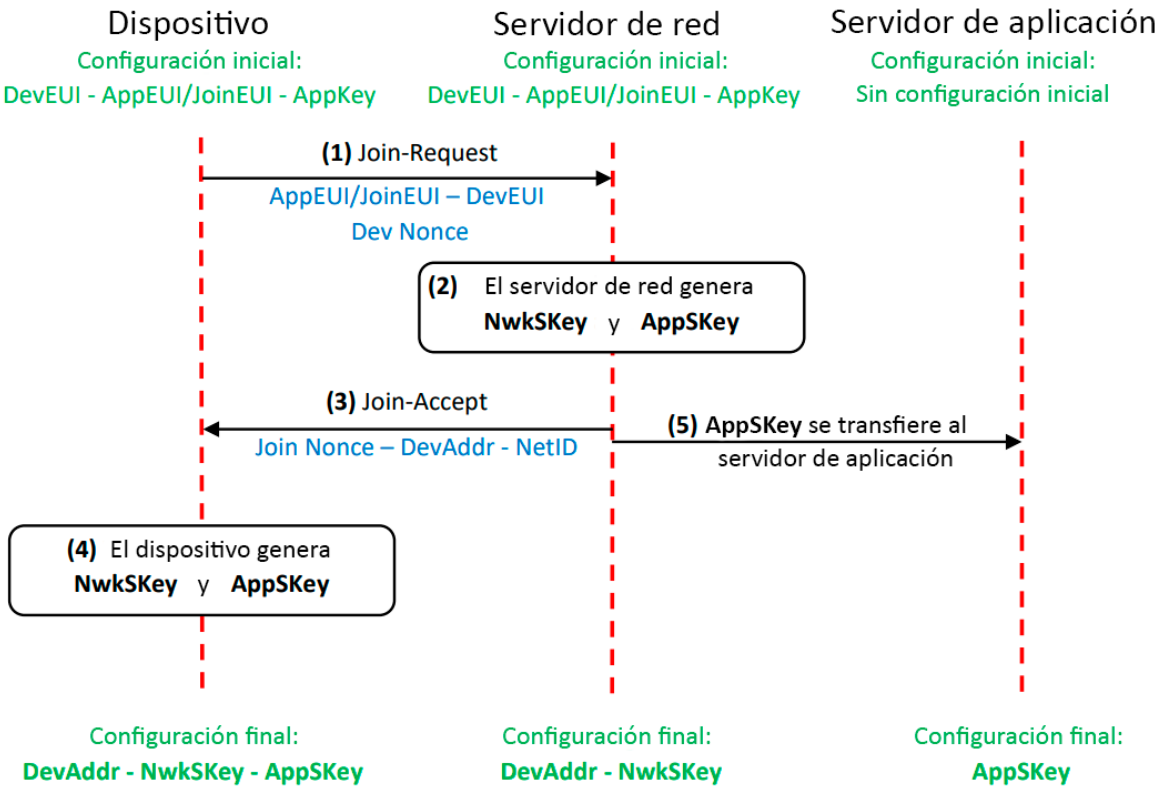

Figura 18. Procedimiento de unión a red LoRaWAN mediante OTAA. [19]

El diagrama de la Figura 18 se muestra el procedimiento de unión en el que se crean y transmiten los parámetros necesarios para realizar y encriptar las transmisiones.

Este proceso comienza con un Join-Requets enviado por el dispositivo final. Este Join-Request no está encriptado, pero contiene un MIC creado a partir de la AppKey, por lo que solo los dispositivos registrados en el servidor serán aceptados. Después de ser aceptado, el servidor genera la NwkSKey y la AppSKey y asigna una DevAddr al dispositivo. El tercer paso es enviar un Join-Accept, el cual se encripta usando la AppKey y contiene la DevAddr y el Join Nonce. Mediante el Join Nonce el dispositivo final puede regenerar la NwkSKey y la AppSKey. Por último, se envía la AppSKey al servidor de aplicación para que pueda desencriptar los mensajes posteriores. [19]

4.1.3.3. Clases de dispositivos finales.

Los dispositivos finales se clasifican en tres categorías o modos con el fin de poder ajustarse a las distintas capacidades. Estas categorías se diferencian en cuando los dispositivos pueden recibir los mensajes del servidor (donwlink). Los tres modos permiten la comunicación bidireccional y pueden enviar el mensaje al servidor (uplink) en cualquier momento. [20]

- Clase A:

  Los dispositivos de clase A tienen un consumo menor. El dispositivo puede transmitir (Uplink) en el momento que quiera de forma asíncrona. Cuando se finaliza una transmisión, se abren dos breves ventanas de tiempo, Rx1 y Rx2, en las que el dispositivo "escuchará" la respuesta del servidor. El servidor, a través del Gateway, puede transmitir (Downlink) en cualquiera de las dos ventadas de tiempo.

  Rx1 se abre una vez transcurrido el tiempo programado después del final del mensaje, y permanece abierta un breve periodo de tiempo (también configurable) que será por lo menos igual a la duración del preámbulo de mensaje, lo mismo ocurre con Rx2. Cuando se detecta el preámbulo la ventana permanecerá abierta hasta el final de la transmisión. [19], [22]

  Después de esto, el dispositivo puede permanecer inactivo (en modo de bajo consumo o suspensión) el tiempo que quiera, definido por la propia aplicación en función de su uso.

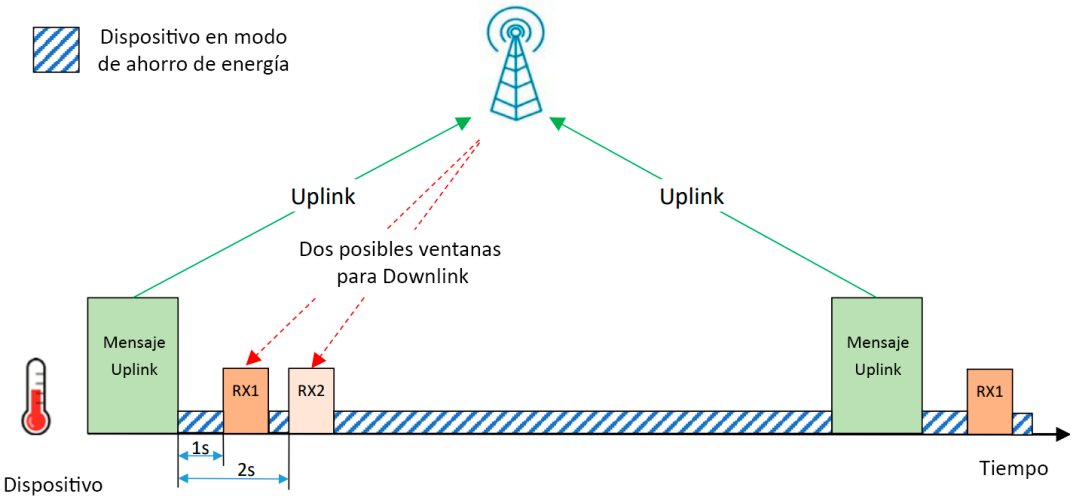

Figura 19. Ventanas de tiempo de recepción y transmisión en la clase A. [19]

- Clase B:

  Cuando se transmite un mensaje, se abren dos ventanas de recepción justo después de la transmisión, funcionando exactamente igual que la clase A. Pero en este modo también se abren ventanas de recepción adicionales de forma periódica sin necesidad de transmitir mensajes. Estas ventanas están sincronizadas con la red mediante balizas (beacons) enviadas de forma periódica. El tiempo entre cada ventana se puede configurar llegando hasta un máximo de 128 segundos entre dos ventadas de recepción [22].

Los dispositivos de clase B consumen más energía que los dispositivos de la clase A, aunque el consumo sigue siendo lo suficientemente bajo para poder ser usado en dispositivos con baterías.

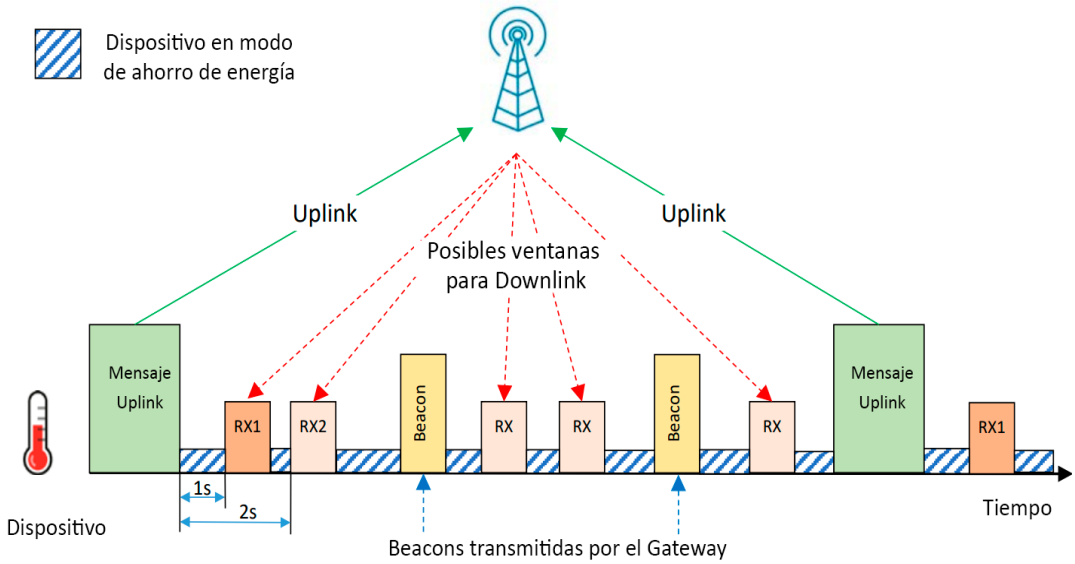

Figura 20. Ventanas de tiempo de recepción y transmisión en la clase B. [19]

- Clase C:

  En este modo la ventana de recepción está continuamente abierta excepto durante la transmisión de un mensaje. Por lo que el servidor puede transmitir en cualquier momento al dispositivo final. Esto tiene como inconveniente que el consumo energético aumenta en gran medida, por lo que esta clase está enfocada en dispositivos que dispongan de energía de forma continua.

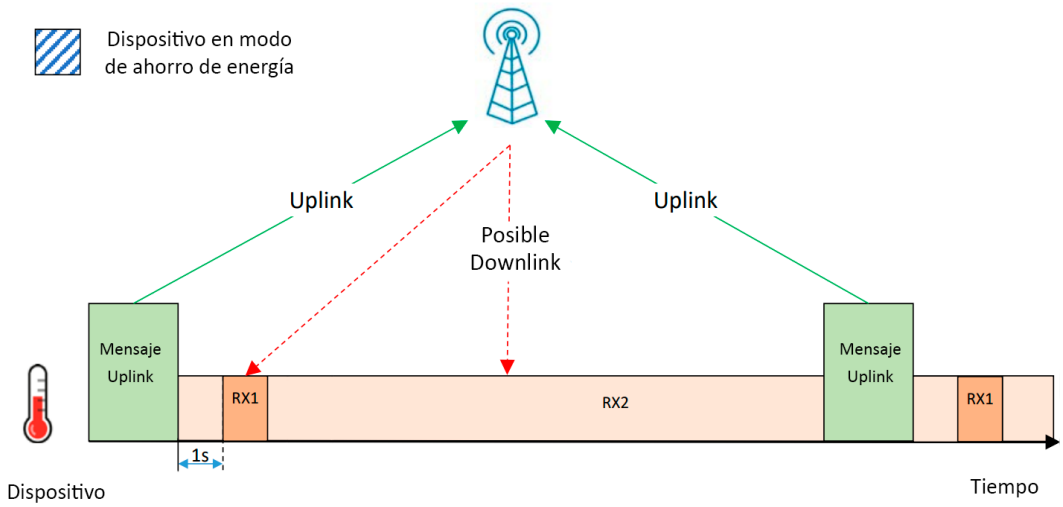

Figura 21. Ventanas de tiempo de recepción y transmisión en la clase B. [19]

4.1.3.4. Formato de las transmisiones LoRaWAN

Como vimos en el apartado 4.1.2.1 sobre la modulación de LoRa, cuando se transmite un mensaje, se envía un marco o cuadro (frame) que se divide en otros más pequeños. La capa física (PHY Layer) se divide en 4 frames y comienza con un preámbulo, seguido de una cabecera, un PHY Payload y, por último, el CRC.

En el PHY Payload se transmite la información del usuario junto con el resto de elementos necesarios para el funcionamiento de LoRaWAN, como la dirección del dispositivo, el código MIC y otros elementos para la administración de la red. Cada elemento se divide en un frame distinto, todos los frames situados dentro del PHY Payload pertenecen a la capa llamada LoRa MAC. En el siguiente esquema se puede ver los distintos frames que componen una transmisión LoRaWAN ordenados en las distintas capas:

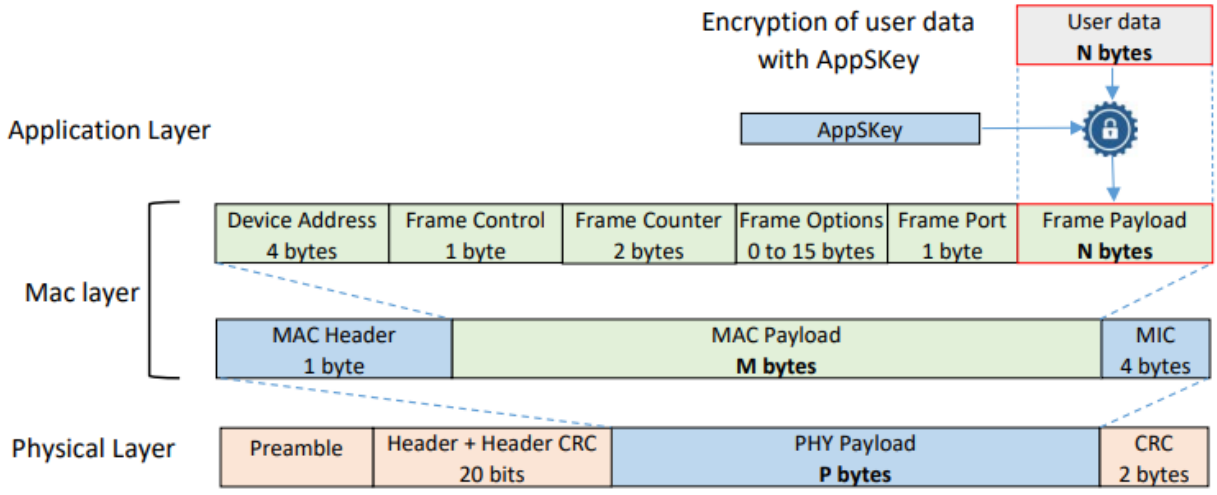

Figura 22. Formato de transmisión LoRaWAN. [19]

En la capa MAC se encuentra toda la información necesaria para el protocolo LoRaWAN:

- MAC Header (cabecera MAC), define el tipo del mensaje entre los 6 posibles: Join-Request, Join-Accept, unconfirmed data uplink, unconfirmed data downlink, confirmed data downlink, RFU, Proprietary. [24]
- Device Addres (DevAdrr), como se ha visto anteriormente, se trata de la dirección asignada al dispositivo.
- Frame Control, contiene la información sobre el ADR (Adaptive Data Rate), este parámetro será explicado en el siguiente apartado, el 4.1.3.5. También contiene el tamaño del Frame Option.

- Frame Counter, contiene la cuenta del número de mensajes enviados, con el fin de evitar los ataques de reinyección (Replay Attack).
- Frame Option, este parámetro es opcional, puede contener comandos MAC cuando sea necesario.
- Frame Port, contiene el puerto de aplicación.
- Frame Payload, contiene la información encriptada del usuario, es la carga útil que se desea transmitir.
- MIC, como se ha visto anteriormente, es el código de integridad del mensaje que permite autentificar el mensaje.

4.1.3.5. Adaptative Data Rate (ADR) y parámetros regionales de LoRaWAN.

La tasa de bits, o Bit Rate, depende del ancho de banda y del factor de propagación (SF) y siguiendo la Ecuación 1, vista en el apartado 4.1.2.1. El factor de propagación (SF) y el ancho de banda están normalizados en los parámetros regionales de LoRaWAN, por lo que solo pueden tomar valores determinados.

La tasa de datos o Data Rate (DR) es la combinación del SF y el ancho de banda, por lo que este parámetro también está normalizado, pudiendo tomar distintos valores entre 0 y 6, en función del SF y el ancho de banda de la transmisión [19]. Para el estándar europeo EU868 de LoRaWAN, el DR toma los siguientes valores en función de la combinación del SF y el ancho de banda:

Tabla 2. DR en función del SF y el ancho de banda. [19]

Data Rate	Factor de propagación (SF)	Ancho de banda
DR 0	SF12	125 KHz
DR 1	SF11	125 KHz
DR 2	SF10	125 KHz
DR 3	SF9	125 KHz
DR 4	SF8	125 KHz
DR 5	SF7	125 KHz
DR 6	SF7	125 KHz

El DR, y por consiguiente el SF, también determinan el número máximo de bytes que se pueden transmitir en el Frame Payload (los N bytes representados en la Figura 22). En la siguiente figura se muestran el máximo número de bytes en función del DR:

Tabla 3. Tamaño del Frame Payload en función del DR. [19]

Data Rate	Tamaño Frame Payload
DR 0	51 bytes
DR 1	51 bytes
DR 2	51 bytes

Data Rate	Tamaño Frame Payload
DR 3	115 bytes
DR 4	242 bytes
DR 5	242 bytes
DR 6	242 bytes

Como se vio en el apartado 4.1.2.1, amentar el SF aumenta el rango de la transmisión a cambio de aumentar el consumo eléctrico y disminuir la tasa de bits. Con el fin de encontrar un valor de DR que minimice el consumo energético, maximice la capacidad de la red y se adapte a la distancia entre el Gateway y el dispositivo final, se emplea la velocidad de datos adaptativa, o Adaptative Data Rate (ADR). [22]

El ADR consiste en que el servidor de red LoRaWAN ajuste el SF y la potencia de transmisión. Para encontrar el mejor ajuste de estos parámetros, el servidor de red emplea el RSSI y el SNR (ver apartado 4.1.2.2), junto con una estimación de paquetes perdidos que obtiene empleando el Frame Counter. El objetivo es mantener el SNR y el RSSI por encima del mínimo aceptado más un margen que permita la conexión sin perdidas de paquetes. [19]

El algoritmo ADR no está definido por la especificación LoRaWAN, por lo que cada servidor emplea su propio algoritmo. Aunque siempre se sigue un esquema básico como el siguiente:

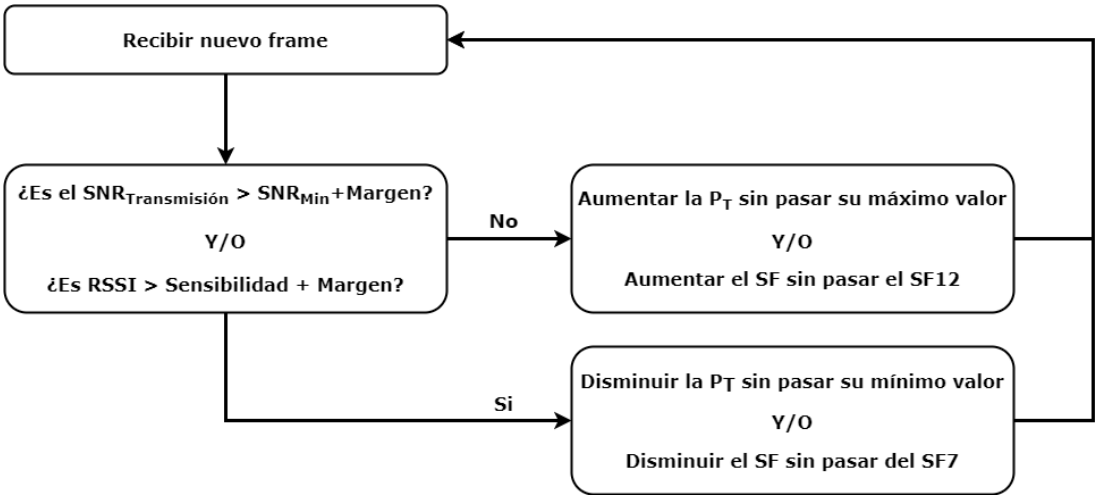

Figura 23. Esquema básico de algoritmo ADR. [19]

Las especificaciones de LoRaWAN varían en función de la región debido a las distintas regulaciones y normativas de cada país. En Europa existen dos estándares, el estándar EU868 y el EU433 haciendo referencia a la banda de frecuencia que emplean, mientras que Estados Unidos es únicamente el estándar US915. En la siguiente tabla se muestran algunos de los parámetros más importantes del estándar EU868 y US915:

Tabla 4. Parámetros importantes de los estándares EU868 y US915. [23], [25]

	EU868	**US915**
Banda de frecuencia	863-870 MHz	902 – 928 MHz
Canales	16	64 + 8 + 8
Ancho de banda uplink	125/250 kHz	125/500 kHz
Ancho de banda donwlink	125 kHz	500 kHz
Potencia de TX uplink	+16 dBm	+30 dBm
Potencia de TX donwlink	+16 dBm	+27 dBm
SF	7-12	7-10
Bit Rate	250 bps – 50 kbps	980 bps – 21.9 kbps
Link Budget uplink	155 dB	154 dB
Link Budget donwlink	155 dB	157 dB

4.1.4. The Things Network

The Things Network (TTN) es una plataforma abierta basada en la comunidad con el fin de crear una red global de IoT, de dispositivos y de soluciones LoRaWAN. [26]

The Things Network ejecuta el servidor The Things Stack Community Edition, se trata de un un servidor de red LoRaWAN de uso gratuito, abierto y descentralizado basado en la comunidad. [26], [27]

The Things Stack Community Edition se basa en The Things Stack y está respaldado por la comunidad de TTN, la comunidad mas grande de LoRaWAN a nivel mundial. Está pensado para ofrecer una forma fácil y gratuita de implementar soluciones LoRaWAN a los desarrolladores. Para proyectos comerciales se recomienda usar su versión de pago The Things Stack, respaldado por The Things Industries. [28]

The Things Network admite todas las clases (A, B, C) y versiones de LoRaWAN, así como todos los parámetros regionales definidos por LoRa Alliance. También admite funciones como el roadming pasivo, las actualizaciones de firmware por aire (FUOTA) o técnicas de agrupamiento y equilibrio de cargas. [28]

Ofrece una serie de integraciones basadas en APIs como gRPC, HTTP, MQTT o integraciones de almacenamiento con las que se facilita la gestión de los datos de los dispositivos. A pesar de que cuenta con todos los servidores necesarios de la arquitectura LoRaWAN (servidor de red, servidor de aplicación y/o servidor de unión), permite utilizar servidores externos, como un servidor de unión externo que almacene y emita las claves de seguridad con el fin de mejorar la seguridad de los datos de los dispositivos, o un servidor de aplicación externo en el que se descifren los mensajes transmitidos. [28]

4.1.4.1. Integración MQTT

Una de las integraciones que posee TTN es la de MQTT. Esta integración permite leer la información de los mensajes recibidos desde aplicaciones externas al suscribirse en su bróker MQTT.

Se trata de un protocolo de comunicación Machine to Machine (M2M) ligero y sencillo. Es del tipo "message queue", se trata de un servicio plublish-susbcribe (publicación – suscripción) en el que se genera una cola de mensajes para cada una de las suscripciones realizadas. Los mensajes publicados en la cola se mantienen hasta que son entregados a los distintos clientes suscritos.

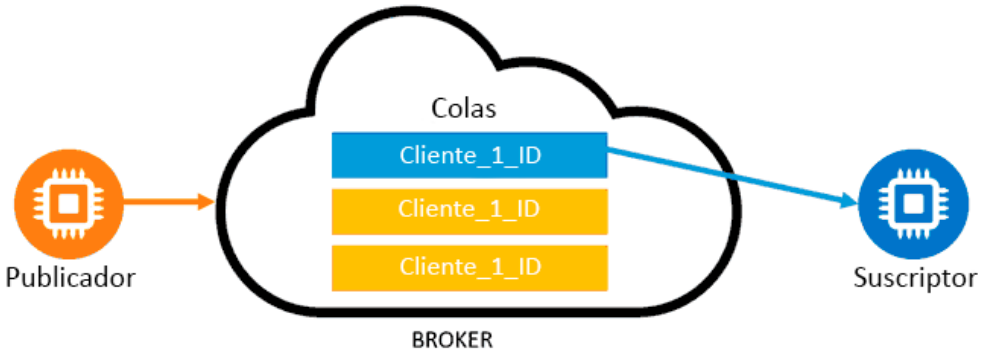

Figura 24. Sistema "Message Queue". [29]

Los clientes se conectan con un servidor central denominado bróker, el cual proporciona los servicios de MQTT, en este caso el bróker es el propio TTN. Los mensajes enviados a los clientes se organizan jerárquicamente en "topics". De esta forma un cliente puede publicar un mensaje en un determinado topic, y el servidor bróker enviará el mensaje a todos los clientes suscritos a dicho topic.

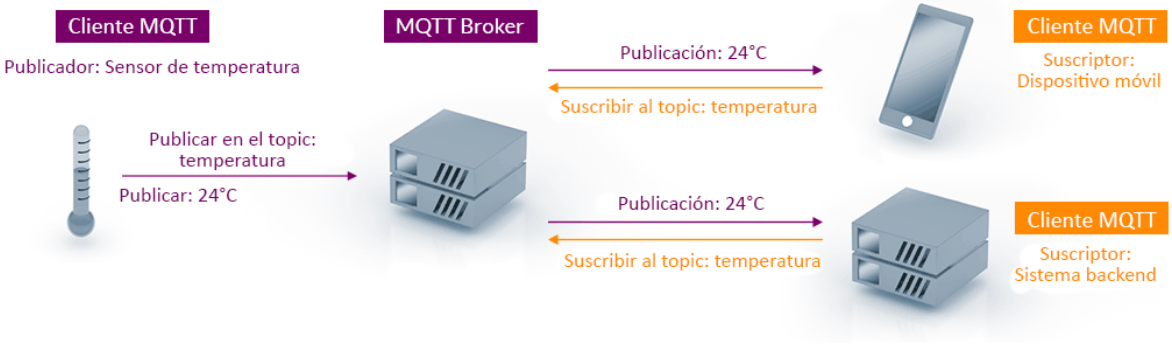

Figura 25. Arquitectura MQTT. [30]

Para realizar estas conexiones se emplean principalmente tres órdenes, connect, publish y subscribe. Connect contiene la información necesaria para la conexión, como el nombre de usuario, la contraseña… Publish se emplea para publicar mensajes en el topic, por lo que contiene el topic y el Payload. Suscribe se emplea para que el servidor envíe el mensaje del topic suscrito en cuanto haya alguno.

La implementación de MQTT en TTN se basa en un bróker MQTT proporcionado por la propia plataforma TTN. Cuando un dispositivo final envía un Uplink, este se publica automáticamente en un topic que sigue la forma de: v3/{application id}@{tenant id}/devices/{device id}/up. De esta forma una aplicación externa se puede suscribir a este topic y leer el JSON publicado, el cual contiene el mensaje desencriptado junto con información adicional de la conexión.

Para enviar un Downlink al dispositivo final se pueden emplear otros topic creados por TTN para este uso. En estos topic se puede publicar un JSON con el mensaje que se desea transmitir, una vez publicado, el servidor TTN se encargará de enviarlo en la siguiente ventana de tiempo de recepción disponible del dispositivo.

Tanto los mensajes leídos como los publicados se encuentran en formato JSON. JSON es un formato ligero de intercambio de datos compuesto por un conjunto de pares nombre/valor. Este formato tiene una sintaxis similar a un objeto JavaScript y puede almacenar los mismos tipos de datos, como cadenas, números, arreglos, booleanos…. Los objetos están delimitados por '{}' y los nombres de los valores están delimitados por "" y seguido por ':', mientras que los pares de nombre/valor están separados por ','. [31]

Como se ve en la Tabla 5, "rssi" es un nombre, mientras que '-57' es su valor asociado. En la Tabla 5 se muestra una parte del JSON publicado en el topic correspondiente al Uplink, donde se pueden ver algunos de los parámetros de la transmisión que pueden ser consultados a través de MQTT, como el RSSI ("rssi"), el SNR ("snr") o el Payload ("frm_payload") entre otros.

Tabla 5. Publicación MQTT de Uplink en formato JSON (Resumido).


```
{
  "uplink_message":{
    "f_port":2,
    "f_cnt":1,
    "frm_payload":"MjYuMzU7MDsw",
    "rx_metadata":[
      {
        "time":"2022-06-06T18:53:54.065Z",
        "timestamp":29934100,
        "rssi":-57,
        "channel_rssi":-57,
        "snr":6.2,
        "channel_index":2
      }
    ],
    "settings":{
      "data_rate":{
        "lora":{
```

```
            "bandwidth":125000,
            "spreading_factor":12
         }
      }
   }
}
```

Como se ve en la tabla, el Payload se encuentra en Base64. Se trata de un sistema de numeración posicional que contiene 64 caracteres ASCII básicos que se pueden emplear para codificar datos. En él los bits forman grupos de 6, creando las 64 posibles combinaciones, cada conjunto de bits corresponde a un carácter ASCII imprimible y básico, como se muestra en la Figura 26.

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
Padding		=									

Figura 26. Índice de Base64. [19]

4.1.4.2. Comunicación entre Gateway y TTN

Uno de los métodos que se puede emplear para la comunicación entre el Gateway y el servidor de red de TTN es el Packet Forwarder. El más famoso es el "Semtech UDP Packet Forwarder", por lo que muchas puertas de enlace incluyen una versión precompilada de este protocolo, lo que facilita su uso para realizar pruebas. [32]

Este protocolo intercambia los enlaces ascendentes, los descendentes y los estados en formato JSON, a través del protocolo UDP entre la puerta de enlace y el servidor de red. Este protocolo solo se recomienda para pruebas de desarrollo por su rápida integración, pero

cuenta con una serie de inconvenientes que lo convierten en una mala opción para aplicaciones comerciales. Alguno de estos inconvenientes son que no cuenta con autentificación para las puertas de enlace, o que el intercambio de los mensajes no es confiable puesto que se realiza a través de UDP. [33]

Para solucionar estos inconvenientes existen otros métodos para comunicar el Gateway con TTN, como el de "Basic Station". Este método soluciona los problemas del "Semtech UDP Packet Forwarder" y añade otras ventajas puesto que emplea el protocolo LNS en lugar de UDP, ofrece autentificación en el protocolo TLS, permite actualizar el software del Gateway y gestionarla, etc... [34]

4.1.5. Soluciones LoRaWAN para MCU

En este apartado se verán distintas soluciones LoRaWAN para los dispositivos finales, viendo sus posibles arquitecturas. Todos los dispositivos finales de LoRaWAN requieren principalmente dos partes:

- El software con el protocolo de LoRaWAN stack (pila de LoRaWAN) que permita ejecutar la secuencia de rutinas de la capa MAC del protocolo LoRaWAN con la que se realiza la transmisión de datos. Este protocolo LoRaWAN stack dependerá de la región para cumplir con las especificaciones. [19]
- Un transceptor LoRa que module la señal de radio frecuencia siguiendo los estándares de la capa física de LoRa. La antena se deberá diseñar específicamente para las frecuencias de cada región.

4.1.5.1. Arquitectura de transceptor más MCU

Este tipo de arquitectura es una de las más empleadas y consiste en un microcontrolador (MCU) y un transceptor LoRa encargado de modular o recibir la señal LoRa.

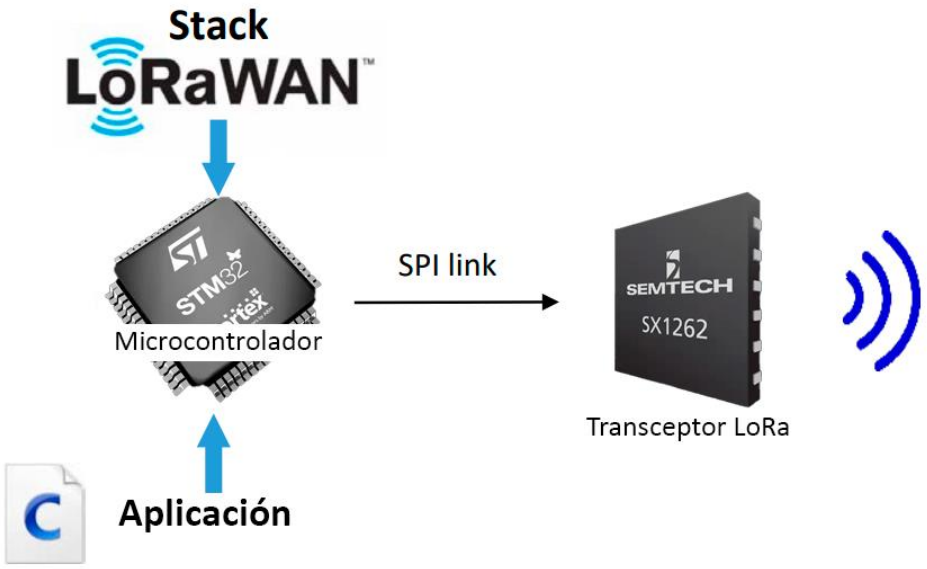

Figura 27. Arquitectura de microcontrolador más transceptor. [19]

El microcontrolador es el encargado de gestionar el protocolo LoRaWAN, implementándolo mediante el software de LoRaWAN stack. El microcontrolador también contendrá el código de la aplicación, mientras que el transceptor solo se encargará de la parte física de la conexión LoRa.

Uno de los transceptores más utilizados son los de Semtech, como el SX1262 u otros de la misma serie. En cuanto a los microcontroladores, existen numerosas opciones, desde un ESP32, o Arduino, hasta la familia de microcontroladores STM32. Esto se debe a que no se requiere ningún microcontrolador especial. La elección dependerá de la aplicación y los requisitos necesarios, como la potencia, consumo energético u otras características especiales.

Esta solución tiene cierta complejidad a la hora de implementar el software, ya que el código de la aplicación de usuario y el de LoRaWAN stack se ejecutan a la vez en el mismo microcontrolador. Por lo que se debe asegurar que una tarea no quita recursos a la otra. Por otro lado, el código se vuelve más complejo, por lo que hay que realizar una separación correcta entre la aplicación y la pila de LoRaWAN. [19]

4.1.5.2. Arquitectura de módulo LoRaWAN

En esta arquitectura se incluye el microcontrolador y el transceptor en un único módulo. Por lo que el diagrama de la arquitectura sería igual al de la Figura 27, pero en este caso se encuentran ya ensamblados en el mismo módulo, como el de la Figura 28.

Figura 28. Módulo LoRaWAN de ejemplo, modelo CMW1ZZABZ. [35]

Esto presenta la ventaja de que el hardware se simplifica, puesto que el módulo también cuenta con otros elementos como el circuito necesario para la antena. Existen multitud de módulos diferentes que combinan distintos microcontroladores y receptores, como en el módulo de Murata CMWX1ZZABZ mostrado en la Figura 28. Como se ve en el diagrama de la Figura 29, cuenta con el microcontrolador STM32L0 y el transceptor SX1276 de Semtech, junto con los elementos necesarios para el funcionamiento de la antena.

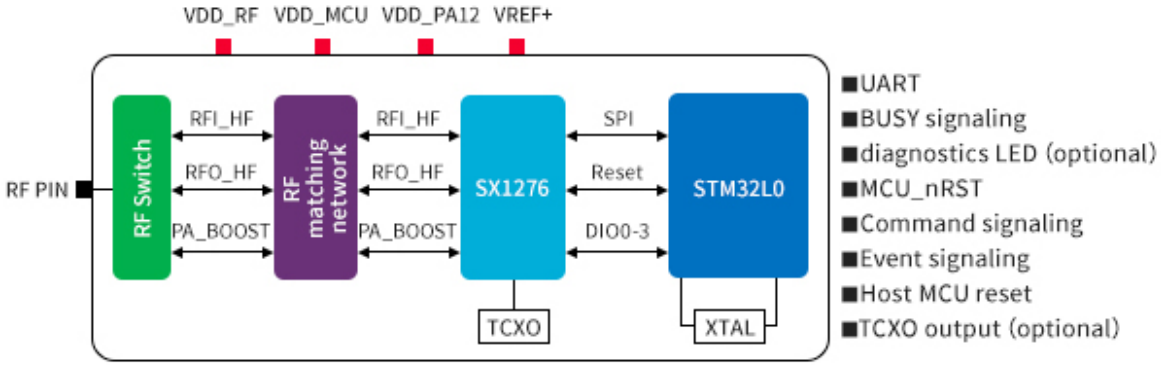

Figura 29. Diagrama de bloques del módulo CMW1ZZABZ. [35]

Por otro lado, mantiene la desventaja en la complejidad de la implementación y gestión del código de aplicación y el de LoRaWAN stack, como se vio en la arquitectura anterior. Por lo que desde el punto de vista del software no existen diferencias.

4.1.5.3. Arquitectura módulo LoRaWAN más MCU

Esta arquitectura soluciona la desventaja de la complejidad del software, puesto que el microcontrolador solo implementa el código de la aplicación. Mientras que el módulo LoRaWAN será el encargado de la gestión del protocolo LoRaWAN stack.

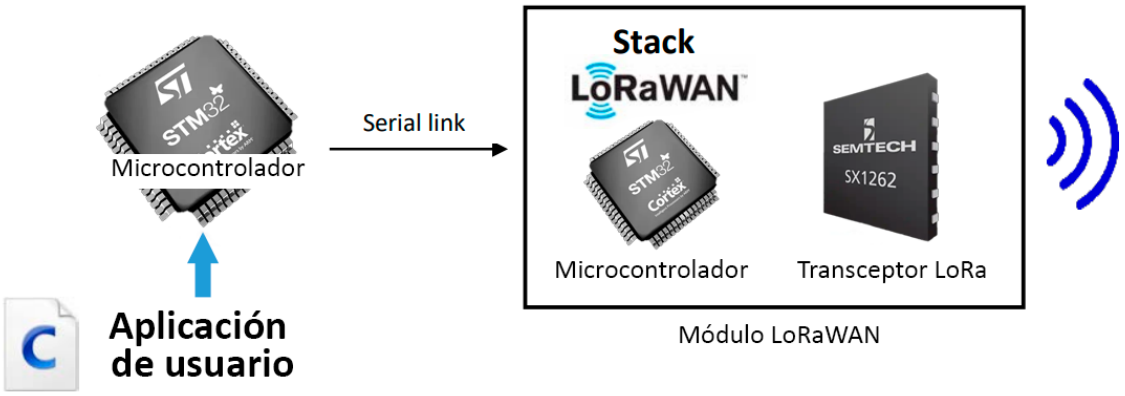

Figura 30. Arquitectura de microcontrolador más módulo LoRaWAN. [19]

El microcontrolador se comunica con el módulo mediante una conexión serial. Los módulos tienen la arquitectura vista en el anterior apartado (4.1.5.2). Por lo que se puede emplear cualquier modelo, como el del ejemplo anterior, el módulo Murata CMW1ZZABZ. La diferencia radica en el software implementado en el módulo, puesto que solo se implementará el software necesario para los protocolos de LoRaWAN stack, y serán controlados mediante comandos AT a través de la conexión serial.

En cuanto al microcontrolador, se podrá emplear cualquiera, en función de los requisitos de la aplicación. Esta vez el procesador tendrá menos carga ya que no tendrá que gestionar el protocolo LoRaWAN.

La principal ventaja de esta arquitectura es la simplicidad a la hora de gestionar el software. Pero a cambio se aumenta el consumo energético y el precio del sistema, ya que se requieren dos microcontroladores, el externo con el código de aplicación, y el interno del módulo, encargado de ejecutar la pila LoRaWAN. [19]

4.1.5.4. Arquitectura de MCU LoRaWAN

Esta arquitectura está desarrollada por la empresa STMicroelectronics, con su serie de microcontroladores STM32WL. Estos son los primeros en integrar el transceptor LoRa en el mismo chip. Esta familia de MCU se divide principalmente en dos categorías: procesadores de un único núcleo (STM32WLEx) o de doble núcleo (STM32WL5x).

Figura 31. Microcontroladore STM32WL. [36]

Cuentan con un transceptor sub-GHz basado en el Semtech SX126x. Admiten múltiples modulaciones como LoRa, (G)FSK, (G)MSK o BPSK. Esto otorga una mayor flexibilidad para crear distintas aplicaciones inalámbricas con LoRaWAN, Sigfox, W-MBUS u otros protocolos.

Los microcontroladores STM32WL cumplen con los requisitos de la capa física de la especificación LoRaWAN. Cuentan con una salida de potencia dual y un rango amplio de frecuencias en el rango de sub-GHz con el fin de garantizar la compatibilidad con las normativas de las distintas regiones. [36]

Esta es una solución muy adecuada en términos de coste, consumo energético y complejidad del hardware. En cuanto al software, se tiene el mismo problema que en las otras arquitecturas con un solo microcontrolador, el código de aplicación y el de la pila de LoRaWAN se gestionan en el mismo procesador, aunque en el caso de los procesadores con doble núcleo se puede separar en los distintos núcleos.

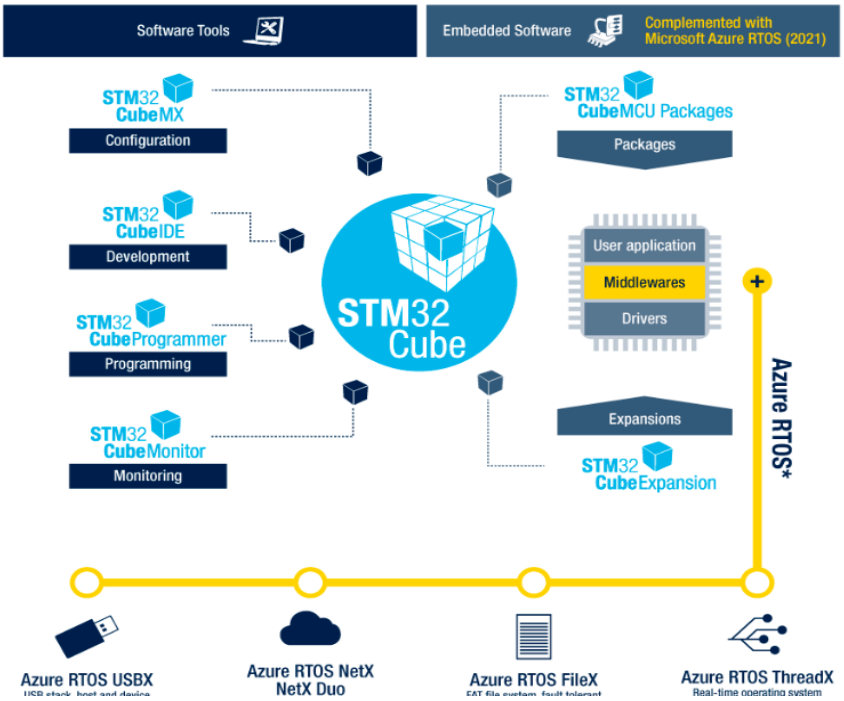

Figura 32. Ecosistema STM32Cube. [36]

Los microcontroladores STM32WL cuentan con el ecosistema STM32Cube, el cual consiste en un conjunto de herramientas de desarrollo que facilitan el desarrollo de la aplicación. Además, cuenta con soporte para la LoRaWAN stack, facilitando su implementación y reduciendo la complejidad a la hora de gestionar ambos softwares (LoRaWAN stack y el código de aplicación).

El soporte para la pila de radio LoRaWAN se logra mediante el paquete de STM32CubeWL, que cuenta con una serie de recursos de software integrados y controladores de periféricos HAL y LL. Este paquete también permite la compatibilidad con la herramienta STM32CubeMX. Esta herramienta permite configurar de forma gráfica los distintos parámetros del MCU, así como los parámetros de la conexión LoRaWAN. Después de realizar la configuración se encarga de generar automáticamente el código necesario para el MCU. [36]

En cuanto a la seguridad, la serie STM32WL cuenta con funciones de seguridad integradas, como el cifrado por hardware AES de 128/256 bits, protección de lectura y escritura PCROP y acelerador de clave pública.

Un ejemplo de MCU de la serie STM32WLE5 (single-core), es el STM32WLE5CC, este microprocesador está basado en un núcleo Arm de 32 bits Cortex -M4, con una frecuencia operativa de hasta 48MHz. En la Figura 33 se puede ver un diagrama con los componentes que integra.

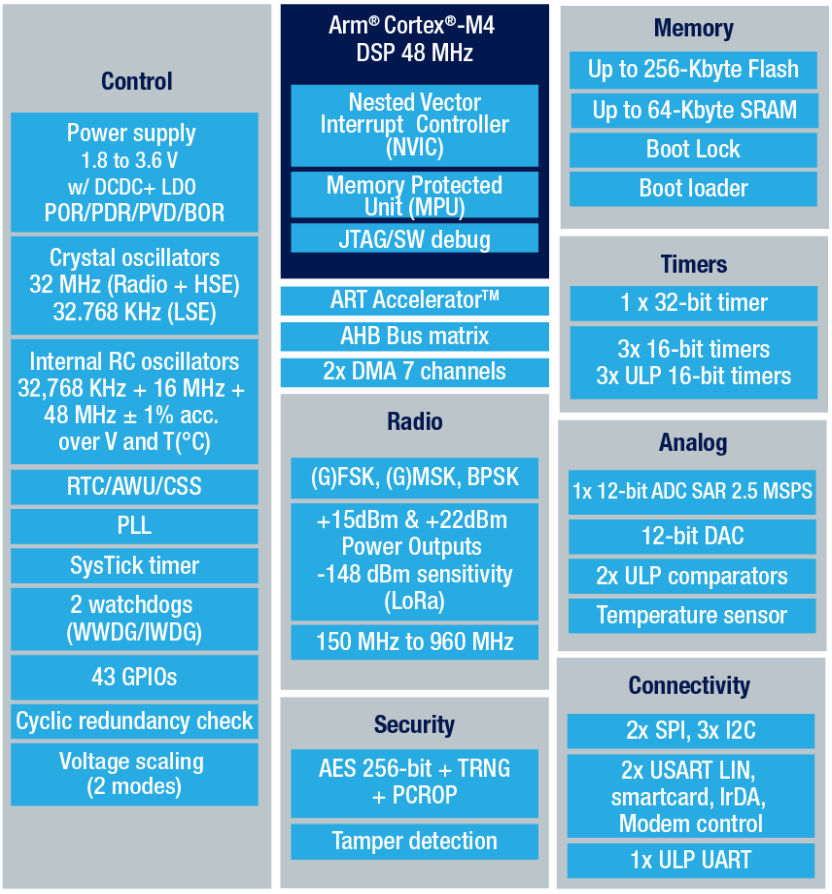

Figura 33. Diagrama del MCU STM32WLE5CC. [37]

Están diseñados para obtener consumos de energía muy bajos y cuentan con numerosas funciones y periféricos, como memorias de alta velocidad (Flash y SRAM) con funciones de protección de lectura y escritura, 43 GPIOs, conversores analógico-digital (ADC) y digital-analógico (DAC) de 12 bits de muestra y retención de baja potencia.

Incorporan un RTC (Real-time clock), reloj en tiempo real, de baja potencia con varios temporizadores de 16 y 32 bits, estos temporizadores son de gran utilidad para gestionar los tiempos de transmisión y recepción LoRaWAN y gestionar los modos de bajo consumo. También cuentan con dos controladores DMA de 7 canales que permiten la transferencia entre las memorias y los periféricos, como la radio de sub-GHz. [37]

4.1.6. Sensores de temperatura.

A la hora de diseñar dispositivos ponibles capaces de detectar la temperatura corporal se deben tener en cuenta varios factores, como el tamaño, el consumo energético, el coste del dispositivo, la precisión o la confiabilidad. Es por ello por lo que se debe tener en cuenta que tipo de sensor se ajusta mejor a la aplicación deseada.

Existen sensores de temperatura que no requieren contacto para medir la temperatura del cuerpo. Estos sensores funcionan mediante infrarrojos (IR), esto lo hacen midiendo la cantidad de energía infrarroja emitida por el cuerpo y su emisividad para determinar su temperatura. [38]

Existen varios tipos de sensores infrarrojos, como los puntuales, los sistemas de escaneo infrarrojos o las imágenes térmicas infrarrojas. Los dos últimos son los más complejos y costosos, siendo los puntuales los más empleados en para medir la temperatura corporal por infrarrojo. Estos a su vez se dividen en varios tipos en función del método empleado para medir la radiación infrarroja, los más comunes para dispositivos wearables son los sensores de termopila y los piroeléctricos.

Los sensores de termopilas convierten la energía térmica en energía eléctrica, para ello emplean varios termopares conectados normalmente en serie. Estos sensores miden la temperatura ambiente y la del objeto para realizar las compensaciones necesarias. [39]

Figura 34. Sensor de temperatura de termopila MLX90614. [39]

Por otro lado, están los sensores de temperatura de contacto. En esta categoría existen tipos de sensores, los dos más comunes en aplicaciones wearables de medición de temperatura corporal son los termistores NTC y los circuitos integrados de sensor de temperatura (IC). [40]

Los termistores de coeficiente de temperatura negativos (NTC) son sensores que miden la temperatura mediante el uso de una resistencia térmicamente sensible la cual reduce su resistencia cuando se aumenta la temperatura. Este sensor es no lineal fuera de su rango operativo, pero se pueden emplear circuitos sencillos que mejoran su linealización como el mostrado en la Figura 35. Son uno de los sensores de temperatura más precisos, con incertidumbres de medición de ±0,1 °C, ya que pequeños cambios de temperatura provocan grandes cambios en su resistencia. [38], [40], [41]

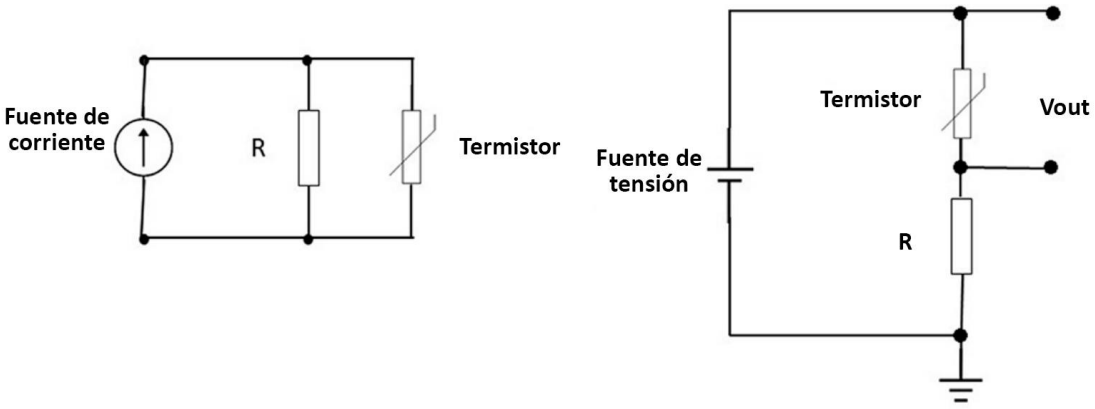

Figura 35. Circuitos de linealización de termistores NTC. [40]

Los sensores de temperatura IC, son sensores basados en semiconductores incorporados en circuitos integrados. Estos utilizan dos transistores, con una corriente de polarización directa constante que muestra una dependencia lineal de la temperatura. Estos sensores miden su temperatura local y pueden tener salidas analógicas o incluir conversores analógico-digitales (ADC) internos junto con el resto de elementos necesarios para procesar o almacenar los datos. [40], [42]

Por ejemplo, el sensor MAX30205, un sensor de temperatura IC digital. Este sensor almacena el ultimo valor de temperatura medido en

su memoria y se puede acceder a él mediante su interfaz I2C. Cumple con la norma de termometría clínica ASTM E1112, y cuenta con una precisión de 0,1 °C en el rango de 37°C a 39°C y una resolución de 16 bits (0,00390625 °C). Su circuito integrado se alimenta a una tensión de 2,7V a 3,3V y un consumo de 600 µA cuando convierte el valor de la temperatura mediante su ADC.

4.1.7. Interfaz I2C

La abreviación I2C (Inter-Integrated Circuit) hace referencia al puerto y protocolo de comunicación desarrollado por Philips en 1982 para la comunicación interna de dispositivos, aunque se ha convertido en un estándar del mercado.

Una de sus ventajas es que emplea únicamente dos cables para el bus de comunicaciones, el cable de señal de reloj (SCL) y el de transmisión de datos (SDA). Este bus es bidireccional y permite la conexión de múltiples dispositivos, pudiendo llegar hasta los 128. [43]

Se basa en una arquitectura maestro/esclavo (master/slave), en la cual el maestro se puede conectar a varios esclavos, se encarga de generar la señal de reloj y enviar la dirección del dispositivo con el que quiere comunicarse. Este funcionamiento mediante direcciones es lo que permite acceder a los distintos dispositivos de forma individual a pesar de que todos los dispositivos se encuentren conectados al mismo bus.

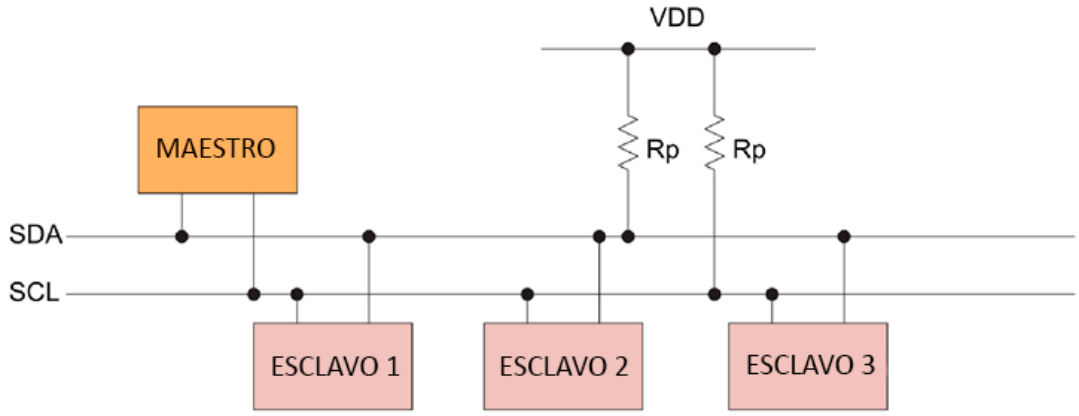

Figura 36. Esquema de conexión I2C. [43]

Al hacer uso de un único cable de transmisión, solo se puede transmitir en un sentido a la vez y de forma secuencial con cada esclavo. La conexión I2C necesita dos resistencias conectadas a la tensión de alimentación de los dispositivos (VDD), si la tensión de funcionamiento de alguno de los dispositivos es diferente, se pueden realizar adaptaciones de tensión.

Para elegir el valor de las resistencias se debe tener en cuenta que un valor alto aumenta el tiempo de cambio de la señal de 0 a VDD, puesto que existen capacitancias parasitas que hacen que el circuito se comporte

como una red RC, donde al variar la R varía el tiempo característico de respuesta (τ). También se debe tener en cuenta que disminuir el valor de la resistencia aumenta el consumo energético al disipar más energía debido a un mayor flujo de corriente.

En cuanto al software, el protocolo I2C establece que el inicio y el fin de las transmisiones solo las puede establecer el maestro mientras la señal del reloj está en alto. La transmisión de datos se realiza con la señal del reloj en alto, como se ve en la Figura 37 y Figura 38, mientras que los cambios de valores se realizan con el reloj en bajo. Además, cada byte transmitido posee un bit de confirmación de recepción o no recepción (ACK/NACK). [43]

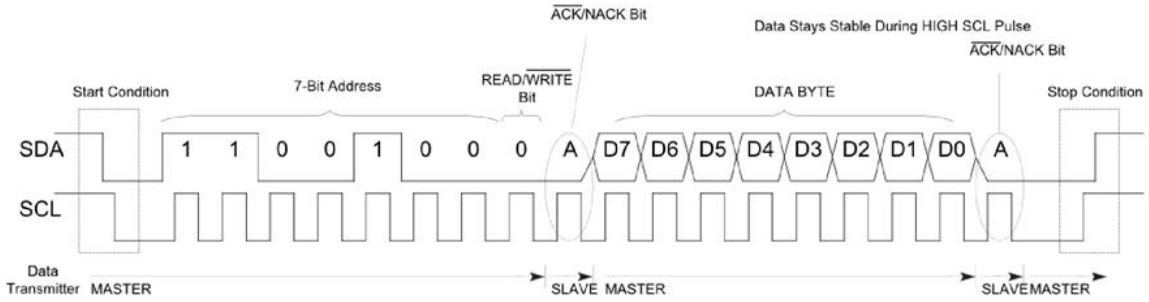

Figura 37. Escritura en esclavo. [43]

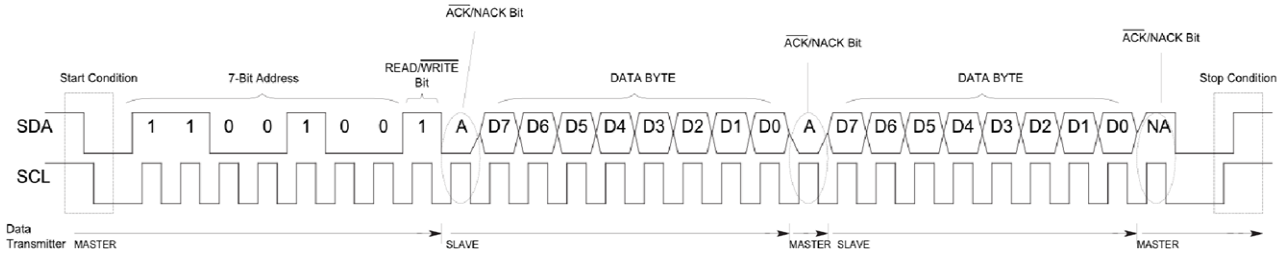

Figura 38. Lectura de dados de esclavo.

4.1.8. Método de medición de corriente eléctrica.

A la hora de medir la corriente eléctrica que fluye por un circuito cerrado existen varios métodos. Los más rápidos consisten en emplear directamente un amperímetro en serie con el circuito del cual se desea medir la corriente. También se puede emplear un amperímetro de pinza que contenga un núcleo de hierro toroidal y sensores de efecto Hall, de esta forma se puede medir el campo magnético creado por el conductor, el cual es proporcional a la corriente que atraviesa el conductor.

Por otro lado, existen métodos de medida de corriente de forma indirecta, como emplear una resistencia shunt o resistencia de derivación. Este método es uno de los más empleados a la hora de medir corrientes, muy grandes o pequeñas, que se encuentren fuera de la escala de medida del equipo. También resulta útil para medir corrientes con un

osciloscopio, puesto que solo se debe medir la caída de tensión de la resistencia.

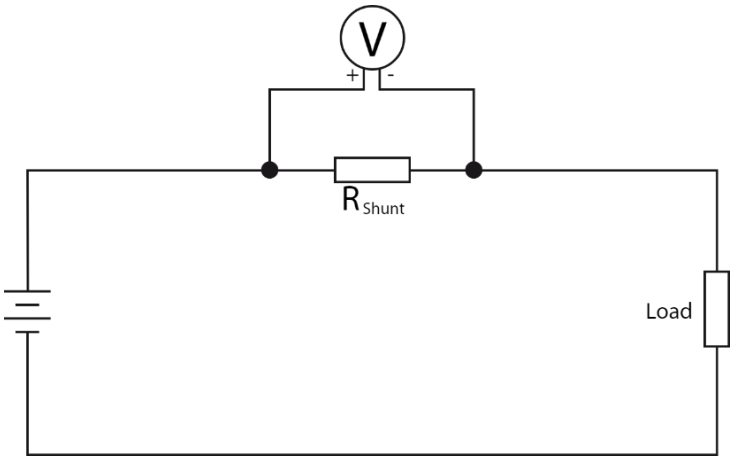

Figura 39. Esquema de conexión de resistencia de derivación o shunt. [44]

El método de la resistencia shunt o resistencia de derivación consiste en determinar la corriente midiendo la caída de tensión en una resistencia conectada en serie con el circuito, como se ve en la Figura 39. Para calcular la corriente se emplea la ley de Ohm:

$$I = \frac{V}{R}$$

Sustituyendo V por el voltaje medido y R por el valor de la resistencia conocida. Para minimizar la perdida de potencia en la resistencia, se deben elegir una resistencia shunt de bajo valor resistivo. Aunque al disminuir el valor de la resistencia disminuye la sensibilidad de la medición, por lo que se debe llegar a una solución de compromiso que consiga una caída de tensión significativa sin afectar al rendimiento del circuito. [45]

La mayoría de los osciloscopios ofrecen funciones para escalar el voltaje medido en la resistencia a una corriente equivalente. Normalmente ofrecen la posibilidad de reescalar los datos verticales, permitiendo a los usuarios modificar las unidades (en este caso amperios) y el factor de escala en unidades/voltio. [45]

También existen otros métodos para medir corrientes empleando un osciloscopio, como el uso de un transformador de corriente que detecte el campo magnético generado en el conductor del circuito, empleando un núcleo de ferrita que cree una corriente inducida en el bobinado secundario. Esta corriente inducida será proporcional a la corriente del conductor y se convierte a tensión al pasar a través de un resistor. También se pueden emplear sondas específicas de osciloscopio para medir corriente.

4.2. DESARROLLO

Se plantea el desarrollo de un prototipo de un dispositivo IoT ponible (wearable) capaz de medir de forma periódica la temperatura corporal de una persona y transmitirla empleando la modulación LoRa y el estándar LoRaWAN a través de su infraestructura de red, para finalmente representarla en un dashboard junto con otros parámetros relevantes sobre la calidad de la transmisión, como el SNR y el RSSI.

Se debe conseguir un consumo energético bajo, puesto que el dispositivo wearable final se alimentará con baterías. Se trata de una característica fundamental en este tipo de dispositivos, por lo que se estudiará el consumo energético del prototipo y se comparará con los resultados del simulador de consumo proporcionado por el fabricante del MCU.

El prototipo también debe ser lo más cercano posible a un dispositivo wearable, por lo que se deben evitar los componentes electrónicos que contengan elementos que no son útiles en un posible diseño final, como núcleos de desarrollo complejos. De esta forma se mantiene el diseño simple del prototipo y con un esquema electrónico y un consumo más próximo al de un dispositivo ponible.

En cuanto al sensor de temperatura, este debe medir de forma precisa y exacta, puesto que se trata de un parámetro fisiológico importante del cuerpo humano, el cual puede dar información sobre una posible alteración en la termorregulación corporal que podría deberse a enfermedades, golpes de calor, ejercicio excesivo u otros factores.

4.2.1. Funcionamiento del sistema

En la Figura 41, se observa un diagrama de bloques del sistema. En el se pueden ver los distintos componentes de la arquitectura LoRaWAN desarrollada. El prototipo desarrollado corresponde con el nodo final en la arquitectura de red LoRaWAN. Este se conecta mediante el protocoló LoRaWAN al Gateway, y está formado por el MCU, el sensor y la fuente de alimentación. También se pueden ver elementos semitransparentes en el diagrama, estos elementos se emplean únicamente para el desarrollo del prototipo y se eliminan después de emplearlos.

El Gateway hace de puente entre el servidor de red y el prototipo, convirtiendo los paquetes LoRaWAN en paquetes UDP. El servidor de red empleado es el proporcionado por The Things Network. Esta plataforma también ofrece el servidor de aplicación y el bróker MQTT al que se conectará el cliente implementado en LabVIEW.

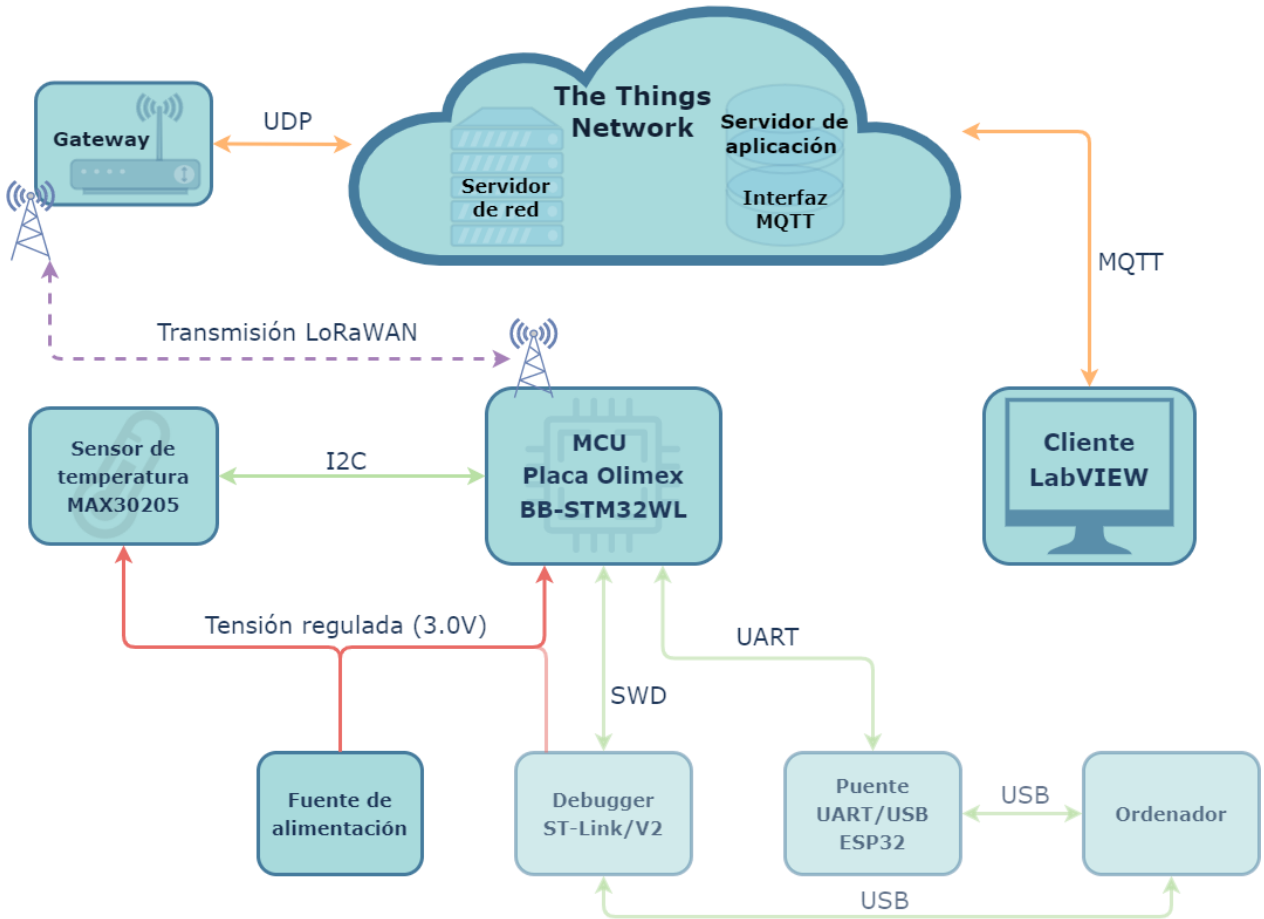

Figura 40. Diagrama de bloques del sistema.

Para medir la temperatura corporal se emplea el módulo Ferver Click [46], el cual cuenta con el sensor MAX30205. Este sensor cumple con las especificaciones de termometría clínica de la norma ASTM E1112. Cuenta con un modo de interrupción o comparación, en el cual el pin OS se activa al superar una cierta temperatura más su valor de histéresis, este funcionamiento similar al de un termostato con el fin de avisar cuando se detecta fiebre, aunque estos modos no se usarán en el prototipo.

Este sensor puede ser utilizado en dispositivos que funcionen con baterías gracias a su bajo consumo de funcionamiento, el cual tiene un valor típico de 600 µA según su data sheet [47], y a sus modos de un disparo (One-Shot) y apagado (Shutdown). Estos modos permiten medir la temperatura únicamente cuando se solicita, manteniendo el conversor analógico-digital (ADC) apagado el resto del tiempo. Por lo que se hará uso de estos modos para reducir el consumo de energía, ya que el dispositivo mide y envía la temperatura en intervalos amplios de tiempo. Este sensor enviará los datos a la MCU a través de la conexión I2C.

La MCU empleada es la STM32WLE5CC de STMicroelectronics (ver apartado 4.1.5.4). Con el objetivo de mantener el diseño del prototipo lo más cercano a un posible dispositivo wearable se descarta emplear el núcleo NUCLEO-WL55JC desarrollado por la misma compañía,

STMicroelectronics. En su lugar, se emplea la placa de Olimex BB-STM32WL [48], la cual cuenta con el STM32WLE5CC de un núcleo Arm de 32-bit Cortex-M4 con un transceptor LoRa integrado, una antena con los componentes electrónicos necesarios para su correcto funcionamiento con LoRa en un rango de frecuencias de 865 a 928 MHz, dos osciladores externos y un Led de usuario.

Esta MCU obtiene los datos del sensor a través de I2C y los envía de forma periódica al Gateway mediante LoRaWAN. Más adelante, en el apartado 4.2.2, se explicará de forma más detallada el diseño del software que se empleará en la MCU.

Para programar la MCU se emplea el depurador (debugger) y programador ST-LINK/V2 [49] a través de su conexión Serial Wire Debug (SWD) y el software STM32CubeIDE. Una vez programado, se desconectará el programador para realizar las mediciones del consumo de energía, para evitar fugas de corrientes que afecten al consumo del prototipo. Al desconectar el programador, el prototipo se alimentará con una fuente de alimentación con el fin de proporcionar una tensión estable de 3.0V durante la medida de la corriente.

También se utiliza un ESP32 conectado mediante UART a la MCU y al ordenador mediante USB, con el objetivo de establecer un puente que sirva como conexión serial con el ordenador para realizar pruebas y comprobaciones de funcionamiento, aunque esta conexión también se eliminará cuando se hayan realizado dichas pruebas y comprobaciones.

Para poder enviar los mensajes del prototipo a los servidores de red y aplicación, es necesario disponer de un Gateway cercano que pueda comunicarse mediante LoRa con el prototipo. Puesto que no se cuenta con ningún Gateway cercano que permita establecer una conexión, se implementará y se configurará uno de forma adecuada para su funcionamiento con el resto de la red LoRaWAN. En concreto el Gateway empleado es el MultiTech Conduit® IP67 Base Station [50], se trata de una puerta de enlace LoRa profesional que soporta las bandas de 868 MHz y 915 MHz. Está diseñada para ser instalada en exteriores, por lo que cuenta con protección IP67. Se conecta a internet a través de Ethernet y cuenta con el software mPower™ Edge Intelligence, el cual permite configurar los distintos parámetros del Gateway y su conexión a los servidores de red LoRaWAN.

Los servidores de red y de aplicación empleados son los proporcionados por The Things Network (TTN), el cual ejecuta The Things Stack Community Edition, una red LoRaWAN abierta y gratuita, como se vio en el apartado 4.1.4. Será en la consola de TTN donde se deberá registrar el Gateway y el prototipo, este proceso se explicará en detalle en el apartado 4.2.4. A la hora de registrar el prototipo, se debe proporcionar la AppKey, el AppEUI y el DevEUI para emplear el método de activación OTAA (ver apartado 4.1.3.2). Los mensajes enviados son desencriptados y mostrados en la consola de aplicación de TTN, junto con

otra información relevante de la transmisión como el RSSI y el SNR. Toda esta información se publica automáticamente como un JSON en un 'topic' de un servidor MQTT proporcionado por TTN, al cual nos podremos suscribir desde otro programa, para leer los distintos datos publicados.

Para leer y mostrar los datos publicados en el topic de MQTT se emplea el software LabVIEW. En el programa desarrollado en LabVIEW se implementa un cliente para la conexión al servidor MQTT y la suscripción al topic del que se leerán los datos de la transmisión publicados por TTN. Estos datos se encuentran en formato JSON, por lo que procesarán de forma adecuada para obtener los datos de interés como el RSSI, la SNR y el Payload, el cual se debe transformar ya que se encuentra en Base 64 (ver apartado 4.1.4.1). Por último, todos estos datos son mostrados en un panel (dashboard) de LabVIEW.

4.2.2. Implementación del nodo

Como se comentó anteriormente, se emplea la placa de Olimex BB-STM32WL, la cual cuenta con la MCU STM32WLE5CC. Para programarla se emplea el entorno de desarrollo integrado (IDE) STM32CubeIDE, el cual cuenta con el software STM32CubeMX, un configurador visual que genera el código necesario para el correcto funcionamiento del MCU en función del modelo de este y los parámetros configurados.

En este apartado se tratará toda la configuración y el código C necesario para el funcionamiento del prototipo. Se debe tener en cuenta que en el desarrollo se muestra directamente la configuración y programación que ofrecen un funcionamiento correcto y con los que se realizaron las pruebas de consumo mostradas en los resultados, sin mostrar el desarrollo de las pruebas realizadas para llegar a ellos.

A continuación, en la Figura 41, se muestra el diagrama UML principal que describe el comportamiento del software implementado mediante el uso de un secuenciador:

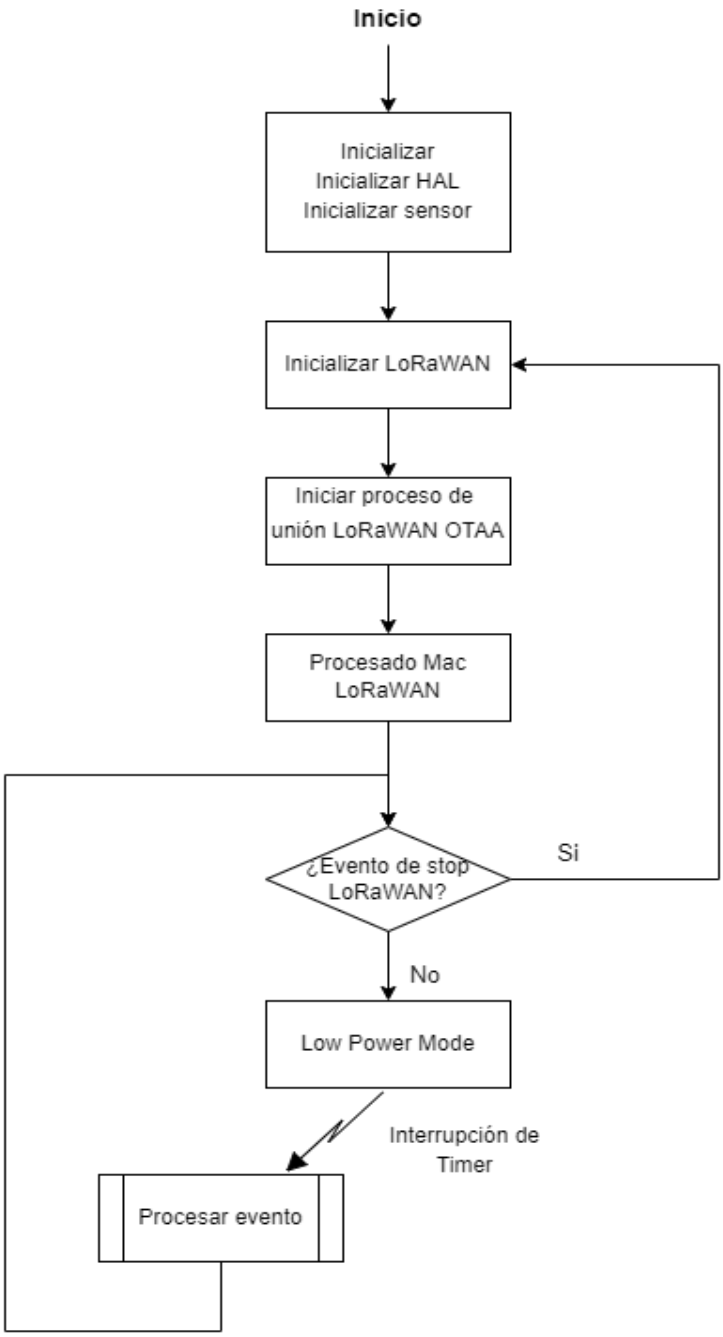

Figura 41. Diagrama UML principal.

Primero se inicializan los periféricos necesarios, así como el protocolo LoRaWAN. Al iniciar el dispositivo, después de las inicializaciones comienza el proceso de unión al servidor LoRaWAN. En este proceso se hace uso del secuenciador, el cual se encarga de administrar las tareas y los modos de bajo consumo.

El secuenciador inicia las tareas de Rx y Tx en función de las interrupciones del Timer. El resto del tiempo el secuenciador llevará al procesador a modo de bajo consumo.

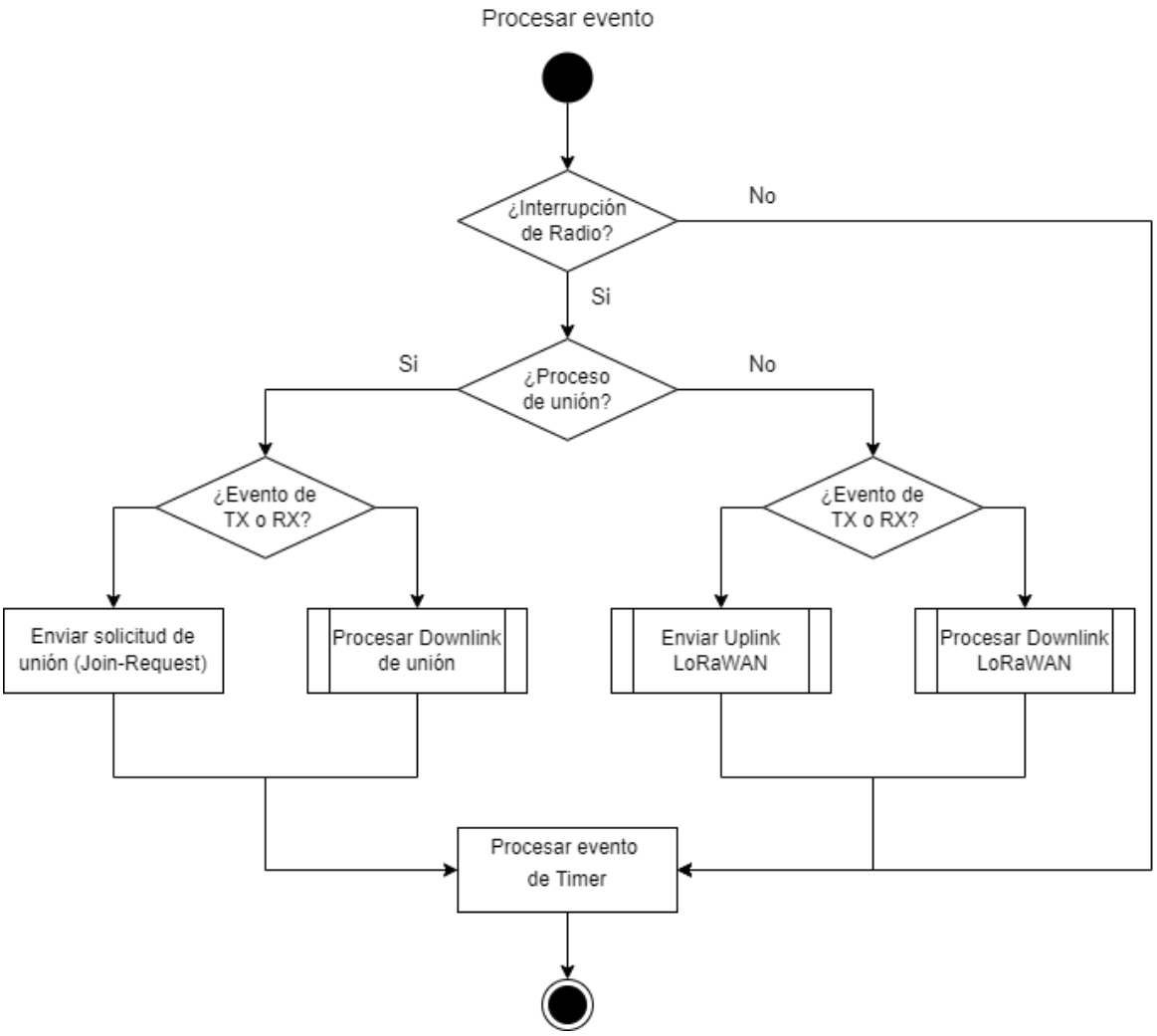

Figura 42. Diagrama UML del bloque 'Procesar evento'.

En la Figura 42, se ve el UML del bloque "Procesar evento". Cuando se produce una interrupción del Timer, se debe comprobar si también hay una interrupción de radio. Si la hay se debe enviar o recibir una transmisión, el tipo de la transmisión dependerá de si se encuentra en el proceso de unión o ya se ha unido.

Cuando se encuentra en el proceso de unión, se puede enviar el Uplink con la solicitud de unión o abrir las ventanas de recepción esperando el Downlink con la unión aceptada y el resto de información necesaria. Como se ve en la Figura 43, cuando la unión se acepta el programa pasará al modo normal de funcionamiento, donde se realizará la aplicación.

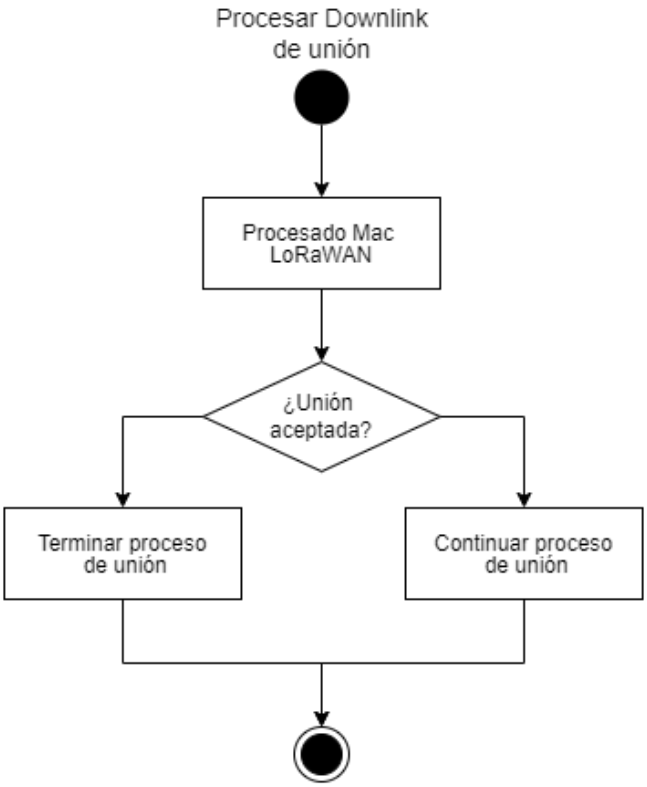

Figura 43. Diagrama UML del bloque 'Procesar Downlink de unión'.

En la Figura 44 se ven los bloques "Enviar Uplink LoRaWAN" y "Procesar Downlink LoRaWAN". Estos bloques pueden ser llamados después de que se acepte la unión al servidor. Cuando se recibe un Downlink se procesa la MAC del mensaje con la pila del protocolo LoRaWAN y se almacenan los valores de RSSI y SNR con los que se recibe el mensaje. Estos valores se enviarán en el próximo Uplink, junto con el valor de la temperatura que se medirá en el momento de la transmisión.

En el bloque de "Procesar Downlink LoRaWAN" se lee la temperatura del sensor. Para ello primero se envía la configuración One-Shot al sensor para que comience una conversión de temperatura, se esperan 60 ms para que la conversión termine y después se leerá la temperatura. Por último, se crea el mensaje con la temperatura leída y los datos de RSSI y SNR del mensaje recibido y se realiza el procesado MAC con la pila del protocolo LoRaWAN.

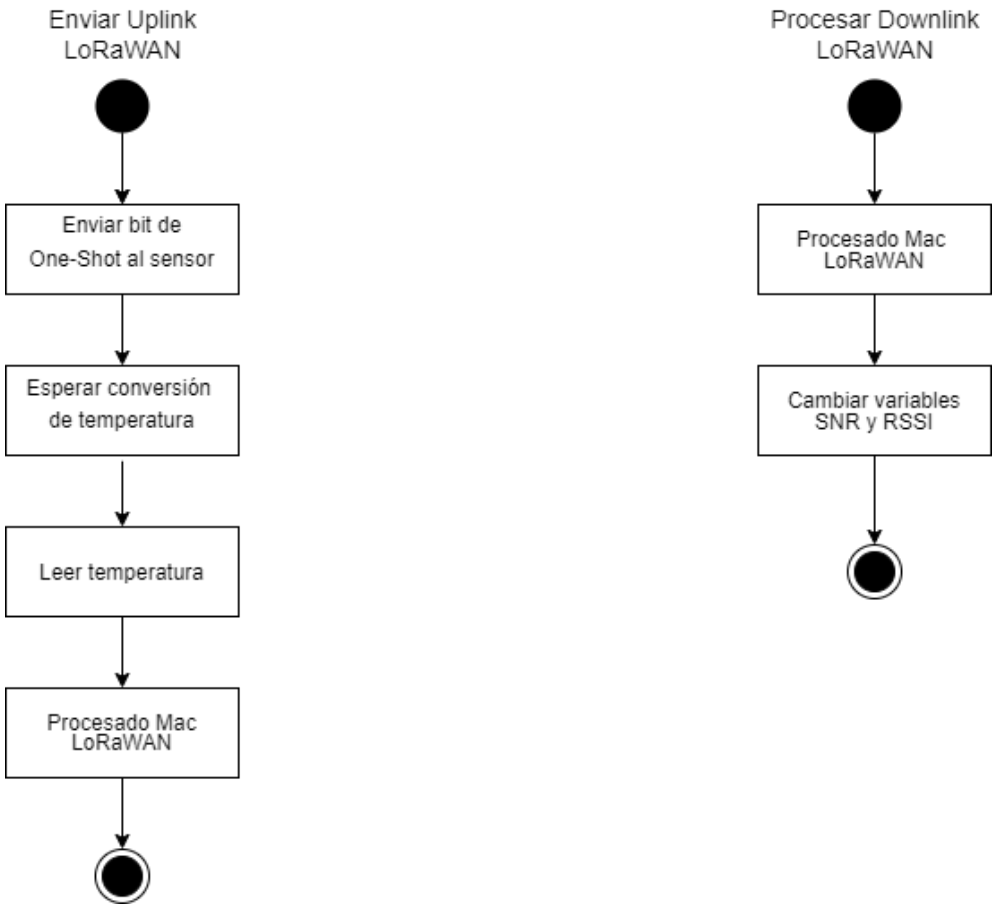

Figura 44. Diagrama UML de los bloques 'Enviar Uplink LoRaWAN' y 'Procesar Downlink LoRaWAN'.

4.2.2.1. Configuración en el STM32CubeMX

El primer paso es crear un nuevo proyecto en el IDE, al crearlo se abrirá el STM32CubeMX con una pestaña de selección en la que se seleccionará la MCU, núcleo o placa que se desea programar. Dado que la placa de Olimex BB-STM32WL no se encuentra en la selección, se selecciona la MCU que integra (STM32WLE5CC) y posteriormente se configurará para que funcione con dicha placa, siguiendo el esquema electrónico proporcionado por Olimex.

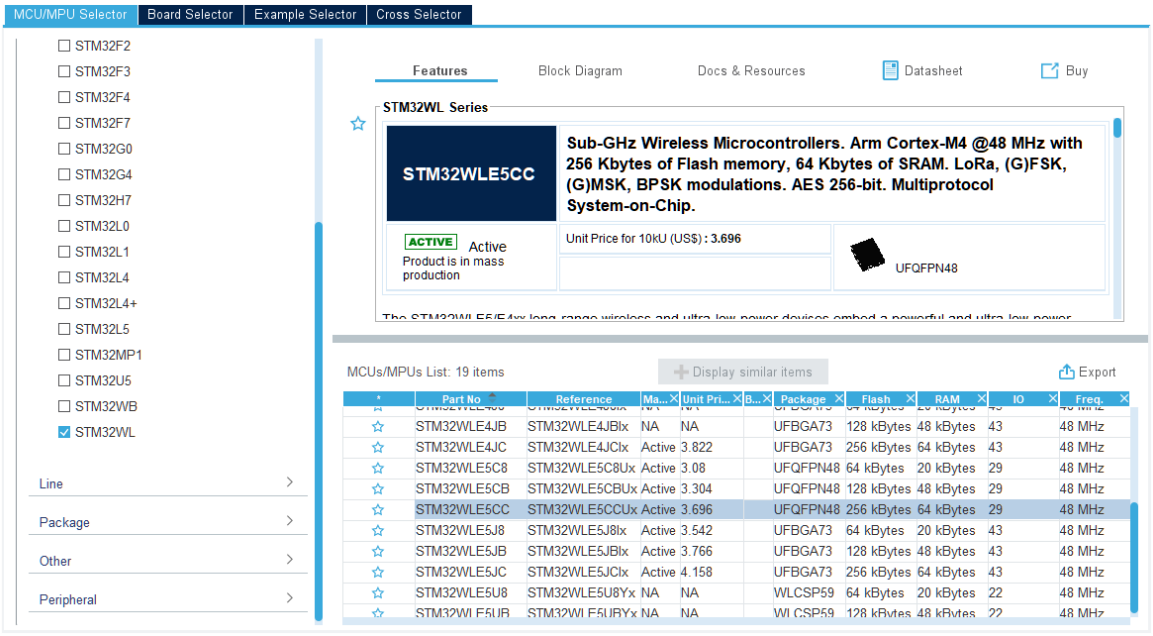

Figura 45. Seleccionar MCU en STM32CubeMX.

Se configuran los distintos pines de la MCU, para ello es necesario comprobar el esquema electrónico de la placa BB-STM32WL (ver anexos apartado 2). Los pines externos de la placa Olimex que se necesitan son los pines de la conexión I2C (I2C_SCL y I2C_SDA) para el sensor de temperatura y los pines para la conexión SWD con el debugger ST-Link V2. La configuración de los pines de la conexión UART para el ESP32 no será explicada puesto que en este apartado solo se explicará la configuración final del prototipo, evitando elementos que puedan aumentar el consumo de energía. Se buscan los pines externos que se usarán y los pines internos correspondientes de la MCU:

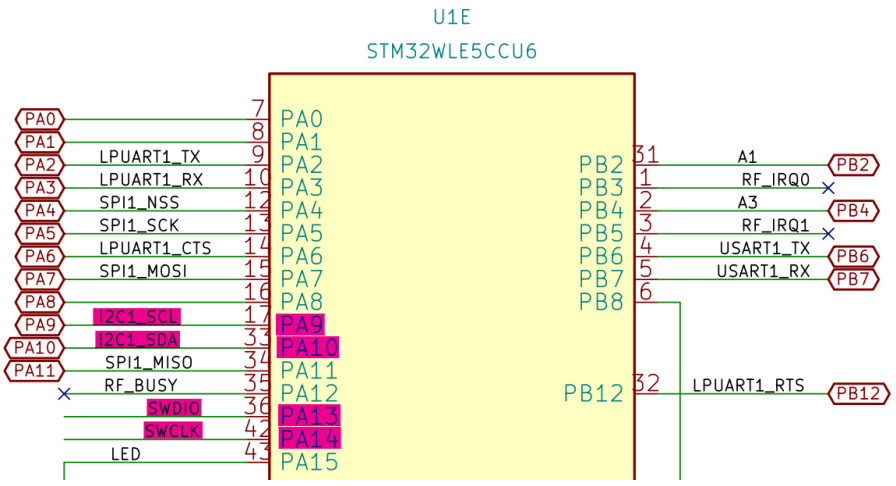

Figura 46. Esquema correspondencia entre los pines exteriores de la placa Olimex y los interiores de la MCU.

Una vez conocidos los pines correspondientes de la MCU, se configuran en el STM32CubeMX en el menú de conectividad habilitando el I2C y configurando sus pines en modo 'Pull-up'. También se selecciona

"Serial Wire" en la pestaña "Trace and Debug" para activar el debugger. Por último, se comprueba que los pines asignados son los correspondientes según el esquema.

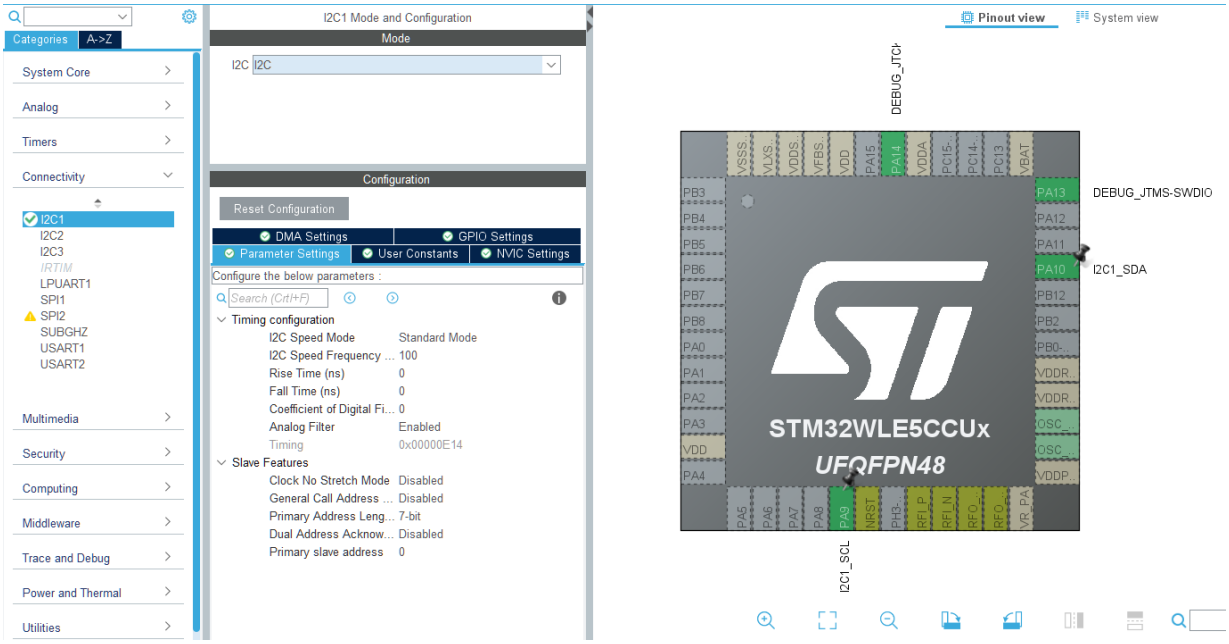

Figura 47. Configuración de los pines de comunicaciones exteriores.

Se configuran de la misma manera el resto de los pines de la MCU conectados a elementos internos de la placa Olimex, como los osciladores y el control del multiplexor de la antena entre la transmisión y recepción. El led, solo se emplea para la realización de pruebas, por lo que no se configurará.

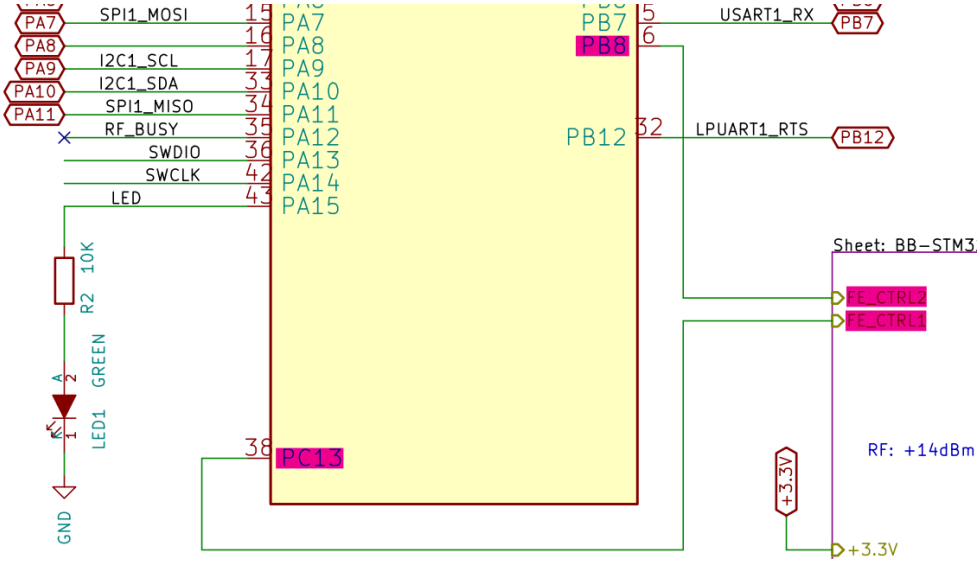

Figura 48. Esquema de los pines conectados a elementos internos de la placa.

Para los controles del multiplexor de la antena (FE_CTRL2 y FE_CTRL1) se establecen los pines correspondientes como salida (GPIO_Output) y se les asigna el nombre. Para los osciladores de cristal

externos, se activarán tanto el 'High Speed Slock' (HSE) como el 'Low Speed Clock' (LSE) en la sección de 'RCC Mode and Configuration'.

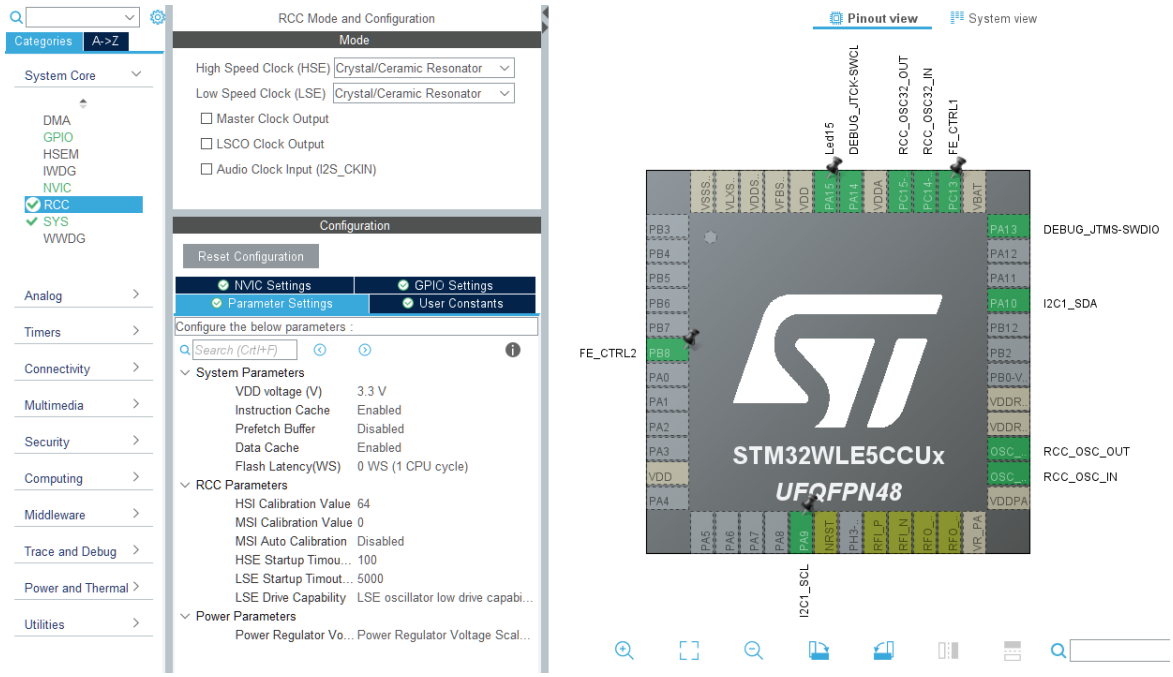

Figura 49. Configuración de los pines de los elementos internos.

Después de habilitar los pines de los osciladores de cristal se debe configurar sus frecuencias. Para ello se debe acceder a la pestaña 'Clock Configuration' en el STM32CubeMX y establecer las frecuencias de los osciladores en 32.768 kHz para el reloj de baja frecuencia (LSE) y en 32 MHz para el de alta frecuencia (HSE), y se seleccionan como fuente de frecuencia en el diagrama. También se usará un prescaler para dividir la frecuencia del reloj de alta frecuencia entre dos, por lo que la frecuencia de funcionamiento será de 16 MHz, como se ve en la siguiente figura:

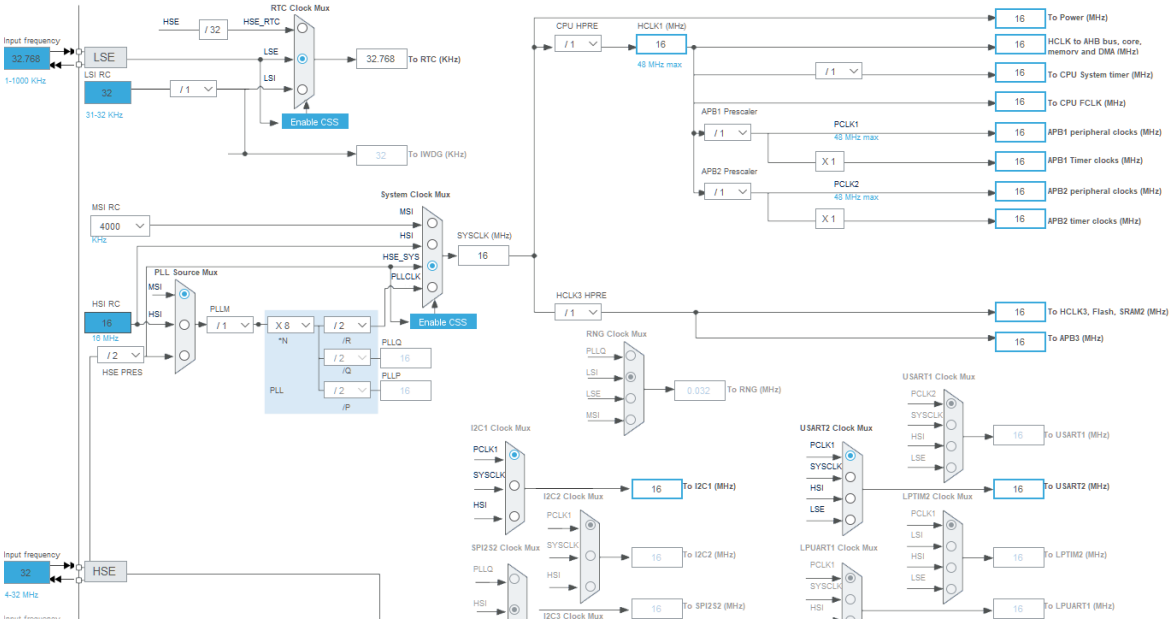

Figura 50. Configuración de la frecuencia de los osciladores externos.

Para configurar LoRaWAN, se debe activar la opción SUBGHZ en el apartado de 'Connectivity' y activar su interrupción 'SUBGHZ Radio Interrupt'. Esto habilitará el menú 'LORAWAN', en el cual se deberán configurar los distintos parámetros. En la pestaña 'LoRaWAN middleware' se seleccionará la frecuencia de la región europea 868, y el parámetro 'Select radio Driver' se cambiará a 'Bsp via extSettings'. Esto activará el controlador BSP (Board support package), permitiendo añadir un conjunto de funciones externas al agregar una serie de archivos en C al proyecto. Este conjunto de funciones administra los servicios de RF de la placa Olimex, como la configuración y el control del interruptor de RF, las distintas configuraciones de la placa o los leds y botones con los que se cuenta [51].

En la pestaña 'LoRaWAN application' se encuentran los parámetros de interés relacionados con LoRaWAN, como cada cuanto tiempo queremos enviar un mensaje (Transmition duty cycle), en este caso será de 10 segundos para realizar pruebas, este parámetro se deberá ajustar para conseguir un valor de compromiso entre el consumo y la frecuencia con la que se desea medir la temperatura. También se define el método de activación (en este caso se empleará el método OTAA), y la clase del dispositivo, la cual será clase A ya que nuestro principal objetivo es enviar datos, no es necesario una clase B o C puesto que aumentan el consumo energético (ver apartado 4.1.3.3). Otros parámetros de interés son el 'Activate Debugger' y 'Disable Low Power Mode' los cuales se deberán activar si se quieren utilizar las opciones de debug, pero deberán permanecer desactivados a la hora de medir el consumo energético puesto que evitan que el MCU entre en modo de ahorro de energía.

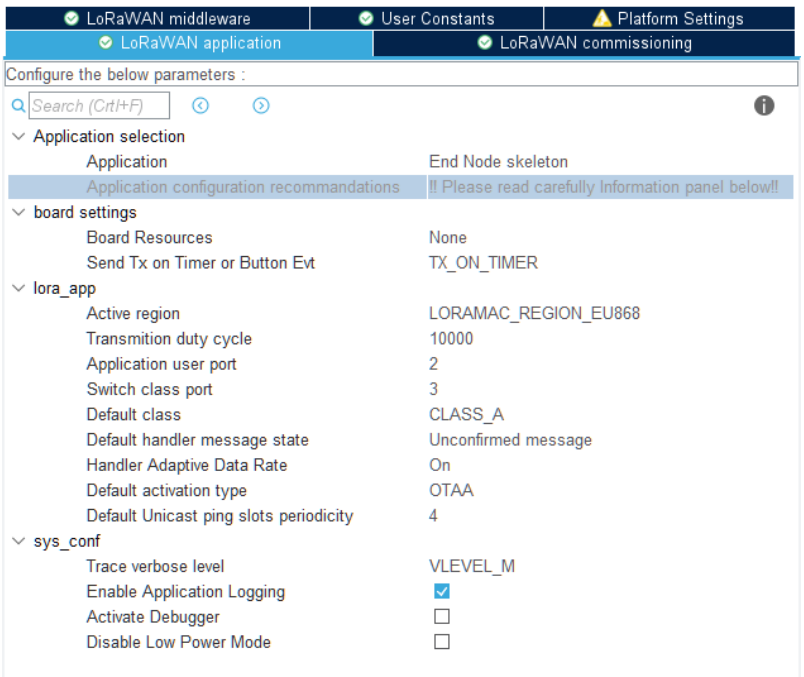

Figura 51. Configuración de LoRaWAN application.

Para terminar la configuración de LoRaWAN se deben seguir las recomendaciones de configuración de aplicación (Application configuration recommandations) que se proporcionan en esta misma pestaña. Por lo que se activa la opción 'Vrefint Channel' en la pestaña 'ADC'. Se activa también el USART2 en modo asíncrono y se añade el USART2_TX al canal 5 del DMA1, habilitando las interrupciones.

En la pestaña 'RTC' se activan las opciones 'Activate Clock Source' y 'Activate Calendar', se selecciona la alarma interna A, se añaden las constantes de usuario que se muestran en la Figura 52 y se activan las interrupciones. De esta forma se podrán crear las interrupciones que despertaran al MCU del modo de bajo consumo para poder abrir las distintas ventanas de recepción o transmisión [51].

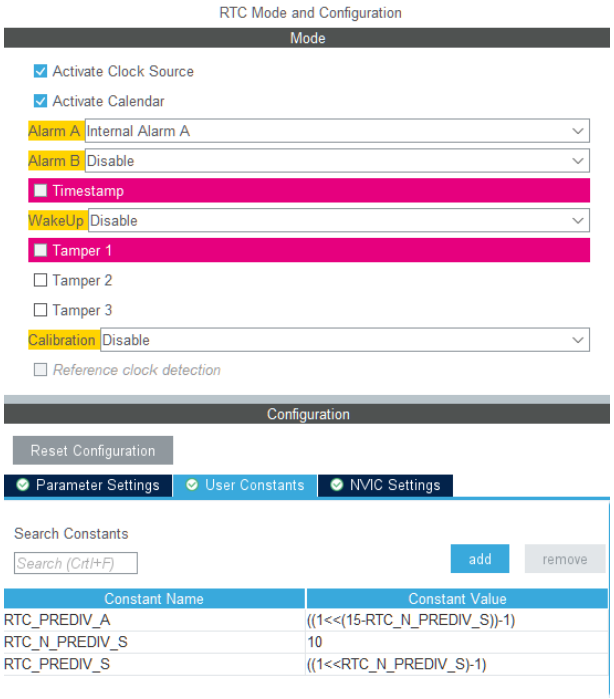

Figura 52. Configuración y constantes de usuario del RTC.

Por último y siguiendo con las recomendaciones de configuración de aplicación proporcionadas por el propio STM32CubeMX, en la pestaña 'Project Manager' > 'Advanced Settings' se desactivan todas las marcas de verificación de la columna 'Visivility (Static)' y se activan las de la columna 'Do Not Generate Function Call' como se muestra en la siguiente figura.

Generated Function Calls

Generate Code	Rank	Function Name	Peripheral Instanc...	Do Not Generate Function Cal	Visibility (Static)
☑	1	SystemClock_Co...	RCC	☐	☐
☑	2	MX_GPIO_Init	GPIO	☐	☐
☑	3	MX_I2C1_Init	I2C1	☐	☐
☑	4	MX_ADC_Init	ADC	☑	☐
☑	5	MX_RTC_Init	RTC	☑	☐
☑	6	MX_SUBGHZ_Init	SUBGHZ	☑	☐
☑	7	MX_USART2_UA...	USART2	☑	☐
☑	8	MX_LoRaWAN_Init	LORAWAN	☐	☐
☑	9	MX_DMA_Init	DMA	☑	☐

Figura 53. Configuración de las opciones avanzadas de gestor de proyecto.

4.2.2.2. Código C del programa en STM32CubeIDE

Después de la configuración anterior el STM32CubeMX autogenera el código necesario y se exporta al STM32CubeIDE. Todo el código que se quiera añadir al código autogenerado se deberá implementar entre los comentarios que se encuentran repartidos en él con el formato: /* USER CODE BEGIN X */ y /* USER CODE END X */. Si el código añadido se encuentra en un archivo distinto al de los códigos autogenerados no será necesario incluir estos comentarios ya que dichos archivos no se modificarán al regenerar el código.

4.2.2.2.1. Implementación del BSP de Olimex

En los pasos anteriores se activó el controlador BSP (Board support package), por lo que ahora se deben añadir los archivos necesarios para el funcionamiento con la placa Olimex. Para ello existen dos opciones, la primera consiste en descargar los archivos plantilla proporcionados por STMicroelectronics en GitHub [52] y modificarlos en función del esquema electrónico de Olimex. La segunda es descargar los archivos del BSP proporcionados directamente por Olimex en GitHub [53], será esta la opción que se realizará ya que es la más rápida y directa.

Para ello se deben copiar los archivos descargados al proyecto y cambiar el código autogenerado en "platform.h" para que incluya los archivos de cabecera con extensión .h "stm32wlxx_olimex.h" y "stm32wlxx_olimex_radio.h", en lugar de "stm32wlxx_nucleo.h" y "stm32wlxx_nucleo_radio.h". Puesto que no se debe cambiar directamente el código autogenerado, ya que los cambios se perderían al volver a generar el código con el STM32CubeMX si fuera necesario cambiar cualquier ajuste en él, se incluirá el siguiente código entre los comentarios destinados al código del usuario:

```
#define USE_BSP_DRIVER
/* USER CODE BEGIN EC */
```

```
#undef USE_BSP_DRIVER
/* USER CODE END EC */

/* Includes ------------------------------------------------------------
*/
#include <stdbool.h>
#include "stm32wlxx.h"
#include "main.h"
#include "stm32wlxx_ll_gpio.h"
#if defined(USE_BSP_DRIVER)
/* code generated by STM32CubeMX does not support BSP.  */
/* In order to use BSP, users can add the BSP files in the IDE project
space */
/* and define USE_BSP_DRIVER in the preprocessor definitions  */
#include "stm32wlxx_nucleo.h"
#include "stm32wlxx_nucleo_radio.h"
#endif /* defined(USE_BSP_DRIVER) */

/* USER CODE BEGIN include */
#define USE_BSP_DRIVER
#include "stm32wlxx_olimex.h"
#include "stm32wlxx_olimex_radio.h"
/* USER CODE END include */
```

Una vez hecho esto, se dispone de un código base para trabajar con LoRaWAN y adaptado a la placa Olimex. Este código cuenta con un bucle infinito en "main.c" donde se llama al secuenciador, el cual llama a la función 'SendTxData()' ubicada en "lora_app.c" cada vez que se activa el evento del Timer para la transmisión (este tiempo se definió anteriormente en la configuración de la aplicación LoRaWAN en STM32CubeMX mediante el parámetro 'Transmition duty cycle'). Cuando esto no ocurre, el secuenciador activará el modo de bajo consumo hasta que se detecte un evento de RTC. Por lo que el código donde se lee y envía la temperatura del sensor se implementará dentro de la función 'SendTxData()', de esta forma cada 10 segundo se enviarán los datos. A los 5 segundos después de terminar la transmisión se abrirá la ventana de recepción (RX 1), si no se recibe ninguna transmisión en esta primera ventana, se abrirá una segunda ventana (RX 2) un segundo después del comienzo de RX1.

4.2.2.2.2. Implementación del sensor MAX30205

El módulo Ferver Click tiene la dirección 0x90 como dirección I2C, la cual está asignada mediante hardware, por lo que no se puede cambiar o configurar. El sensor MAX30205 funciona internamente mediante cuatro subdirecciones a las que se acceden mediante un puntero de registros. En la siguiente figura se puede ver el nombre de cada registro con su correspondiente subdirección o valor del puntero de registro.

REGISTER NAME	ADDRESS (Hex)	POR STATE		POR STATE (°C)	READ/ WRITE
		Hex	BINARY		
Temperature	00	0000h	0000 0000 0000 0000	0	Read-only
Configuration	01	00h	0000 0000	—	R/W
T_{HYST}	02	4B00h	0100 1011 0000 0000	75	R/W
T_{OS}	03	5000h	0101 0000 0000 0000	80	R/W

Figura 54. Registros del sensor MAX30205. [47]

El registro de temperatura es solo de lectura, por lo que al acceder a él se devolverá el valor actual del registro. El resto de los registros son tanto de lectura como de escritura, por lo que se podrán cambiar sus valores. Para cambiar el valor de la temperatura de histéresis (T_{HYST}) o del OS (T_{OS}), se debe enviar el byte con la dirección del sensor, seguido del byte con la dirección del registro y por ultimo los dos bytes correspondientes al valor de la temperatura.

Los valores de las distintas temperaturas se almacenan en los correspondientes registros y tienen un tamaño de 16 bits. El formato de la temperatura se muestra en la Figura 55, mientras que en la Figura 56 se pueden ver algunos ejemplos de la temperatura correspondiente a cada conjunto de bytes. Por lo que para obtener el valor de la temperatura medida, se deberá acceder a su registro (0x00) y multiplicar del valor decimal de los dos bytes obtenidos por el valor del bit menos significativo (0.00390625 °C).

UPPER BYTE								LOWER BYTE							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
S	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	0.5°C	0.25°C	0.125°C	0.0625°C	0.03125°C	0.015625°C	0.0078125°C	0.00390625°C
	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}

Figura 55. Formato de los registros de temperatura. [47]

TEMPERATURE (°C)	NORMAL FORMAT		EXTENDED FORMAT	
	BINARY	Hex	BINARY	Hex
+64	0100 0000 0000 0000	4000h	0000 0000 0000 0000	0000h
+25	0001 1001 0000 0000	1900h	1101 1001 0000 0000	D900h
+0.5	0000 0000 1000 0000	0080h	1100 0000 1000 0000	C080h
0	0000 0000 0000 0000	0000h	1100 0000 0000 0000	C000h

Figura 56. Ejemplo de la temperatura correspondiente a un conjunto de bytes. [47]

En cuanto registro de configuración, está formado por un byte en el que cada bit corresponde a un parámetro de configuración diferente. En la Figura 57 se puede ver cómo está formado este byte. Para cambiar la configuración se debe enviar el byte con la dirección del dispositivo, con el byte del registro de configuración (0x01) seguido del byte con los valores de configuración que se necesiten.

D7	D6	D5	D4	D3	D2	D1	D0
ONE-SHOT	$\overline{\text{TIMEOUT}}$	DATA FORMAT	FAULT QUEUE [1]	FAULT QUEUE [0]	OS POLARITY	$\overline{\text{COMPARATOR}}$/ INTERRUPT	SHUTDOWN

Figura 57. Formato del registro de configuración. [47]

- Shutdown: Si su valor es 1 el ADC se apagará, por lo que el registro con el valor de la temperatura no se actualizará, en este estado el sensor consumirá menos de 3,5 µA según el datasheet.
- Comparator/Interrupt: Con este bit en 0 el pin OS se encontrará en modo comparador. En este modo el pin OS se activará al superar la temperatura de histéresis y se apagará cuando este por debajo de la histéresis. Cuando está en 1, funcionara como interrupción donde se activará al superar la temperatura de histéresis y no se desactivará hasta que haya alguna lectura de cualquier registro.
- OS Polarity: Cuando se encuentra en 0, el pin OS se activará en bajo. Si se encuentra en 1, el pin OS se activará en alto.
- Fault Queue: Estos dos bits determinan el número de fallos que deben de ocurrir para activar el pin OS, con el fin de evitar falsos disparos.
- Data Format: Si este bit se encuentra en 0, se usará el formato normal para los registros de temperatura. Si se encuentra en 1, se usará el formato extendido, ampliando la temperatura máxima de medida.
- TimeOut: 1 para deshabilitar el tiempo de espera del bus I2C. 0 para reiniciar el I2C cuando el SDA se encuentre en bajo por más de 50 ms.
- One-Shot: Cuando el sensor se encuentra en modo Shutdown, se puede usar el bit One-Shot para realizar mediciones puntuales cuando no es necesario medir la temperatura continuamente, lo que permite ahorrar energía. Cuando este bit se encuentra en 1, el ADC realizará una única conversión y actualizará el registro de temperatura con el valor obtenido. El tiempo máximo que llevará realizar esta conversión es de 50 ms según el datasheet [47]. Después de esto se volverá a apagar y el bit One-Shot regresará automáticamente a 0.

Para poder leer la temperatura del sensor MAX30205 conectado por I2C, deberemos utilizar una librería que facilite su uso. Puesto que no se ha encontrado una librería en C adecuada para el funcionamiento del sensor con el entorno de trabajo STM32CubeIDE, se modificarán y combinarán las librerías escritas en C++ de Maxim Integrated en Mbed [54] y de Protocentral en GitHub [55], con el fin de construir una librería en C adaptada a las funciones de I2C de la biblioteca HAL del STM32Cube.

El sensor se usará en el modo One-Shot, por lo que la función encargada de inicializar el sensor cargará en él la configuración necesaria para el funcionamiento de este modo. No se empleará el pin OS del sensor por lo que no se necesita configurar los parámetros relacionados con su uso. En esta función se envía el byte de configuración, donde el bit shutdown debe de encontrarse en 1, a la subdirección del sensor correspondiente a la configuración (0x01), como se ve en el siguiente código:

```
void MAX_begin(void){
      max_Configuration_u.bits.shutdown = 1;
      max_Configuration_u.bits.comp_int = 0;
      max_Configuration_u.bits.os_polarity = 0;
      max_Configuration_u.bits.fault_queue = 0;
      max_Configuration_u.bits.data_format = 0;
      max_Configuration_u.bits.timeout = 1;
      max_Configuration_u.bits.one_shot = 1;

      uint16_t local_config = (0x00FF & max_Configuration_u.all);
      MAX_writeRegister(MAX30205_CONFIGURATION, local_config);
}
```

El código completo de la librería se encuentra en los anexos, en el apartado 1.1. La función encargada de leer el valor de temperatura primero debe cambiar el bit 'One-Shot' de la configuración a 1 para que el sensor realice la lectura de la temperatura una única vez, después de esto el bit 'One-Shot' cambiará automáticamente a 0. Se deberá esperar a que el sensor termine la conversión de la temperatura, según el Data Sheet [47], este tiempo es de un máximo de 50 ms. Transcurrido este tiempo, más un pequeño margen, se leen los dos bytes correspondientes en la subdirección del sensor donde se almacena el valor obtenido por el conversor analógico digital, esta subdirección es la 0x00 según el Data Sheet [47]. Su valor se almacena en un entero de 16 bits y se multiplica por 0.00390625 para obtener la temperatura en grados centígrados, como se muestra a continuación en el código de la función:

```
float MAX_getTemperature(void){
      max_Configuration_u.bits.shutdown = 1;
      max_Configuration_u.bits.one_shot = 1;

      uint16_t local_config = (0x00FF & max_Configuration_u.all);
      MAX_writeRegister(MAX30205_CONFIGURATION, local_config);

      HAL_Delay(60);          //50 ms de conversión según datasheet

      uint16_t readRaw;
      MAX_readRegister(MAX30205_TEMPERATURE, &readRaw);
      MAX_temperature = readRaw * 0.00390625;
```

```
      return  MAX_temperature;
}
```

Ahora solo es necesario llamar a la función MAX_begin() en el 'main.c' y a la función MAX_getTemperature() en dentro de la función SendTxData() en 'lora_app.c' para obtener la temperatura antes de transmitir el mensaje.

4.2.2.2.3. Funciones de aplicación LoRaWAN

Para transmitir mediante LoRaWAN se empleará la function LmHandlerSend, a la cual se le pasa 4 parámetros, el más relevante es un puntero a una estructura tipo LmHandlerAppData_t que contiene el puerto de la aplicación, el buffer que contiene la información que se enviará y el tamaño de dicho buffer. Primero se debe declarar y definir dicha estructura, la cual se llamará AppData y contendrá un buffer tipo de uint8_t.

```
static uint8_t AppDataBuffer[LORAWAN_APP_DATA_BUFFER_MAX_SIZE];
static LmHandlerAppData_t AppData = { 0, 0, AppDataBuffer };
```

Después, dentro de la función SendTxData() se establecerá el valor del puerto de la aplicación, en este caso el puerto 2 el cual ya se definió con el STM32CubeMX. Mediante la función snprintf() se imprimirá una cadena de caracteres en código ASCII en el buffer de AppData con la temperatura obtenida de la función MAX_getTemperature() y el valor del RSSI y SNR del último mensaje recibido. A continuación se muestra el código implementado en la función SendTxData() para enviar la temperatura mediante LoRaWAN:

```
static void SendTxData(void)
{
  /* USER CODE BEGIN SendTxData_1 */

      float temperatura = 0;
      UTIL_TIMER_Time_t nextTxIn = 0;

      temperatura = MAX_getTemperature();

      AppData.Port = LORAWAN_USER_APP_PORT;
      AppData.BufferSize = snprintf((char *) AppData.Buffer,
LORAWAN_APP_DATA_BUFFER_MAX_SIZE, "%.2f;%d:%d", temperatura,
snr_recibido, rssi_recibido);
```

```
      LmHandlerSend(&AppData, LORAWAN_DEFAULT_CONFIRMED_MSG_STATE,
nextTxIn, false);

  /* USER CODE END SendTxData_1 */
}
```

Después de esto, solo queda implementar un código en la función OnRxData() que almacene los valores RSSI y SNR de la última transmisión recibida por el prototipo para ser enviados en la próxima transmisión al Gateway junto con la temperatura. Esta función es llamada cada vez que se recibe un mensaje pasándole como parámetros una estructura tipo LmHandlerAppData_t, como la empleada para enviar el mensaje, pero en este caso con los datos del mensaje recibido, y una estructura tipo LmHandlerRxParams_t que contiene información relacionada con la transmisión recibida, como el RSSI, el SNR o el Data Rate.

El código propuesto actualiza las variables snr_recibido y rssi_recibido con los valores correspondientes cuando se recibe una 'X' en un mensaje en el puerto de aplicación, para ser enviados posteriormente.

```
static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t
*params)
{
  /* USER CODE BEGIN OnRxData_1 */

      if(appData->Port == LORAWAN_USER_APP_PORT){
            if(appData->Buffer[0] == 'X'){
                  snr_recibido = params->Snr;
                  rssi_recibido = params->Rssi;
            }
      }

  /* USER CODE END OnRxData_1 */
}
```

4.2.2.3. Medida de consumo energético.

Primero se deberá subir el programa a la MCU con el código visto anteriormente. Para ello se conecta el debugger ST-LINK/V2 siguiendo el siguiente esquema creado a partir de su manual de usuario [56]:

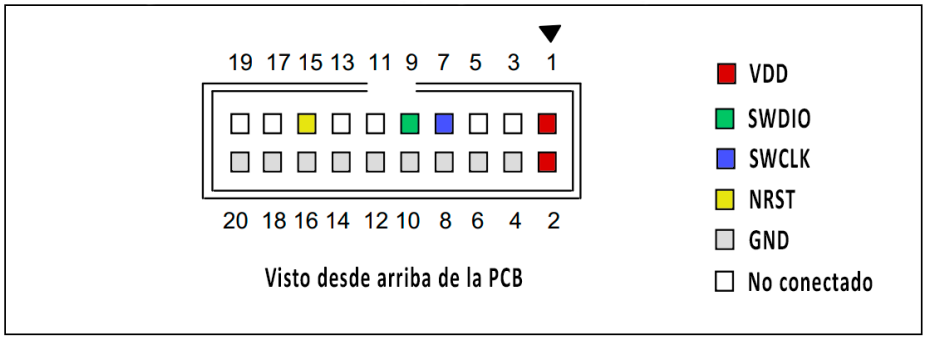

Figura 58. Esquema de conexiones de los pines del ST-LINK V2.

La conexión de la placa Olimex con el debugger y con el resto de los elementos, como el sensor de temperatura y el debugger, se realizará en una placa de pruebas (protoboard) como se muestra en la Figura 60. Al correr el programa en el STM32CubeIDE, aparecerá la configuración del proyecto, la cual se dejará en sus valores por defecto, comprobando que se utiliza el ST-LINK como debugger.

El sensor de temperatura MAX30205 está montado en el módulo Ferver Click, el cual cuenta con tres resistencias pull-up y un diodo led que se encuentra encendido siempre que se alimenta el módulo, como se muestra en el esquema de la Figura 59. Esto supone un consumo extra muy significativo, por lo que se retiraran todas estas resistencias del módulo, de esta forma el led permanecerá apagado. Retirar las resistencias pull-up no supondrá un problema para la conexión I2C, puesto que los pines del MCU se han configurado en modo pull-up.

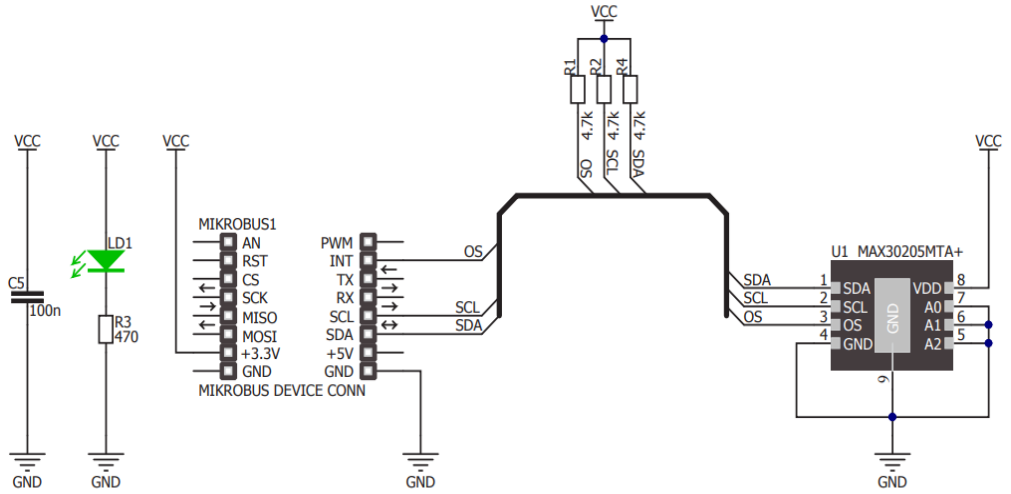

Figura 59. Esquema electrónico del módulo Ferver Click. [57]

Para poder medir el consumo del prototipo se empleará una resistencia shunt (ver apartado 4.1.8), de 10 Ω conectada en serie entre GND de la fuente el circuito. De esta forma se podrá calcular la intensidad de la corriente midiendo la caída de tensión en la resistencia shunt con el osciloscopio GW Instek MSO-2204EA y aplicando la ley de Ohm (I=V/R).

De esta forma se podrá configurar el osciloscopio en modo de medición de corriente, colocando la sonda en los extremos de la resistencia como se muestra en la Figura 60. En este modo el osciloscopio calcula directamente la corriente calibrando la sonda con una relación entre la caída de tensión y la corriente de 10V/1A, puesto que la resistencia es de 10 Ω.

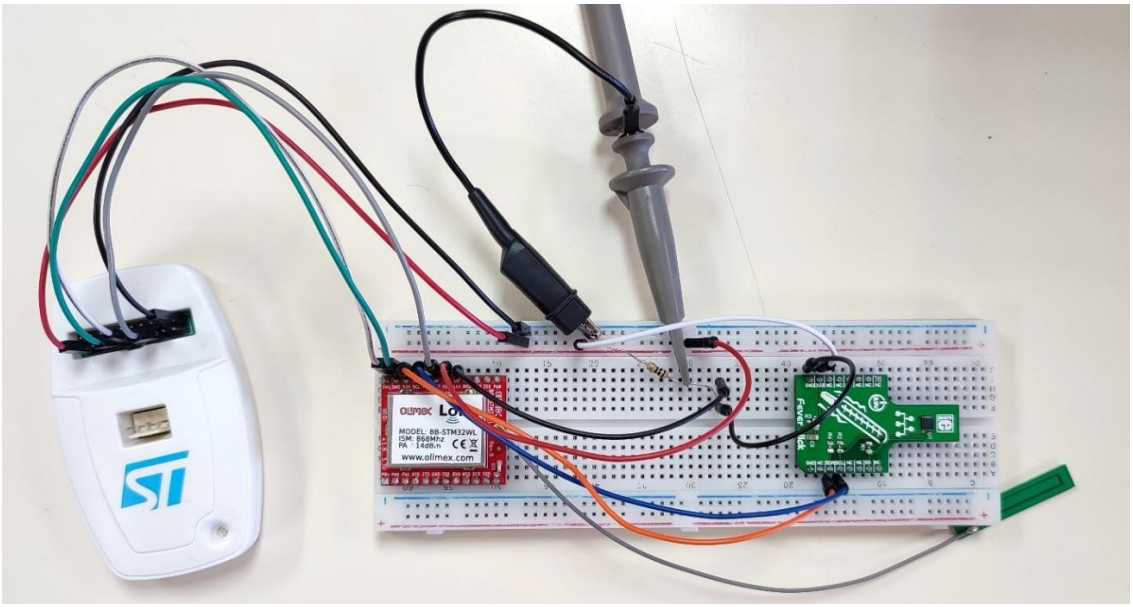

Figura 60. Montaje físico del prototipo en una placa de pruebas.

El principal inconveniente de este método es que el osciloscopio solo permite seleccionar escalas fijas para el uso de resistencias de valores como 1 Ω, 10 Ω, 100 Ω…, por lo que no se tiene en cuenta el valor real de la resistencia. En este caso al medir el valor real de la resistencia con el multímetro de mesa FLUKE 45 Dual Display Multimeter, se obtiene un valor de 10,5 Ω, por lo que se comete un error del 5% al suponer una resistencia de 10 Ω. Para reducir este error se deben corregir los valores de corriente medidos, para ellos se debe multiplicar por 10 (debido a la relación 10V / 1A del osciloscopio) y aplicar la ley de Ohm con el valor real de la resistencia:

$$V = I_{Osciloscipo} \cdot 10$$

$$I = \frac{V}{R} = \frac{V}{10{,}5} \rightarrow I = \frac{I_{Osciloscopio} \cdot 10}{10{,}5}$$

También se debe tener en cuenta que la escala del osciloscopio debe ajustarse lo mejor posible a la escala de la etapa medida, con el fin de reducir el error, puesto que, al aumentar la escala disminuyen la resolución y exactitud.

Por último, se realizará una media aritmética para calcular el consumo medio en cada una de las etapas y se corregirá su valor con la formula anterior para ajustarlo al valor real de la resistencia. Después se compararán estos valores con los consumos medios simulados.

Para simular los consumos del MCU STM32WLE5CC, se empleará el simulador proporcionado por el fabricante, disponible en el software STM32CubeMX. Se deberán configurar los distintos modos por los que pasa el MCU en cada ciclo, existiendo principalmente 5 modos distintos:

- Run más sensor: En este modo el MCU se encuentra en su estado normal, sin entrar en bajo consumo. Al mismo tiempo el sensor se encuentra realizando la conversión de temperatura, por lo que se deberá añadir el valor de la corriente del sensor de 600 µA.
- Run: Este modo se encuentra justo antes de comenzar la transmisión durante un breve periodo de tiempo, en el que el MCU se encuentra fuera del modo de ahorro de energía.
- Sleep: Este corresponde al modo de ahorro de energía en el que entra el MCU cuando se encuentra ocioso. En concreto entra en el modo de Sleep, uno de los modos de ahorro de energía menos profundos. La mayor parte del tiempo en el que el MCU está funcionando se encuentra en modo Sleep.
- Ventana Tx: El MCU se encuentra en modo Run al mismo tiempo que abre las ventanas de transmisión de un Uplink LoRaWAN. En este modo es cuando más se consume debido a la potencia de transmisión requerida y tiene una duración variable en función de los parámetros de la transmisión.
- Ventana Rx: El MCU se encuentra en modo Run al mismo tiempo que abre las ventanas de recepción de un Downlink LoRaWAN. Su duración dependerá de si se recibe o no un mensaje y de los parámetros de este.

Una vez conocidos los distintos modos, se deberán introducir en el simulador. Se debe tener en cuenta que la frecuencia de funcionamiento es de 16 MHz, y se alimenta a 3.0V. También se deberán marcar todos los periféricos empleados, en el caso de las ventanas de transmisión y recepción, se establecerá el estado adecuado del 'SUBGHZ' y se configurará la potencia de transmisión y la modulación empleada en los ajustes globales del simulador.

En el modo Run se añadirá el consumo adicional del sensor, como se ve en la Figura 61. Los tiempos en los que permanecerá cada modo se establecerán en función de los tiempos medidos en el osciloscopio, con el fin de poder comparar el consumo medio de todo el proceso, ya que este contiene modos que pueden tener distintas duraciones, como el caso de las ventanas de recepción y transmisión.

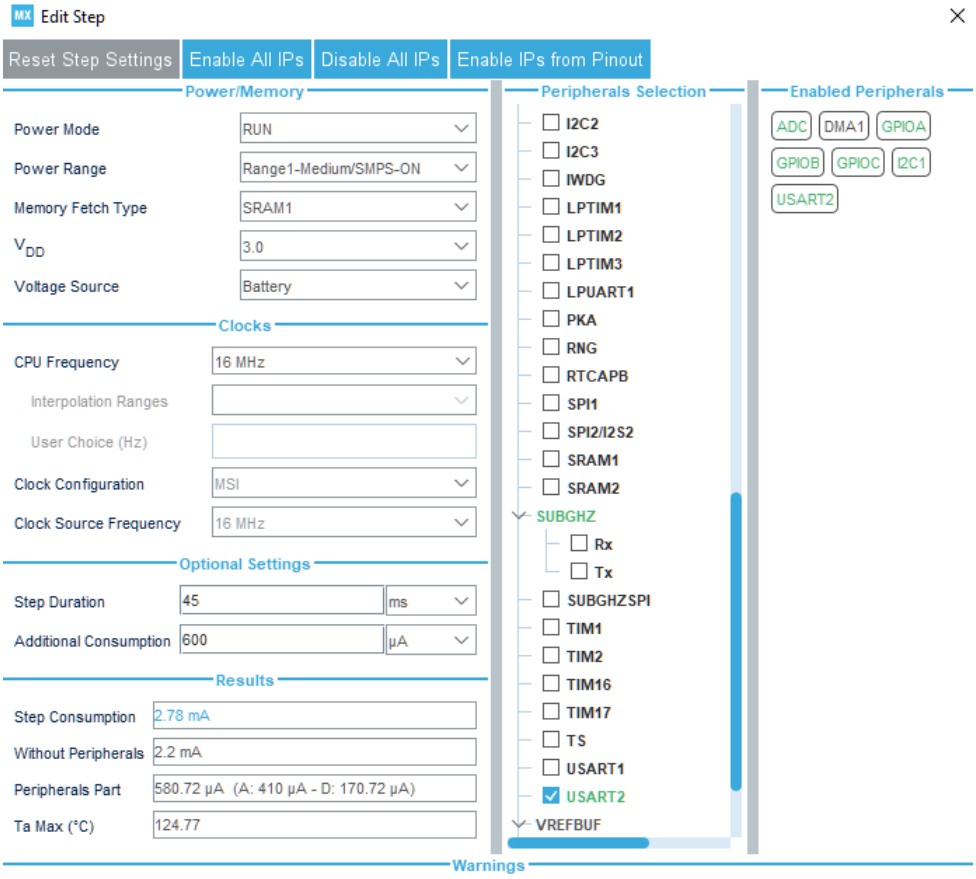

Figura 61. Configuración del modo Run en el simulador de corriente.

4.2.3. Configuración del Gateway MultiTech Conduit® IP67 Base Station

Como se dijo en el apartado 4.2.1, se usará una puerta de enlace profesional, en concreto el modelo MultiTech Conduit® IP67 Base Station [50] mostrado en la Figura 62, que cuenta con el software mPower™ Edge Intelligence Coundit para facilitar su configuración. Para poder acceder a este software y realizar una configuración básica del Gateway se debe conectar al ordenador mediante el cable de Ethernet y acceder a través de la dirección IP 192.168.2.1, por defecto.

Figura 62. Gateway MultiTech Conduit® IP67 Base Station.

Primero se pedirá establecer un nombre de usuario y contraseña, después de esto se abrirá un asistente de configuración inicial. En él se configurará la hora; la interfaz de red, que permanecerá en sus valores por defecto; la conexión a través de redes móviles, la cual no se usará ya que se conectará a Internet a través de Ethernet; y la gestión remota del Gateway mediante el servidor devicehq.com, que permanecerá deshabilitada puesto que no es necesaria.

Después de terminar con el asistente de configuración inicial se abre la interfaz principal, la cual cuenta con un menú en la parte izquierda. Puesto que solo se realizará una configuración básica que permita la conexión con los servidores LoRaWAN de The Things of Network (TTN), la mayor parte de los parámetros y funciones del Gateway permanecerán en sus valores por defecto. Se cambiará la interfaz de red 'eth0' en el apartado Setup -> Network interfaces, por la configuración mostrada en la siguiente imagen:

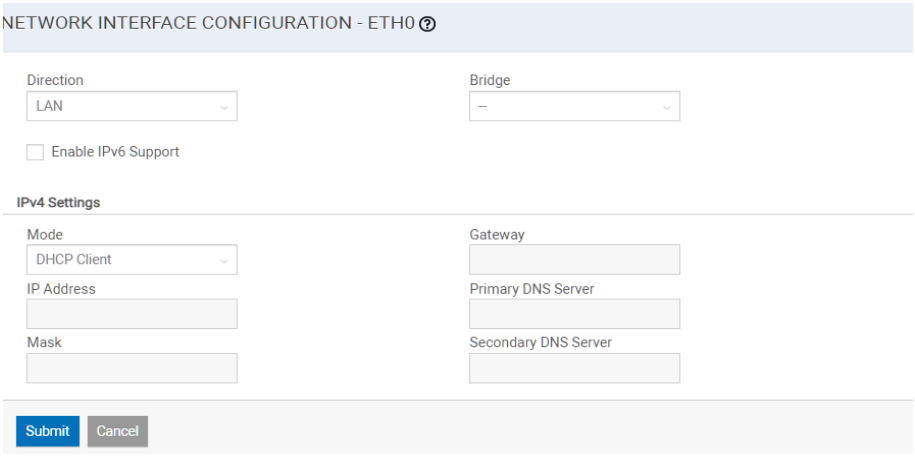

Figura 63. Configuración de la interfaz de red 'eth0'.

De esta forma el Gateway se podrá conectar a Internet a través de Ethernet, en lugar de usarse como puente para la configuración. Ahora para acceder a la configuración se conecta el Gateway a un rúter con el cable Ethernet y se entra a ella mediante la dirección IP local asignada automáticamente.

Para finalizar con la configuración del Gateway se debe ir al apartado LoRaWAN y activar el modo Packet Forwarder. En la configuración del Packet Forwader se configura el servidor al que se enviarán los paquetes, en este caso será el servidor de The Things Network con la dirección: eu1.cloud.thethings.network. La configuración será la que se muestra en la Figura 64, dejando el resto de los parámetros en su valor por defecto.

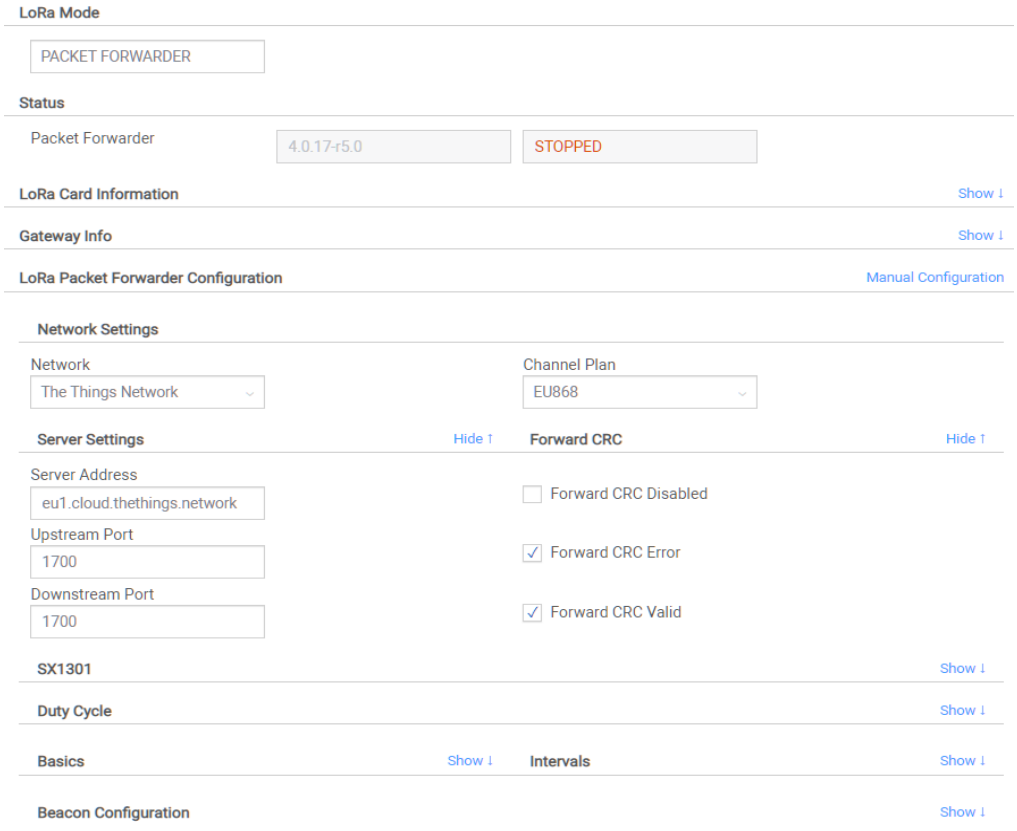

Figura 64. Configuración del modo Packet Forwarder.

4.2.4. Configuración de The Things Network

Los servidores de red y de aplicación LoRaWAN que se emplean en este proyecto son los proporcionados por The Things Network (TTN). Será en esta plataforma donde se registrará el Gateway y el prototipo desarrollado. Para poder hacerlo se debe tener una cuenta, con el plan gratuito es suficiente para añadir un Gateway, 10 dispositivos y una aplicación.

4.2.4.1. Registro del Gateway en TTN.

Primero se registrará el Gateway, añadiendo un nuevo Gateway en la consola en el servidor europeo. Para ello se introducen una serie de parámetros:

- El Gateway ID, el cual es un identificador único creado por el usuario y que puede contener letras minúsculas, números y guiones.
- El Gateway EUI, un identificador único de 64 bits proporcionado por el Gateway y que se obtiene en la sección LoRaWAN->LoRa Card Information del menú en el software de configuración del Gateway visto en el apartado anterior.
- El nombre y la descripción para el Gateway.
- 'Require authenticated connection', traducido como requiere conexión autentificada, esta opción permanecerá deshabilitada, puesto que si se habilita no se permitirá la conexión mediante Packet Forwarder.
- El plan de frecuencias, se seleccionará la banda 868 europea con un SF9 para la segunda ventada de Rx, 'Europe 863-870 MHz (SF9 for Rx2 - recommended).

El resto de los parámetros permanecerán en sus valores por defecto, como se ve en las siguientes figuras:

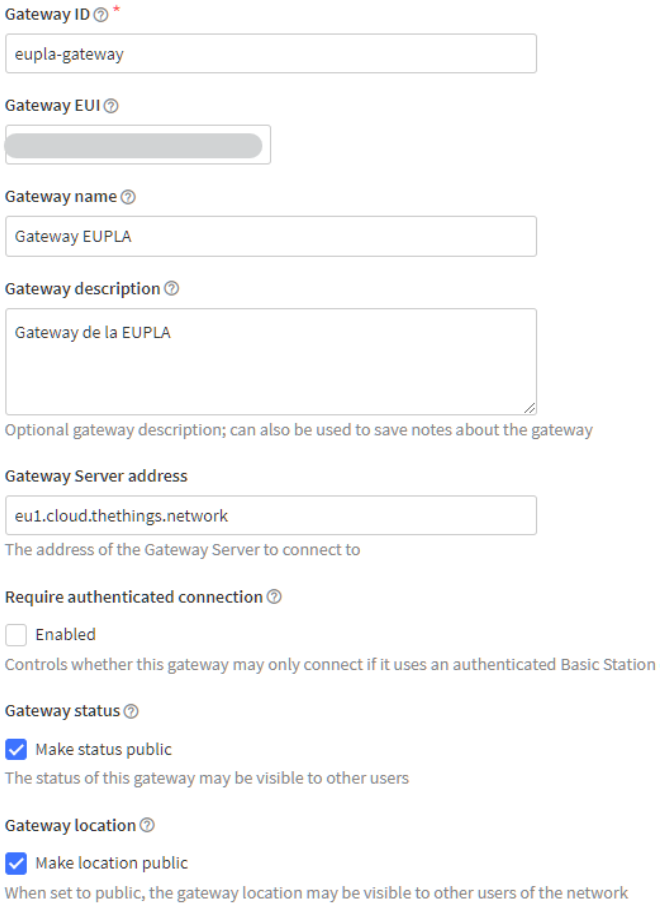

Figura 65. Registro del Gateway en TTN (1).

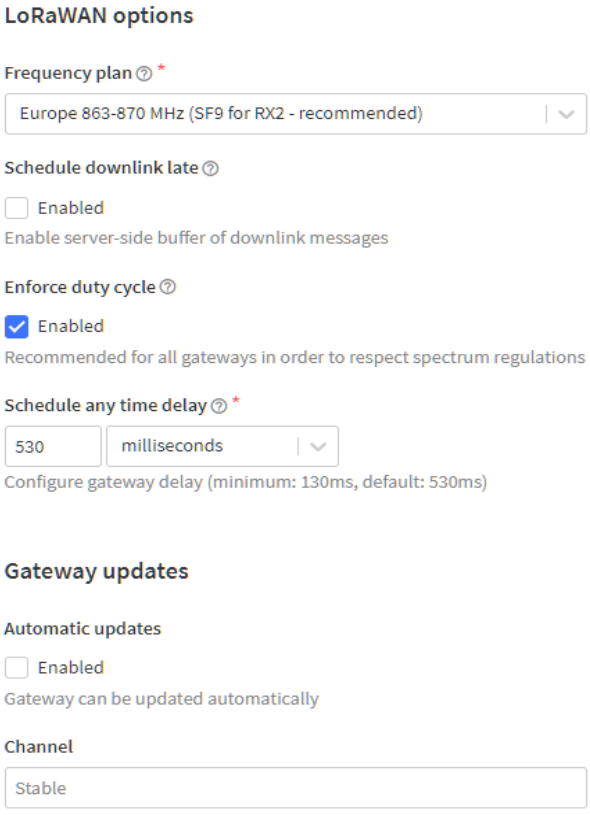

Figura 66. Registro del Gateway en TTN (2).

4.2.4.2. Registro del prototipo y creación de la aplicación en TTN

Una vez registrado el Gateway, se debe crear una aplicación en la misma consola del servidor europeo. Para ello se requiere introducir un nombre de aplicación, una descripción y un 'Application ID' o ID de aplicación, el cual es un identificador único, como el Gateway ID, creado por el usuario de la misma forma.

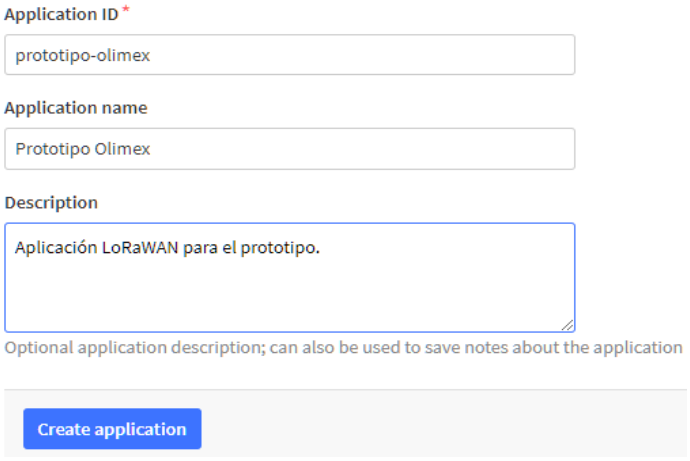

Figura 67. Configuración de aplicación TTN.

Una vez se creada la aplicación se podrán añadir dispositivos a ella. Para registrar el prototipo en la aplicación se añade un dispositivo final en el apartado 'End devices' y se seleccionan los siguientes parámetros:

Figura 68. Registro del prototipo en la aplicación de TTN.

Los parámetros DevEUI, AppEUI y AppKey son los que debe conocer tanto el servidor como el dispositivo final para poder realizar el proceso de activación con el método OTAA (ver apartado 4.1.3.2 Seguridad LoRaWAN). El DevEUI es un identificador único de la MCU proporcionado por el fabricante, aunque también se encuentra almacenado en la memoria flash, pudiéndose acceder a él mediante el código del programa. El AppKey y AppEUI se pueden consultar y modificar en el configurador STM32CubeMX, en el apartado Middleware -> LORAWAN -> LoRaWAN commissioning.

En este punto los datos enviados por el prototipo se podrán visualizar en la sección 'Live data' del dispositivo registrado. Como se ve en la Figura 69, los mensajes se reciben y se muestran desencriptados y en código ASCII en hexadecimal, junto con información como el Data Rate, el RSSI y el SNR.

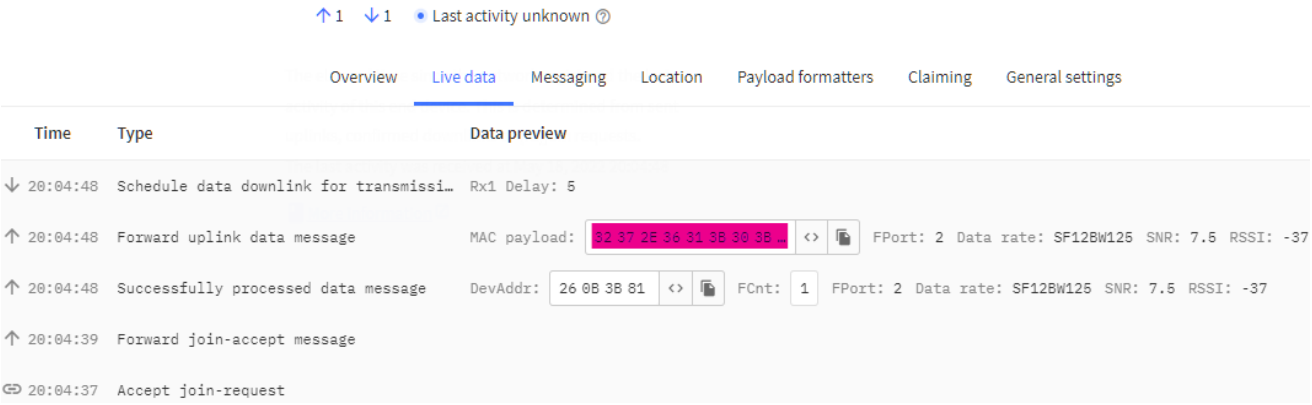

Figura 69. Consola 'Live data' con mensajes enviados por el prototipo.

Estos datos serán enviados como un JSON serializado a través de MQTT para ser procesados y mostrados en LabVIEW. El servidor MQTT será proporcionado por la propia plataforma The Things Network. Se puede encontrar la dirección publica con el puerto correspondiente en el apartado Integrations -> MQTT de la aplicación creada, junto con el usuario y la contraseña, la cual se creará automáticamente al presionar el botón para generar una nueva clave API.

Figura 70. Servidor MQTT de TTN.

Cuando se recibe un mensaje, este se publica automáticamente en el servidor MQTT de TTN en el topic 'v3/{application id}@ttn/devices/{device id}/up' con un formato JSON en el que se incluyen todos los datos de la transmisión, incluyendo el Payload desencriptado y en código ASCII en base 64. Para enviar un mensaje se deberá publicar en el topic v3/{application id}@ttn/devices/{device id}/down/replace un JSON serializado que contenga el puerto de la aplicación, el Payload en base 64 y la prioridad. [58]

4.2.5. Cliente en LabVIEW

En LabVIEW se diseñará un programa que se suscribirá al topic del servidor MQTT donde se publican los mensajes recibidos del prototipo. El JSON leído en el topic se procesará para extraer la información de interés, la cual será el SNR, el RSSI, el SF, el tiempo en el aire y el Payload. El Payload se procesará de forma adecuada para pasar de base 64 a ASCII el String que contiene el mensaje enviado por la MCU con la temperatura, el SNR y el RSSI separados por ';'. Toda la información obtenida será mostrada en un dashboard. También se deberá publicar en el topic correspondiente para transmitir el mensaje 'X' al prototipo, y que se envié de vuelta el SNR y el RSSI con la que recibe el prototipo.

Primero se debe tener un programa capaz de suscribirse al topic del servidor MQTT de The Things Network. Para ello se utilizará un proyecto de MQTT publicado en GitHub [59], que contiene un archivo de ejemplo "test.vi" sobre el que se añadirá el resto del programa necesario para procesar y mostrar los datos. Este programa de ejemplo se basa en una máquina de estados a la que se le pasan los parámetros necesarios para poder conectarse al servidor MQTT como la dirección del servidor, el puerto, el usuario o la contraseña.

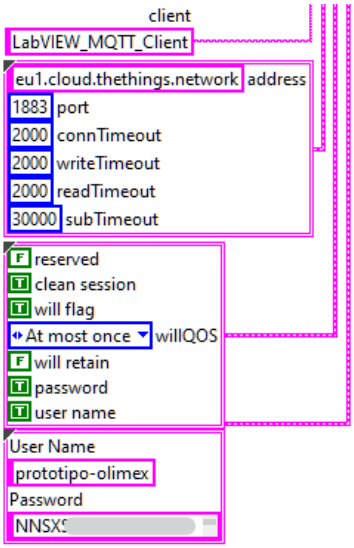

Figura 71. Datos de la máquina de estados para la conexión al servidor MQTT.

La máquina de estados se basa en estados como el de conexión al servidor, el de publicar, suscribirse, desconectarse, nuevo mensaje en el topic suscrito, etc. La conexión al servidor y la suscripción al topic se gestionan en el dashboard mostrado en la siguiente imagen:

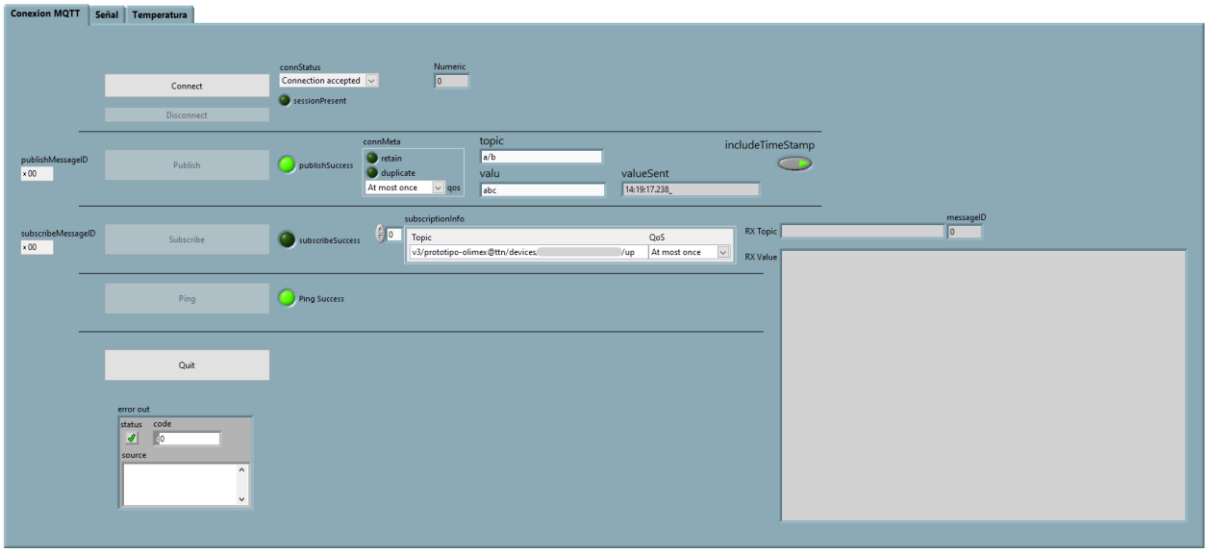

Figura 72. Panel en LabVIEW de la conexión MQTT.

Cuando el mensaje recibido por el servidor TTN se publica en el topic v3/prototipo-olimex@ttn/devices/{device id}/up', el contenido se puede visualizar como un JSON serializado, en el estado de la máquina que se activa cuando hay una nueva publicación en el topic suscrito. Para trabajar con el mensaje fuera de la máquina de estados se empleará una cola de mensajes, situando un productor dentro de dicho estado que almacenará el String recibido en la cola, y un consumidor dentro un bucle con un tiempo de espera de 500 ms.

Dentro de ese estado también se publicará en el topic v3/prototipo-olimex@ttn/devices/{device id}/down/replace el JSON serializado que se ve en la Figura 73. Con esta publicación se enviará un mensaje al prototipo a través de LoRaWAN que contendrá el carácter 'X' necesario para que la MCU actualice las variables que almacenan los valores de SRN y RSSI de dicho mensaje.

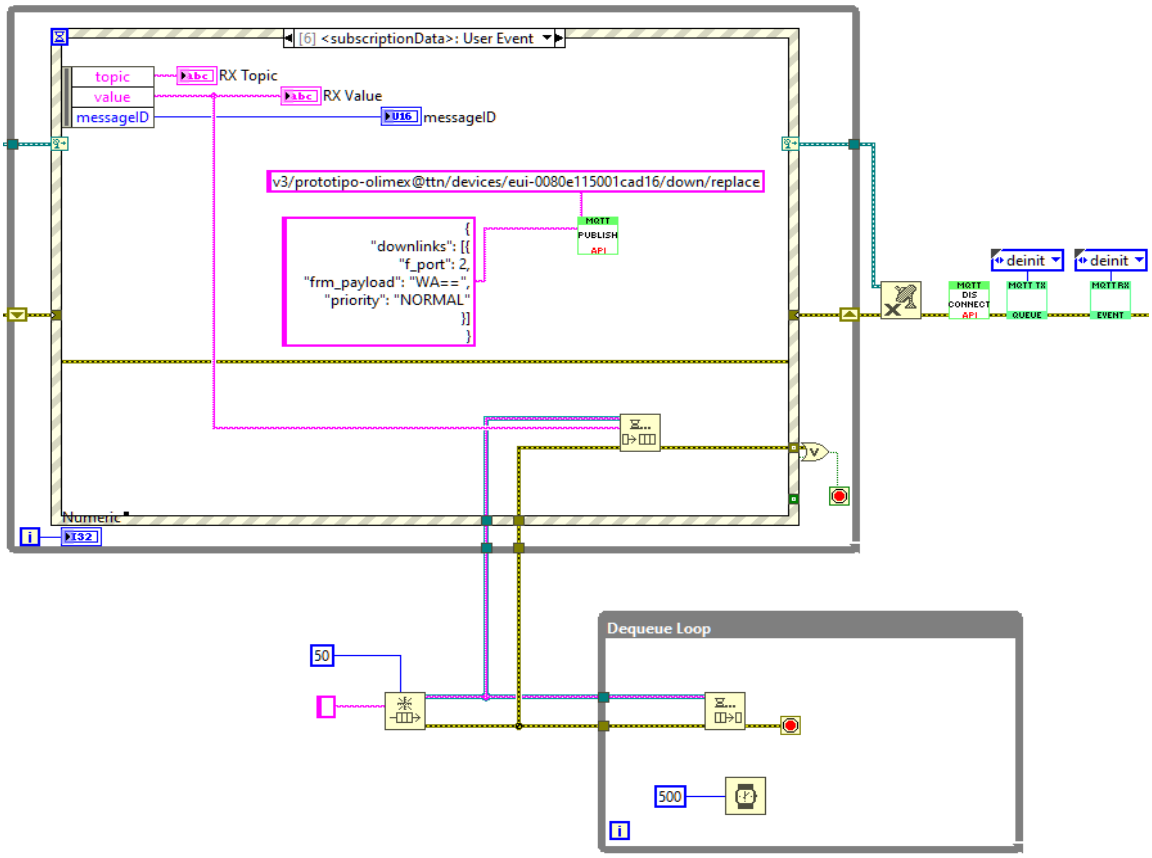

Figura 73. Implementación de la cola de mensajes.

Ahora se dispone del JSON serializado en el bucle (loop). Para poder trabajar con él, se debe emplear el complemento de LabVIEW llamado JKI JSON [60]. Con él se podrá pasar el JSON serializado anidado a un grupo (cluster) que se podrá procesar en LabVIEW. Para ello se necesita conocer la estructura del JSON y el nombre de los parámetros que se buscan, con esto se creará un grupo que siga dicha estructura y contenga las variables buscadas. Este grupo se pasará como parámetro junto con el JSON serializado a los dos bloques que los procesarán y los transformarán para devolver un grupo con el valor de las variables buscada, como se muestra en la siguiente imagen:

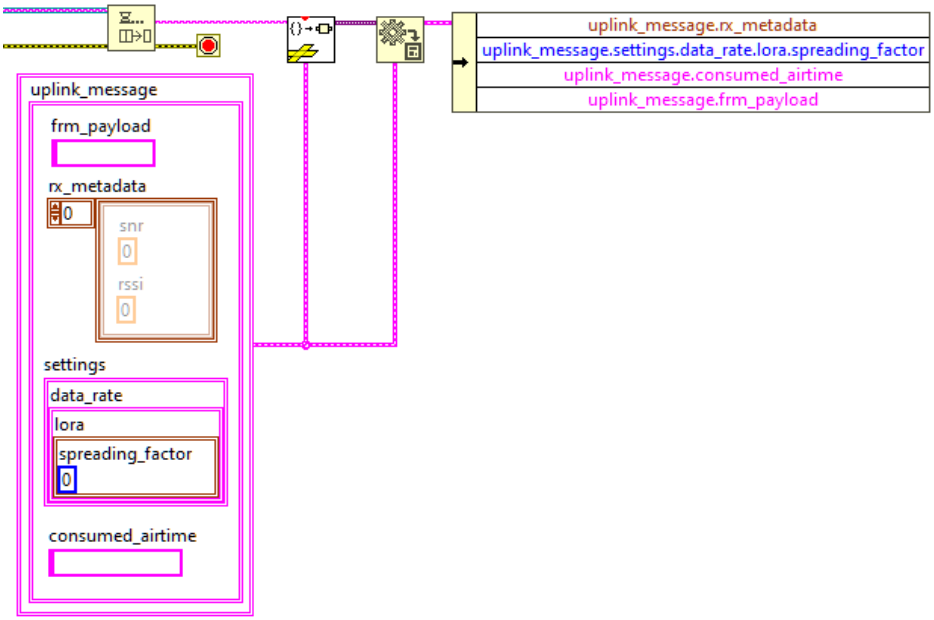

Figura 74. Transformación del JSON a grupo.

Después de esto se pueden mostrar los parámetros en el panel. El valor del SNR y el RSSI se encuentran en un grupo dentro de un vector, una vez se accede a ellos se muestran en una gráfica. El factor de propagación, SF, se obtiene directamente como un entero y se muestra en el panel. El tiempo en el aire se obtiene en un String y debe ser transformado a un número decimal para ser mostrado en el panel.

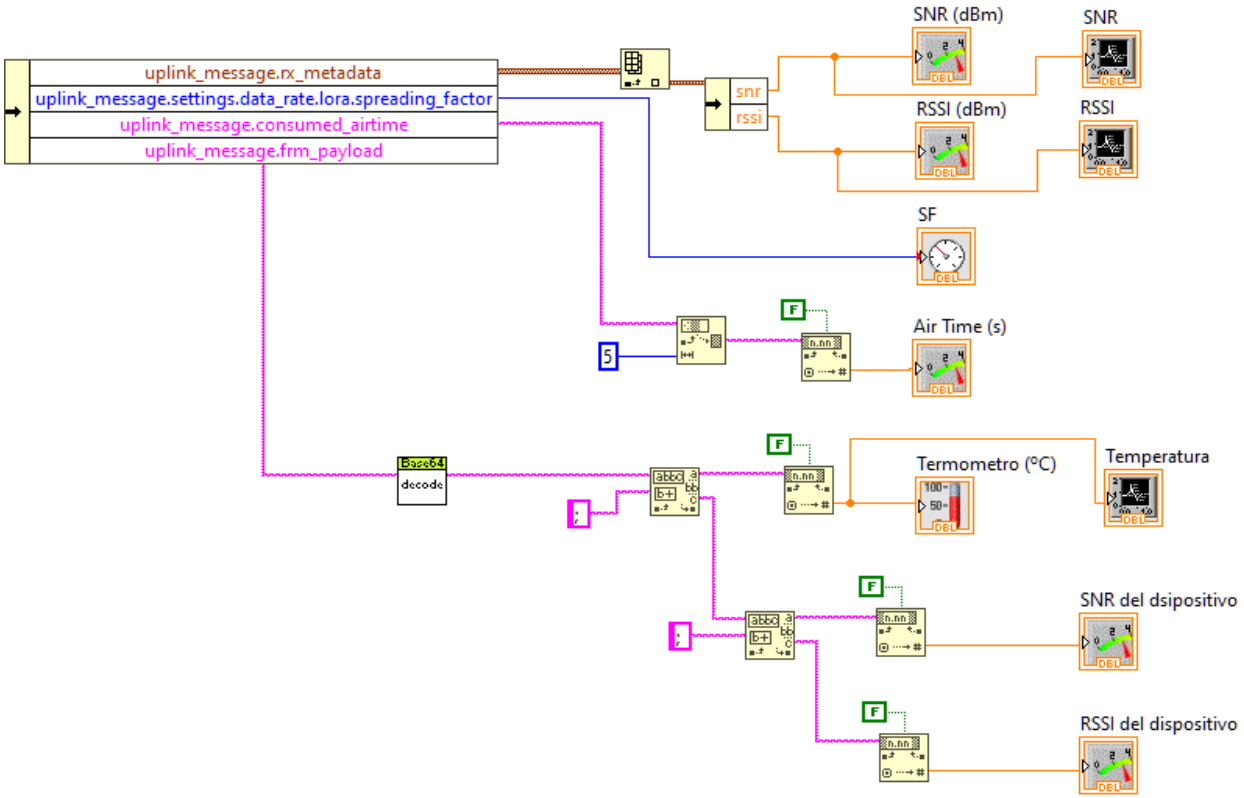

Figura 75. Transformación y representación de los distintos parámetros.

El Payload se encuentra en una cadena de caracteres, y contiene la información útil en código ASCII en base 64, por lo que se debe pasar a sus correspondientes caracteres ASCII. Para ello se empleará un archivo VI de decodificador rápido de base 64 [61], este bloque devolverá un String que contiene los valores de RSSI y de SNR de la transmisión recibida por el prototipo, y el valor de la temperatura, todos ellos separados por ';'. Este String se procesa para separar los distintos valores buscando el carácter ';'. Cada parámetro se transforma a un valor numérico y se muestra en los paneles de las Figura 76 y Figura 77.

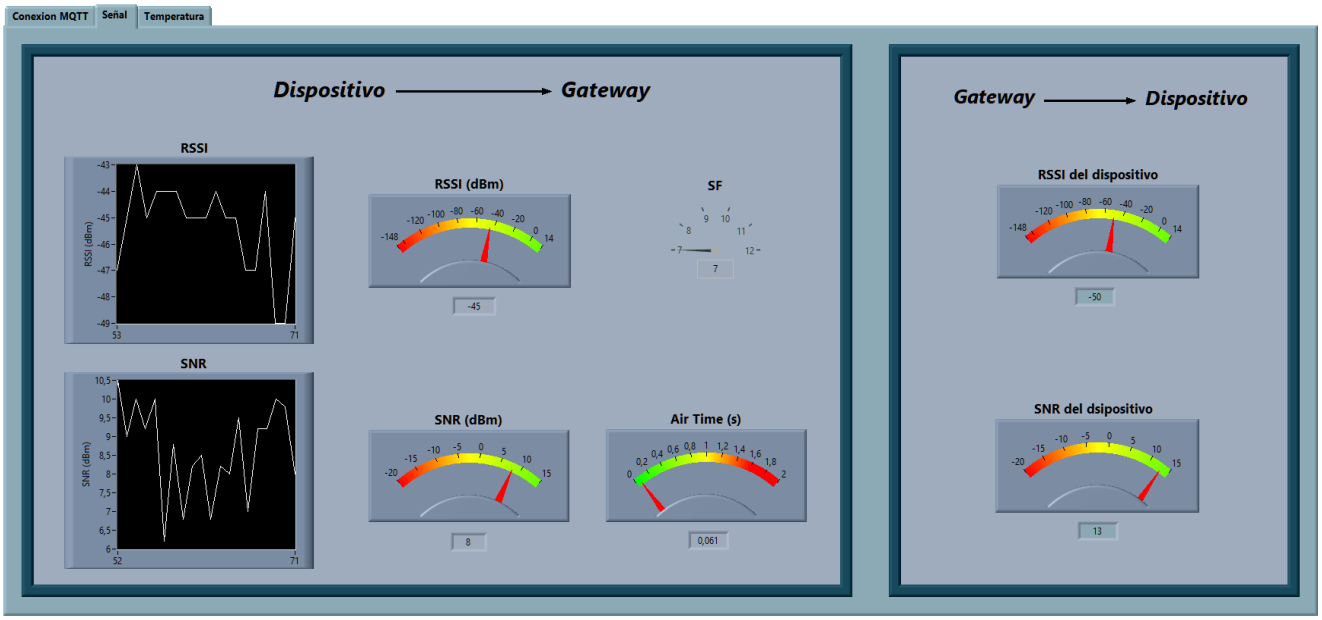

Figura 76. Panel de parámetros de la señal en LabVIEW.

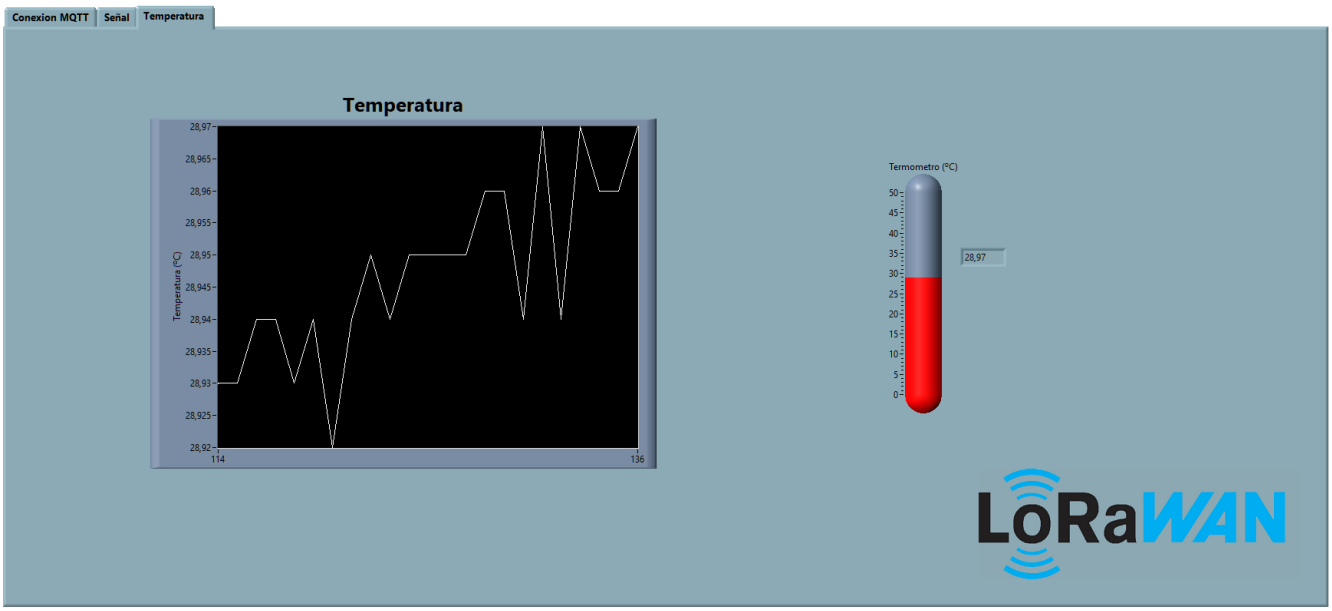

Figura 77. Panel con temperatura en LabVIEW.

5. RESULTADOS

5.1. ANÁLISIS DE CONSUMOS REALES

En este apartado se mostrarán los resultados de consumo de corriente obtenidos en el laboratorio empleando el osciloscopio GW Instek MSO-2204EA. Estas medidas corresponden a distintas etapas del ciclo de transmisión y recepción del MCU. Los valores de la corriente en las gráficas se encuentran sin correcciones para la resistencia de 10,5 Ω, pero los valores medios calculados que se muestran contienen las correcciones necesarias para ajustarlos al valor real de la resistencia.

5.1.1. Corriente en modo Sleep

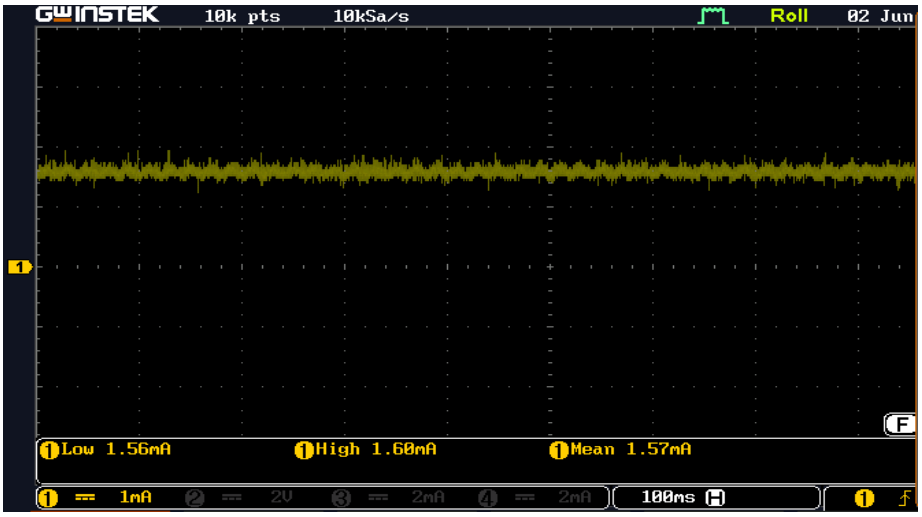

Figura 78. Captura de osciloscopio de la corriente en modo Sleep.

En la Figura 78 se puede ver el consumo de corriente medido por el osciloscopio con una escala de 1 mA/cuadrado, con el fin de obtener una resolución mayor de la corriente consumida por el MCU en modo Sleep. El consumo medio es de 1,57 mA según la función 'Mean' del osciloscopio.

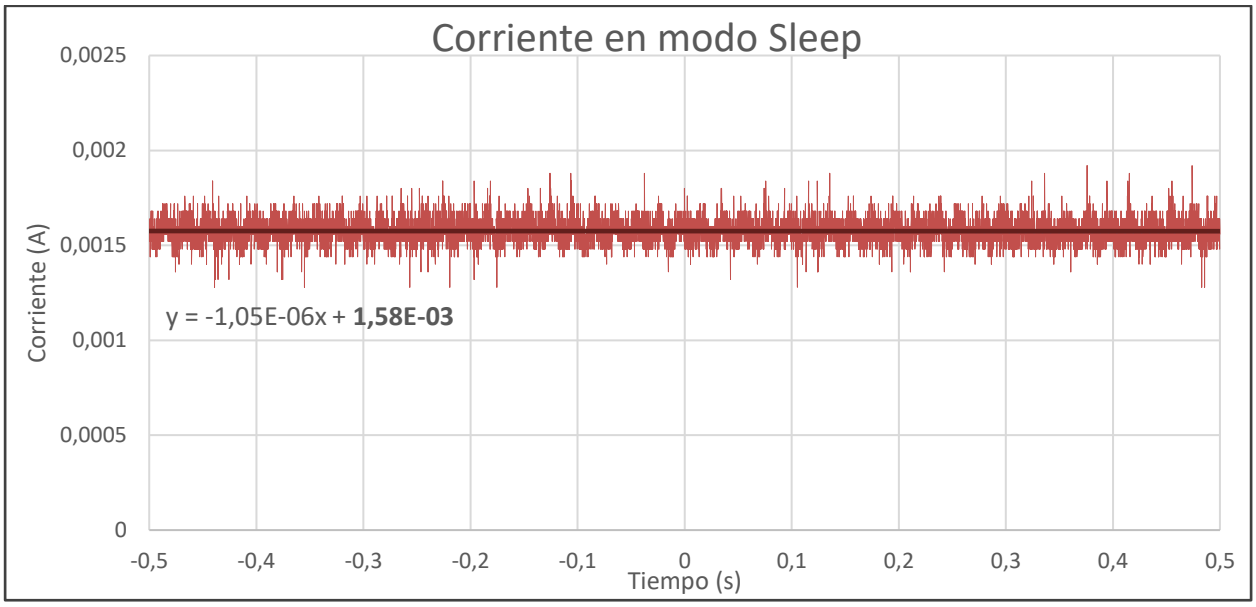

Figura 79. Gráfico de corriente en modo Sleep.

La Figura 79 corresponde a la misma medida de osciloscopio de la Figura 78, pero esta vez grabando el valor de cada punto y graficándolos con el fin de poder analizarlo en detalle.

En la gráfica se puede ver la línea de tendencias de la señal, donde la pendiente es despreciable y el offset es de 1,58 mA. Este offset corresponde con la media de la corriente en el modo Sleep.

Con el fin de obtener un valor de forma más analítica sin depender del valor de la pendiente, se calcula la media aritmética de los datos obtenidos. De esta forma se obtiene una corriente media de 1,50 mA, una vez se corrige su valor para una resistencia de 10,5 Ω.

5.1.2. Corriente en modo Run

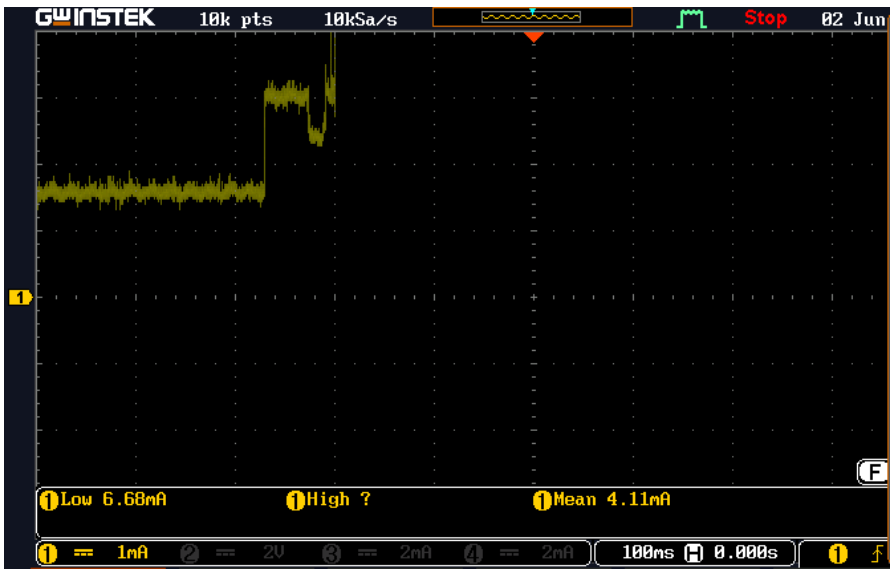

Figura 80. Captura de osciloscopio de la corriente en modo Run.

Al igual que en el caso anterior, la Figura 80 corresponde a una captura de osciloscopio en la que se puede ver consumo de corriente manteniendo la escala 1 mA/cuadrado. En este caso corresponde al consumo antes de realizar una transmisión LoRaWAN, donde en el primer "pico" de corriente el procesador se encuentra en modo Run y el sensor se encuentra realizando la conversión de temperatura. Cuando el sensor de temperatura termina la conversión de temperatura se puede apreciar como el consumo disminuye, en ese punto el MCU se encuentra en modo Run. Finalmente comienza la ventana de transmisión, la cual se sale de la escala en esta captura.

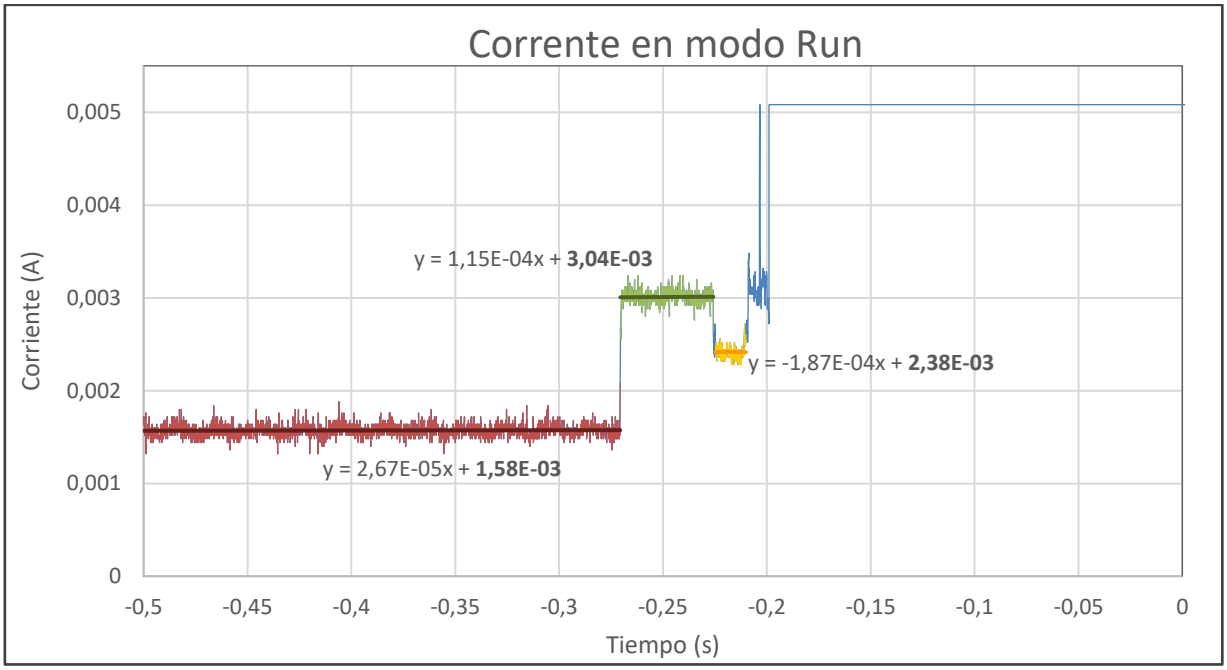

Figura 81. Gráfico de corriente en modo Run.

En la Figura 81 se pueden ver las distintas etapas diferenciadas por colores. La primera, en color rojo, corresponde al modo Sleep visto anteriormente. Al igual que el caso anterior, al calcular la media aritmética, se obtiene un valor de 1,50 mA.

Después de esta etapa, vemos el tramo de color verde, donde el microcontrolador pasa a modo Run y el sensor comienza la conversión de temperatura en modo One-Shoot. Esta etapa dura 44 ms, un valor muy cercano al tiempo típico de conversión proporcionado por el data sheet del sensor de temperatura, el cual es de 45 ms. Su media de corriente es de 2,88 mA.

Cuando el sensor termina la conversión el consumo baja, esto lo vemos en el tramo de amarillo. En este tramo el MCU se encuentra en modo Run, y se debe al tiempo de espera implementado en el código de 60 ms para esperar a la conversión de temperatura. Esta etapa dura 14,9 ms, al sumar ambas etapas (verde y amarilla) se obtienen los 60 ms de espera implementados en el código. Esta etapa tiene una media de corriente de 2,32 mA.

Por último, comienza el proceso de transmisión, el cual se sale de la escala de medida debido a que su consumo es mucho mayor. Por lo que la línea recta horizontal que se observa al final del grafico no es un valor real de consumo.

5.1.3. Corriente de las ventanas RX.

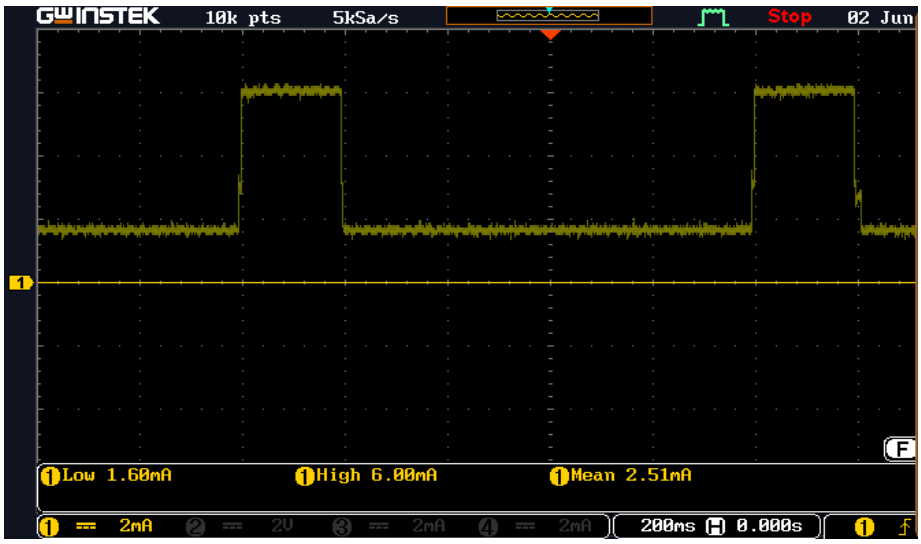

Figura 82. Captura de osciloscopio de la corriente durante las ventanas RX.

En la Figura 82 se pueden observar las dos ventanas de recepción que se abren de forma automática a la espera de recibir un Downlink. En este caso la escala de la corriente se ha aumentado a 2 mA/cuadrado con el fin de poder visualizar la corriente de las ventanas de recepción. El resto del tiempo corresponde a la corriente del MCU en modo Sleep.

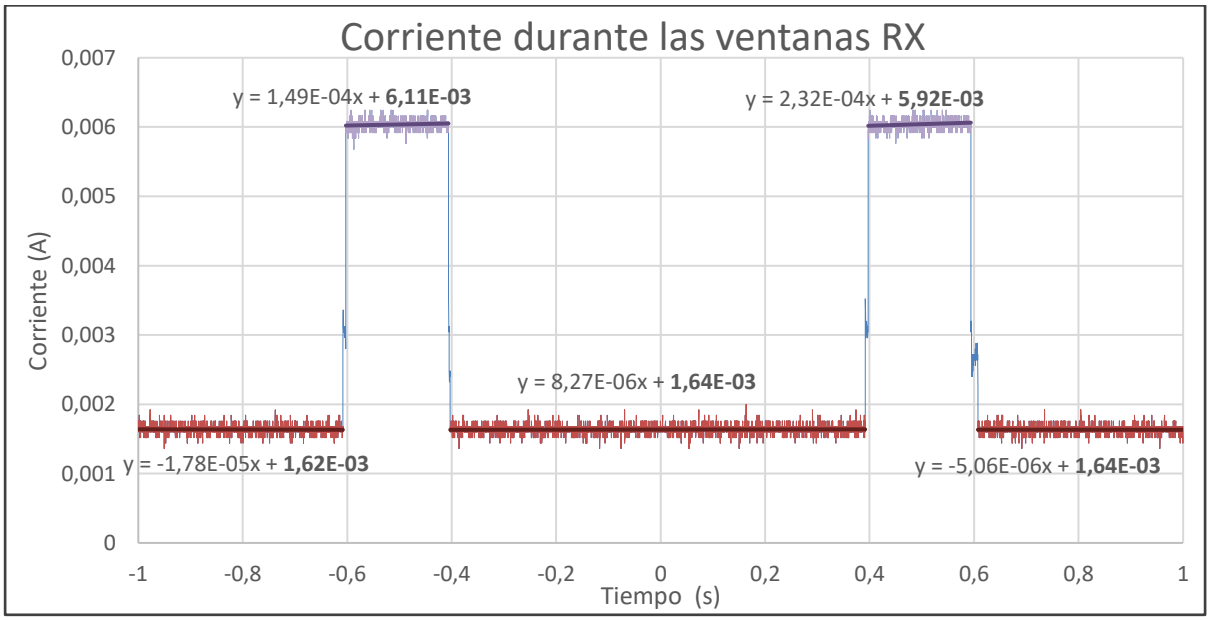

Figura 83. Gráfico de corriente durante las ventanas RX.

Al analizar la Figura 83 se diferencian principalmente dos estados. Uno correspondiente al modo Sleep (color rojo) y otro correspondiente a al estado de recepción LoRaWAN (color violeta). El estado de recepción (RX) se activa en ventanas de tiempo programadas. En este caso en ninguna de las dos ventanas se ha recibido un mensaje, es por eso que ambas permanecen abiertas el mismo tiempo. El tiempo de apertura medido es de 194 ms, mientras que el tiempo que separa el inicio de la primera ventana con el inicio de la segunda es de 1,00 segundos.

En cuanto al consumo medio, se obtienen valores de 5,76 mA en la etapa de recepción y de 1,56 mA en el modo Sleep. Se puede observar como el consumo de Sleep ha aumentado, pero esto solo se debe a que, al aumentar la escala, la resolución y exactitud han disminuido. Por ello que el consumo en modo Sleep que se tendrá en cuenta es el obtenido anteriormente con escalas menores, sin tener en cuenta el obtenido en esta medición.

5.1.4. Corriente de la ventana TX.

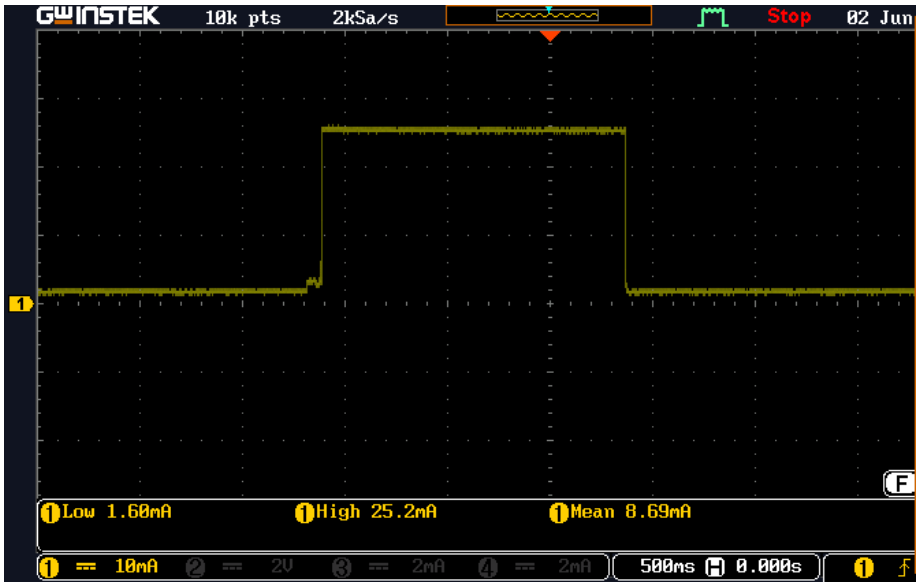

Figura 84. Captura de osciloscopio de la corriente durante la ventana TX.

En la Figura 82 se pueden observar la ventana de transmisión (TX) durante la cual se envía el Uplink de LoRaWAN. Esta ventana se abre de forma periódica según el tiempo especificado por la aplicación. La escala de corriente se ha vuelta a aumentar, en este caso a 10 mA/cuadrado debido al aumento de la corriente durante la ventana de transmisión.

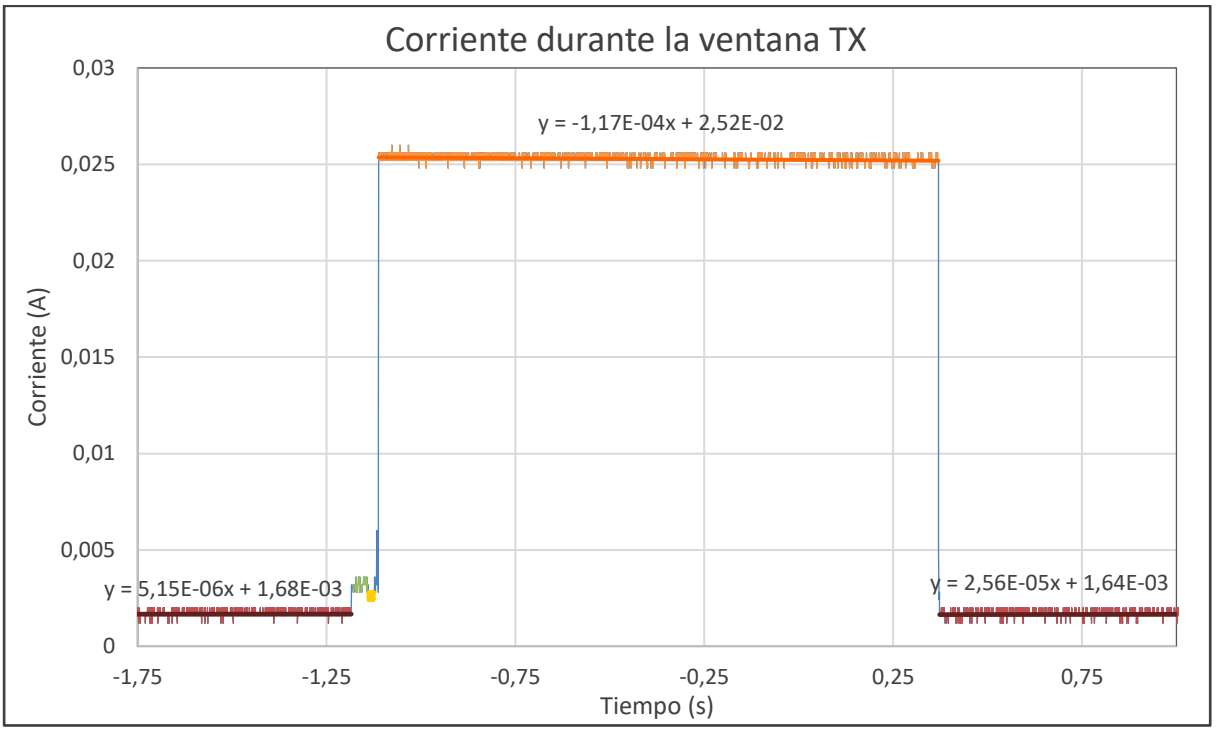

Figura 85. Gráfico de corriente durante la ventana TX.

En la Figura 85 se diferencian, de nuevo, dos estados principales, el de Sleep (color rojo) y el de transmisión LoRaWAN (color naranja). También se pueden apreciar los estados de Run más sensor y de Run analizados anteriormente.

El estado de transmisión se activa periódicamente y permanece activo hasta que termina la transmisión del mensaje. El tiempo que permanece abierta la ventana de transmisión depende de varios factores, como la longitud del mensaje o el factor de propagación (SF). En este caso concreto se ha transmitido con un SF de 12 y ha permanecido abierta durante 1,48 segundos.

En cuanto al consumo medio, se han obtenido valores de 24,1 mA en la etapa de transmisión y de 1,59 mA en el modo Sleep. Al igual que en el caso anterior, tampoco se tendrá en cuenta el consumo en modo Sleep de esta medición debido a la escala.

5.1.5. Corriente del proceso de transmisión y recepción

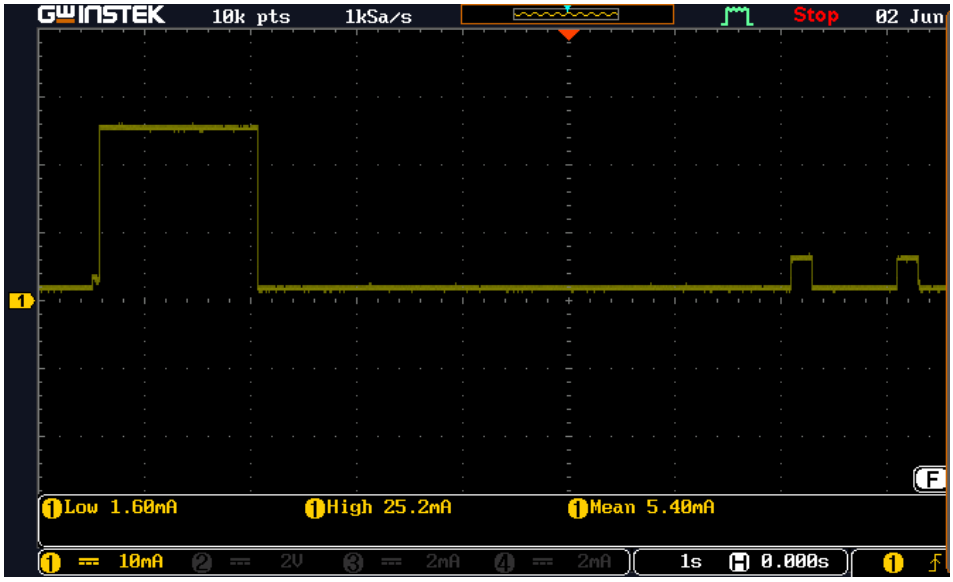

Figura 86. Captura de osciloscopio de la corriente durante todo el proceso TX y RX.

En esta figura se observa el proceso completo de una transmisión LoRaWAN de clase A, incluyendo las dos ventanas de recepción que se abren 5 segundos después de la transmisión. Este proceso se repite de forma periódica, puesto que comienza con la transmisión donde se envía el Uplink, la cual se realiza cada vez que transcurre el tiempo definido por la aplicación. La escala de corriente es la misma que la empleada para la ventana de transmisión, 10 mA/cuadrado, debido a que es la que mayor corriente consume.

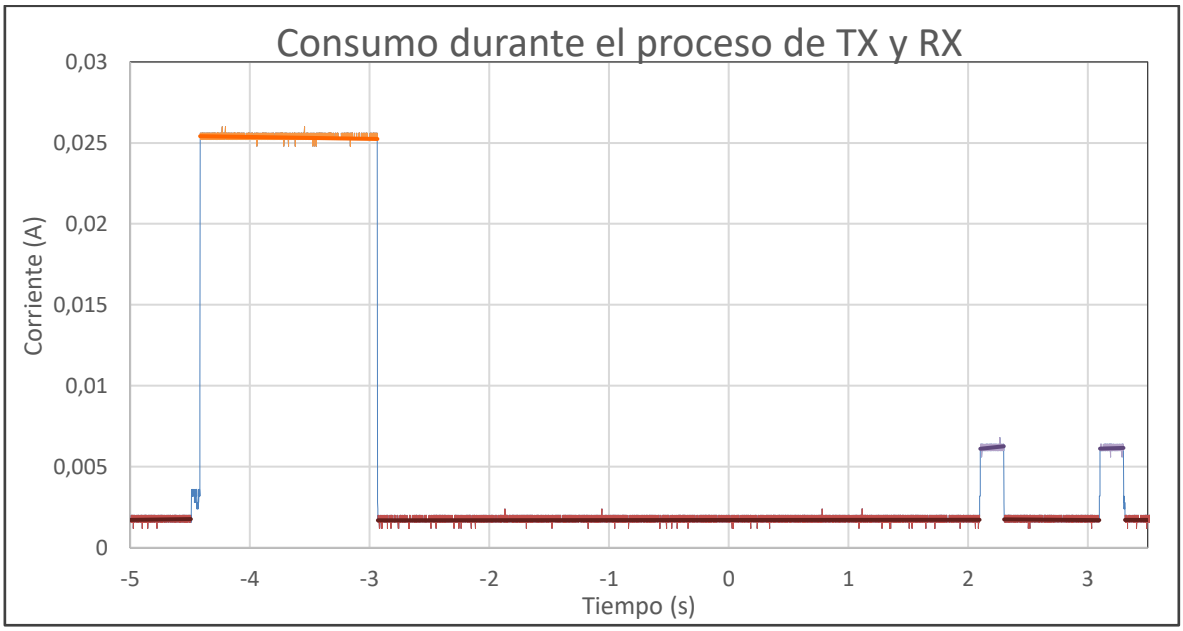

Figura 87. Gráfico del consumo durante todo el proceso de TX y RX.

En la Figura 87 se ven todas las etapas analizadas anteriormente, las cuales forman todo el proceso de transmisión y recepción de la clase A de LoRaWAN. El proceso comienza con la etapa de Run más la conversión de temperatura del sensor, y termina con la última ventana de recepción y tiene una duración total de 7,79 segundos. Desde el final de la transmisión hasta el inicio de la primera ventana de recepción transcurren 5,03 segundos, mientras que hasta el principio de la segunda ventana de recepción transcurren 6,03.

En cuanto al consumo medio de este proceso, es de 6,14 mA. Aunque se debe tener en cuenta que el tiempo que permanecen abiertas las ventanas de transmisión y recepción es variable, por lo que este consumo también será variable y dependerá de las condiciones de la transmisión y de si se recibe o no algún Downlink durante las ventanas de recepción.

En el caso de la ventana de transmisión, a la hora de realizar estas mediciones, ha permanecido en abierta durante 1,48 segundos debido a que se encontraba en una de las condiciones más desfavorables con un SF de 12. Aunque existen varios factores que afectan al tiempo de transmisión, el más relevante es el SF, si este disminuye, el tiempo que permanece abierta la ventana de transmisión también se reducirá.

El consumo medio de este proceso no es el consumo medio del prototipo, puesto que no se tiene en cuenta el tiempo que permanece en Sleep entre cada proceso de transmisión. Este consumo corresponde únicamente al periodo de tiempo transcurrido entre el inicio del proceso de transmisión hasta la última ventana de transmisión.

5.2. COMPARACIÓN ENTRE CONSUMO REAL Y TEÓRICO

En la siguiente figura se pueden observar el valor simulado de la corriente en cada una de las distintas etapas configuradas en el programa STM32CubeMX. El tiempo de cada etapa se ha introducido de forma que coincida con los tiempos medidos anteriormente.

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bu...	Clock Config	Peripher...	Step Current	Duration
1	RUN	3.0	Range1-Medium/SMPS-ON	SRAM1	16 MHz	MSI	ADC:fs_...	2.78 mA	44 ms
2	RUN	3.0	Range1-Medium/SMPS-ON	SRAM1	16 MHz	MSI	ADC:fs_...	2.18 mA	15 ms
3	RUN	3.0	Range1-Medium/SMPS-ON	SRAM1	16 MHz	MSI	ADC:fs_...	25.68 mA	1.48 s
4	SLEEP	3.0	Range1-Medium/SMPS-ON	FLASH	16 MHz	MSI	ADC:fs_...	1.48 mA	5 s
5	RUN	3.0	Range1-Medium/SMPS-ON	SRAM1	16 MHz	MSI	ADC:fs_...	6.98 mA	200 ms
6	SLEEP	3.0	Range1-Medium/SMPS-ON	FLASH	16 MHz	MSI	ADC:fs_...	1.48 mA	0.8 s
7	RUN	3.0	Range1-Medium/SMPS-ON	SRAM1	16 MHz	MSI	ADC:fs_...	6.98 mA	200 ms

Figura 88. Etapas del proceso total de transmisión en el simulador de STM32CubeMX.

Tabla 6. Valor simulado de la corriente de las distintas etapas.

Modo	Corriente (mA)	Duración
Run + sensor	2,78	44 ms
Run	2,18	15 ms
Run (Tx)	25,68	1,48 s
Sleep	1,48	5 s
Run (Rx)	6,98	200 ms
Sleep	1,48	800 ms
Run (Rx)	6,98	200 ms
Proceso completo	6,4	7,74 s

El simulador de consumo de STM32CubeMX, también ofrece una gráfica con las etapas introducidas y su consumo correspondiente, Figura 89. También nos proporciona el valor medio del consumo de corriente de todo el proceso de transmisión, el cual es de 6,4 mA.

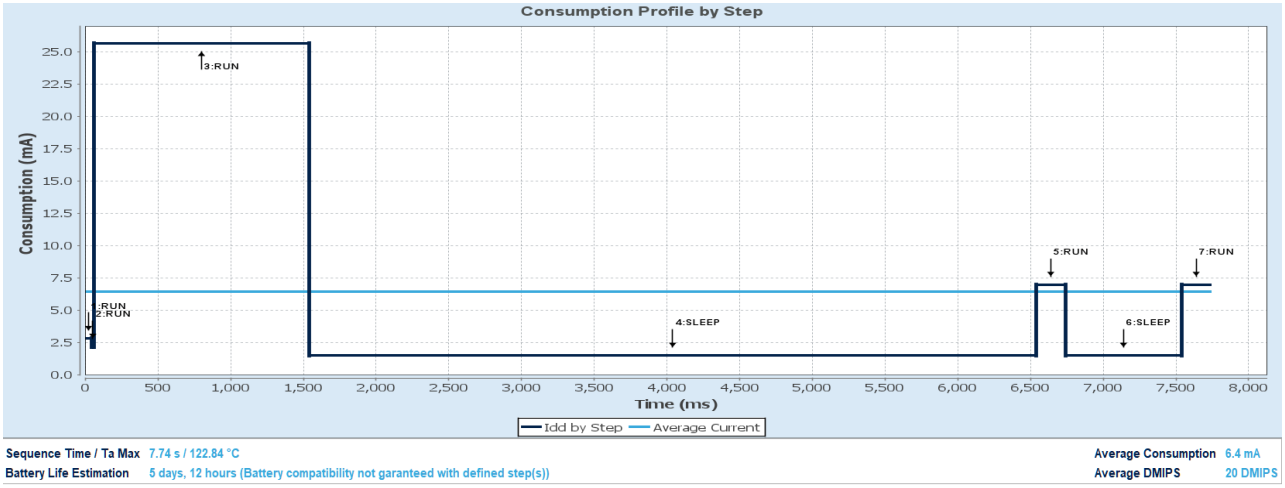

Figura 89. Gráfica de simulación del consumo.

En la siguiente tabla se muestra compara y se muestra el error entre el consumo real y el simulado. Se debe tener en cuenta que el valor del consumo real es el obtenido después de realizar la corrección para adaptar las medidas del osciloscopio a una resistencia de 10,5 Ω.

Modo	Corriente real (mA)	Corriente simulada (mA)	Error relativo
Run + sensor	2,88	2,78	3,44 %
Run	2,32	2,18	6,30 %
Run (Tx)	24,1	25,68	6,23 %

Modo	Corriente real (mA)	Corriente simulada (mA)	Error relativo
Run (Rx)	5,76	6,98	17,54 %
Sleep	1,50	1,48	1,37 %
Proceso completo	6,14	6,4	4,00 %

El simulador nos ofrece la posibilidad de estimar la vida de la batería. Si se tiene en cuenta el tiempo de Sleep entre transmisión y transmisión, para el caso del prototipo transmitiendo cada 10 segundos, el consumo medio es de 5,29 mA y la vida de una batería de Li-MnO2 de 850 mAh es de 6 días y 16 horas.

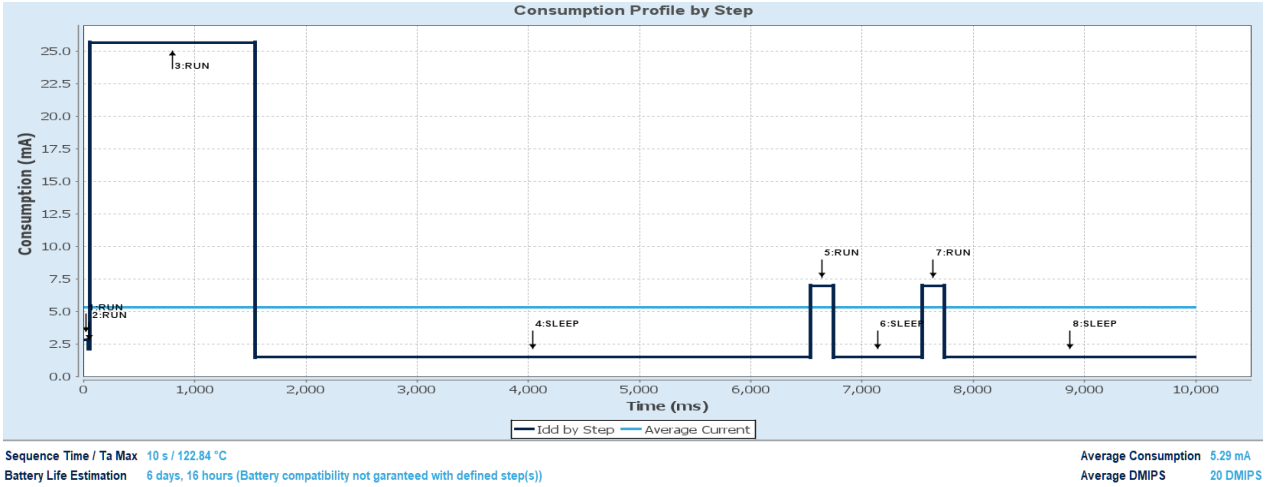

Figura 90. Gráfica de simulación del consumo con Sleep entre transmisiones.

El tiempo de transmisión de 10 segundos se estableció para la realización de pruebas. En una aplicación real este tiempo sería mayor ya que la temperatura del cuerpo humano varía de forma mucho más lenta. En la siguiente tabla se pueden apreciar los consumos medios y la autonomía suponiendo la misma batería de litio de 850 mAh en distintos casos de uso, variando la frecuencia de las transmisiones.

Tabla 7. Casos de uso con distintos tiempos de transmisión.

Tiempo entre transmisiones	Consumo medio (mA)	Autonomía de la batería
2 minutos	1,8	16 días y 5 horas
10 minutos	1,54	18 días y 21 horas
30 minutos	1,5	19 días y 10 horas
1 hora	1,49	19 días y 13 horas
3 horas	1,48	19 días y 15 horas

6. CONCLUSIONES

Se ha logrado implementar un prototipo funcional empleando toda la arquitectura LoRaWAN, desde la captura del sensor hasta el cliente final donde se muestran los datos. Se ha conseguido empleando en el prototipo una nueva arquitectura de nodo final formada por un único microcontrolador con un transceptor LoRa integrado y empleando las herramientas de desarrollo del entorno STM32Cube.

Al emplear el microcontrolador de la familia STM32WL se ha visto como el hecho detener el transceptor LoRa integrado en el mismo encapsulado simplifica considerablemente en diseño del hardware y reduce el tamaño del dispositivo.

El hecho de no haber empleado núcleos complejos en el diseño del hardware del prototipo hace que su esquema electrónico sea mucho más simple y cercano al de un dispositivo wearable. Esto facilita un posible desarrollo de una PCB (placa de circuito impreso) que integre los elementos empleados en el prototipo y ayuda a predecir su consumo, puesto que debería ser muy similar a los resultados obtenidos con el prototipo.

Las herramientas de desarrollo del entorno STM32Cube que ofrece la compañía STMicroelectronics y la completa integración del protocolo LoRaWAN en ellas, simplifica en gran medida el diseño del software. El configurador gráfico STM32CubeMX ha simplificado y facilitado el proceso de configuración de los distintos periféricos, incluyendo la configuración relativa a LoRaWAN. También ha resultado muy útil por la facilidad que ofrece para realizar cambios en los distintos periféricos a la hora de realizar pruebas para ajustar el consumo energético.

Hacer uso del secuenciador que implementa y gestiona los modos de bajo consumo del microcontrolador de forma automática ha ayudado a mejorar el ahorro de energía y ha facilitado enormemente la integración de estos modos, ya que lleva al procesador al modo Sleep automáticamente cuando el procesador se encuentra ocioso, sin necesidad de realizar cambios en el programa generado. Aunque para llevarlo a modos más profundos de ahorro de energía es necesario implementarlos manualmente en el código.

En cuanto a los consumos energéticos medidos, estos cumplen con los obtenidos en el simulador, aunque existen algunas desviaciones. El mayor error se obtiene en el consumo de las ventanas de transmisión, el cual es un 17,54% menor al simulado, esto podría deberse a la configuración física de la antena o a la diferencia en alguno de los parámetros de LoRaWAN o la radio sub-GHz entre la simulación y el prototipo. A pesar de este error o de los errores de más del 6% en el modo de transmisión y Run, el error general del proceso de transmisión y recepción es del 4%. Esto se debe a que la mayor parte del tiempo de

este proceso, el microcontrolador se encuentra en modo Sleep, un modo en el que se ha obtenido un consumo muy cercano al simulado, con un error del 1,37%.

Cuanto mayor es el tiempo entre transmisiones, más tiempo permanece en modo Sleep y menor es el consumo. Cuando el tiempo que permanece en Sleep es lo suficientemente grande, el resto de los modos apenas afectan al consumo medio, por lo que el error cometido con respecto al valor simulado sería prácticamente el mismo que se comete en el modo Sleep, mayor al 1%. Por ello el simulador resulta ser una herramienta bastante precisa para predecir el consumo de un dispositivo wearable en función de su configuración y de sus periféricos. Aunque se debe emplear adecuadamente, conociendo la relación entre los parámetros configurables del simulador y los del MCU, de lo contrario se obtendrían previsiones que no se ajustan a un dispositivo ponible.

Teniendo en cuenta esto, se puede emplear el simulador para realizar previsiones de consumo y vida de la batería, así como obtener soluciones de compromiso entre el tiempo de transmisión y el consumo. Mediante el empleo del simulador se ha visto que para tiempos de transmisión superiores a la media hora apenas existe una mejora en el consumo y la vida de la batería. En este caso, a partir de ese periodo de transmisión el consumo medio depende casi en su totalidad del consumo del modo de ahorro de energía, por lo que para obtener resultados mejores se deberán implementar modos de ahorro de energía más profundos que el modo Sleep.

Con el modo Sleep como modo de bajo consumo, se consigue una autonomía aproximada de 19 días con una batería de 850 mAh. Si bien se trata de una aplicación en la que el reemplazo o recarga de la batería no supone un problema, esta autonomía tiene un amplio margen de mejora utilizando modos de ahorro de energía con menor consumo.

7. OBJETIVOS DE DESARROLLO SOSTENIBLE

Los objetivos de este Trabajo Fin de Grado están alineados con los siguientes Objetivos de Desarrollo Sostenible (ODS) y metas, de la Agenda 2030:

- Objetivo 3 - Garantizar una vida sana y promover el bienestar para todos en todas las edades.
- Meta 3.4 - Para 2030, reducción de las enfermedades no transmisibles y promover la salud mental y el bienestar.

- Objetivo 9 - Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación
- Meta 9.B - Apoyar el desarrollo de tecnologías, la investigación y la innovación nacionales en los países en desarrollo.

8. BIBLIOGRAFÍA

[1] «Estadísticas, hechos y tendencias de Internet de las cosas para 2022», 15 de febrero de 2022. https://findstack.com/es/internet-of-things-statistics/ (accedido 1 de junio de 2022).

[2] M. Choi, G. Li, R. Todrzak, Q. Zhao, J. Raiti, y P. Albee, «Designing a LoRa-based Smart Helmet to Aid in Emergency Detection by Monitoring Bio-signals», en *2021 IEEE Global Humanitarian Technology Conference (GHTC)*, oct. 2021, pp. 72-75. doi: 10.1109/GHTC53159.2021.9612483.

[3] C. Orfanidis, R. B. H. Hassen, A. Kwiek, X. Fafoutis, y M. Jacobsson, «A Discreet Wearable Long-Range Emergency System Based on Embedded Machine Learning», en *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, mar. 2021, pp. 182-187. doi: 10.1109/PerComWorkshops51409.2021.9430981.

[4] U. Kulkarni, S. Badhan, H. Bandi, S. Bhagat, y M. Parmar, «Implementation of Backtracking Algorithm for navigation in LoRa-enabled wearable device for Dementia patients», en *2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA)*, oct. 2020, pp. 308-312. doi: 10.1109/ICCCA49541.2020.9250891.

[5] S. Kumar *et al.*, «A Wristwatch-Based Wireless Sensor Platform for IoT Health Monitoring Applications», *Sensors*, vol. 20, n.º 6, Art. n.º 6, ene. 2020, doi: 10.3390/s20061675.

[6] «Protocolo MiWi™ | Tecnología de microchip». https://www.microchip.com/en-us/products/wireless-connectivity/sub-ghz/miwi-protocol (accedido 15 de abril de 2022).

[7] «CareBand 4 LoRaWAN Wearable with Panic Button - EU868», *CareBand Store*. https://store.carebandremembers.com/products/careband-4-lorawan-wearable-with-panic-button-eu868 (accedido 15 de abril de 2022).

[8] «LoRaWAN®-Based Button LW004-PB», *MOKOSmart #1 Smart Device Solution in China*. https://www.mokosmart.com/lorawan-button-lw004-pb/ (accedido 15 de abril de 2022).

[9] «WearLOC-2 » Products » ProEsys Srl», *ProEsys Srl*. http://proesystech.com/products/wearloc-2/ (accedido 15 de abril de 2022).

[10] «¿Qué son las redes LPWAN?», *Grupo Sinelec*, 5 de mayo de 2021. https://gruposinelec.com/que-son-las-redes-lpwan/ (accedido 18 de abril de 2022).

[11] «What is LPWAN?» https://www.avsystem.com/blog/LPWAN/ (accedido 18 de abril de 2022).

[12] «Cómo iniciar rápidamente la detección de IoT inalámbrica de baja potencia con módulos RF LPWAN», *Digi-Key Electronics*. https://www.digikey.com.mx/es/articles/how-to-quickly-start-low-power-wireless-iot-sensing (accedido 18 de abril de 2022).

[13] F. Campos, «LPWAN: qué son y para qué se utilizan», *M2M - Logitek*, 21 de julio de 2020. https://www.m2mlogitek.com/lpwan-que-son-y-para-que-se-utilizan/ (accedido 18 de abril de 2022).

[14] «Diferencias entre NB-IOT y LTE-M · Accent Systems», *Accent Systems*, 3 de mayo de 2018. https://accent-systems.com/es/diferencias-nb-iot-lte-m/ (accedido 20 de mayo de 2022).

[15] «NB-IoT vs LTE-M, the new IoT king of the 4G world», 1 de septiembre de 2020. https://www.pickdata.net/es/noticias/nb-iot-vs-lte-m-nuevo-rey-iot-mundo-4G (accedido 20 de mayo de 2022).

[16] «¿Qué es LoRaWAN?» https://www.dlem.es/telegestion/gestion-dispositivos-inteligentes/ (accedido 21 de mayo de 2022).

[17] «LoRa and LoRaWAN: Technical overview | DEVELOPER PORTAL». https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/ (accedido 21 de abril de 2022).

[18] T. T. Network, «¿Qué es la tecnología LoRa y por qué es importante para IoT?», *The Things Network*. https://www.thethingsnetwork.org/community/santa-rosa/post/que-es-la-tecnologia-lora-y-por-que-es-importante-para-iot (accedido 21 de abril de 2022).

[19] S. Montagny, *LoRa - LoRaWAN and Internet Of Things*. [En línea]. Disponible en: https://www.univ-smb.fr/lorawan/livre-gratuit/

[20] «Desarrollar con LoRa para aplicaciones IoT de baja tasa y largo alcance», *Digi-Key Electronics*. https://www.digikey.es/es/articles/develop-lora-for-low-rate-long-range-iot-applications (accedido 25 de abril de 2022).

[21] «Homepage», *LoRa Alliance*®. https://lora-alliance.org/ (accedido 26 de abril de 2022).

[22] «What is LoRaWAN® Specification», *LoRa Alliance*®. https://lora-alliance.org/about-lorawan/ (accedido 26 de abril de 2022).

[23] LoRa Alliance, *What is LoRaWAN*®. [En línea]. Disponible en: https://resources.lora-alliance.org/getting-started-with-lorawan/what-is-lorawan

[24] «LoRaWAN® Link Layer Specification 1.0.4». Accedido: 29 de abril de 2022. [En línea]. Disponible en: https://lora-alliance.org/resource_hub/lorawan-104-specification-package/

[25] «RP2-1.0.3 LoRaWAN® Regional Parameters». Accedido: 4 de mayo de 2022. [En línea]. Disponible en: https://lora-alliance.org/resource_hub/rp2-1-0-3-lorawan-regional-parameters/

[26] «The Things Network». https://www.thethingsindustries.com/docs/getting-started/ttn/ (accedido 6 de junio de 2022).

[27] «What Is The Things Stack?» https://www.thethingsindustries.com/docs/getting-started/what-is-tts/ (accedido 11 de mayo de 2022).

[28] «Quick Start», *The Things Network*. https://www.thethingsnetwork.org/docs/quick-start/ (accedido 6 de junio de 2022).

[29] «Protocolos de comunicación para IoT», *Luis Llamas*. https://www.luisllamas.es/protocolos-de-comunicacion-para-iot/ (accedido 23 de mayo de 2022).

[30] «MQTT - The Standard for IoT Messaging». https://mqtt.org/ (accedido 23 de mayo de 2022).

[31] «Trabajando con JSON - Aprende sobre desarrollo web | MDN». https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON (accedido 6 de junio de 2022).

[32] «Semtech UDP Packet Forwarder», *The Things Network*. https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/semtech-udp/ (accedido 6 de junio de 2022).

[33] «Gateway connection to TTN», *The Things Network*. https://www.thethingsnetwork.org/docs/gateways/start/connection/ (accedido 6 de junio de 2022).

[34] «LoRa Basics™ Station», *The Things Network*. https://www.thethingsnetwork.org/docs/gateways/basics-station/ (accedido 6 de junio de 2022).

[35] «Type ABZ(CMWX1ZZABZ-078) | Product Lineup», *Murata Manufacturing Co., Ltd.* https://www.murata.com/en-eu/products/connectivitymodule/lpwa/overview/lineup/type-abz-078 (accedido 12 de junio de 2022).

[36] «Serie STM32WL - STMicroelectronics». https://www.st.com/en/microcontrollers-microprocessors/stm32wl-series.html (accedido 12 de junio de 2022).

[37] «STM32WLE5CC - Sub-GHz Wireless Microcontrollers. Arm Cortex-M4 @48 MHz with 256 Kbytes of Flash memory, 64 Kbytes of SRAM. LoRa, (G)FSK, (G)MSK, BPSK modulations. AES 256-bit. Multiprotocol System-on-Chip. - STMicroelectronics». https://www.st.com/en/microcontrollers-microprocessors/stm32wle5cc.html (accedido 12 de junio de 2022).

[38] «Advantages of Contact Thermometers over Non-Contact/Infrared Thermometers», *TEGAM*, 14 de noviembre de 2015. https://www.tegam.com/advantages-of-contact-thermometers-over-non-contactinfrared-thermometers/ (accedido 12 de junio de 2022).

[39] «Quickly Integrate Clinical-Grade Temperature Sensing into Portable, Wearable Medical Designs», *Digi-Key Electronics*. https://www.digikey.es/en/articles/quickly-integrate-clinical-grade-temperature-sensing-into-medical-designs (accedido 12 de junio de 2022).

[40] T. Tamura, M. Huang, y T. Togawa, «Current Developments in Wearable Thermometers», *ABE*, vol. 7, n.º 0, pp. 88-99, 2018, doi: 10.14326/abe.7.88.

[41] «4 Most Common Types of Temperature Sensor», *Ametherm*, 11 de noviembre de 2020. https://www.ametherm.com/blog/thermistors/temperature-sensor-types (accedido 12 de junio de 2022).

[42] «IC (Integrated circuit) temperature transducers», *Industrial Automation, PLC Programming, scada & Pid Control System*, 26 de abril de 2018. https://forumautomation.com/t/ic-integrated-circuit-temperature-transducers/3509 (accedido 12 de junio de 2022).

[43] «I2C Primer: What is I2C? (Part 1) | Analog Devices». https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html (accedido 31 de mayo de 2022).

[44] «Cómo Medir Corriente Usando Sensores de Corriente | Dewesoft». https://dewesoft.com/es/daq/como-medir-corriente-usando-sensores-de-corriente (accedido 13 de junio de 2022).

[45] «Conocer, seleccionar y usar eficientemente sondas de corriente», *Digi-Key Electronics*. https://www.digikey.es/es/articles/understanding-selecting-effectively-using-current-probes (accedido 13 de junio de 2022).

[46] «Fever click — board with MAX30205 human body temperature sensor | MikroElektronika», *MIKROE*. http://www.mikroe.com/fever-click (accedido 31 de mayo de 2022).

[47] «MAX30205.pdf». Accedido: 15 de mayo de 2022. [En línea]. Disponible en: https://datasheets.maximintegrated.com/en/ds/MAX30205.pdf

[48] «BB-STM32WL», *Olimex*. https://www.olimex.com/Products/IoT/LoRa/BB-STM32WL/ (accedido 5 de mayo de 2022).

[49] «ST-LINK/V2 - ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32 - STMicroelectronics». https://www.st.com/en/development-tools/st-link-v2.html (accedido 5 de mayo de 2022).

[50] «MultiTech Conduit IP67 LoRa Gateway | Outdoor LoRa Deployment». https://www.multitech.com/brands/multiconnect-conduit-ip67 (accedido 5 de mayo de 2022).

[51] «an5406-how-to-build-a-lora-application-with-stm32cubewl-stmicroelectronics.pdf». Accedido: 12 de mayo de 2022. [En línea]. Disponible en: https://www.st.com/resource/en/application_note/an5406-how-to-build-a-lora-application-with-stm32cubewl-stmicroelectronics.pdf

[52] «STM32CubeWL/stm32wlxx_nucleo.h at main · STMicroelectronics/STM32CubeWL», *GitHub*. https://github.com/STMicroelectronics/STM32CubeWL (accedido 14 de mayo de 2022).

[53] «LoRa-STM32WL-DevKIT/SOFTWARE/BB-STM32WLE-WAN/LoRaWAN/Target at main · OLIMEX/LoRa-STM32WL-DevKIT», *GitHub*. https://github.com/OLIMEX/LoRa-STM32WL-DevKIT (accedido 14 de mayo de 2022).

[54] «MAX30205 - Human body temp sensor library | Mbed». https://os.mbed.com/teams/MaximIntegrated/code/MAX30205/ (accedido 15 de mayo de 2022).

[55] «Protocentral_MAX30205/Protocentral_MAX30205.cpp at master · Protocentral/Protocentral_MAX30205», *GitHub*. https://github.com/Protocentral/Protocentral_MAX30205 (accedido 15 de mayo de 2022).

[56] «um1075-stlinkv2-incircuit-debuggerprogrammer-for-stm8-and-stm32-stmicroelectronics.pdf». Accedido: 16 de mayo de 2022. [En línea]. Disponible en: https://www.st.com/resource/en/user_manual/um1075-stlinkv2-incircuit-debuggerprogrammer-for-stm8-and-stm32-stmicroelectronics.pdf

[57] «fever-click-schematic_v100 (1).pdf». Accedido: 7 de junio de 2022. [En línea]. Disponible en: https://download.mikroe.com/documents/add-on-boards/click/fever/fever-click-schematic-v100.pdf

[58] «MQTT Server». https://www.thethingsindustries.com/docs/integrations/mqtt/ (accedido 18 de mayo de 2022).

[59] cowen71, *mqtt-LabVIEW*. 2022. Accedido: 18 de mayo de 2022. [En línea]. Disponible en: https://github.com/cowen71/mqtt-LabVIEW

[60] «JKI JSON Toolkit for LabVIEW - Download - VIPM by JKI». https://www.vipm.io/package/jki_lib_json_serialization/ (accedido 18 de mayo de 2022).

[61] E. artículo *et al.*, «Fast Base64 Encoder/Decoder using LabVIEW», 20 de julio de 2012. https://forums.ni.com/t5/Example-Code/Fast-Base64-Encoder-Decoder-using-LabVIEW/ta-p/3503281 (accedido 19 de mayo de 2022).

Relación de documentos

(X) Memoria 109 páginas

(_) Anexos 17 páginas

La Almunia, a 20 de 06 de 2022

Firmado: Elioenay Pérez López