Trabajo Fin de Grado
Ingeniería de Electrónica, Robótica y Mecatrónica

Mención en Instrumentación Electrónica y Control

Desarrollo firmware de un driver de control para el módulo Sigfox AX-SF10 ANT21-868

Autora: María del Carmen Delgado Venzalá

Tutores: Ramón González Carvajal

Eduardo Hidalgo Fort

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado
Ingeniería de Electrónica, Robótica y Mecatrónica

Desarrollo firmware de un driver de control para el módulo Sigfox AX-SF10 ANT21-868

Autora:

María del Carmen Delgado Venzalá

Tutores:

Ramón González Carvajal

Catedrático de Universidad

Eduardo Hidalgo Fort

Investigador PostDoctoral

Dpto. Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Grado: Desarrollo firmware de un driver de control para el módulo Sigfox AX-SF10 ANT21-868

Autora: María del Carmen Delgado Venzalá

Tutores: Ramón González Carvajal

Eduardo Hidalgo Fort

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mí misma y a ti,
que estas leyendo esto.

AGRADECIMIENTOS

En primer lugar, a mis tutores, Ramón González Carvajal, Eduardo Hidalgo Fort y Jesús Carrión Risquez, por haberme dado la oportunidad de participar en este bonito e ilusionante proyecto y por dejarme aprender un poco más de ellos cada día.

A los amigos que estaban, a los que llegaron, incluso desde la otra punta del mundo, a los que se fueron y a los que quedan, a toda la gente que en algún momento fue importante para mí, por apoyarme en alguna parte de este camino y compartirlo conmigo.

A toda mi familia, que siempre ha confiado en mí, por los valores que me han inculcado desde que era pequeña y he podido potenciar a lo largo de estos años.

Y finalmente, a Nadim, por compartir el sufrimiento del amor por la ingeniería conmigo, apoyarme, aguantarme y levantarme cuando ni siquiera yo podía, enta hyt w ana bhbk.

María del Carmen Delgado Venzalá
Sevilla, 2021

RESUMEN

El trabajo que leerá a continuación presenta el desarrollo firmware de un sistema inalámbrico, cuya funcionalidad será controlar un módulo de la tecnología Sigfox. Para ello se hará uso del módulo AX-SF10 ANT21-868, un transceptor de Sigfox que permite enviar y recibir datos desde el *backend* de Sigfox, donde se pueden visualizar o redirigir a un servidor externo en el cual manipular esos datos.

El desarrollo del firmware para este dispositivo será el núcleo de desarrollo de este trabajo, con el fin de explotar las capacidades que este puede ofrecer. Además, esta aplicación se implementa haciendo uso del microcontrolador STM32L152RET6 mediante el lenguaje de programación C y es compatible con el sistema operativo FreeRTOS.

En este proyecto se pueden encontrar distintas fases de desarrollo, desde conceptos sobre las tecnologías inalámbricas, no solo sobre Sigfox, sino también sobre otras como LoRaWAN o NB-IoT, enfocadas en el *Internet of Things* (IoT), hasta el proceso de desarrollo software de la aplicación y su implementación en el sistema operativo FreeRTOS. Además, se incluyen pruebas del funcionamiento de la aplicación para enviar mensajes al *backend*.

Por último, se estudiarán las posibles líneas futuras de implementación de este tipo de tecnologías, sus ventajas y limitaciones y se expondrán las conclusiones que hayan podido alcanzarse a partir de este estudio.

ABSTRACT

The paper you are about to read introduces you to a wireless system's firmware development, which functionality will be to control and transceiver from the Sigfox technology. Therefore, the module AX-SF10 ANT21-868 will be used, it is a Sigfox transceiver which allows the user to send and receive data from the backend of Sigfox, where you can also display the data or redirect it an external server in which to manage the data.

The firmware development for this device will be the core of this paper in order to exploit the capabilities that this device can offer. Furthermore, this application is implemented making use of the STM32L152RET6 microcontroller using the C programming language and is compatible with the FreeRTOS operating system.

In this paper you can find different development phases, from concepts about wireless technologies, not only about Sigfox, but also about others like LoRaWAN or NB-IoT, which all focus in Internet of Thing (IoT), to the process of the software development of an application and its implementation in the operating system FreeRTOS. In addition, tests of the operation of the application to send messages to the backend are included.

Finally, future work about the implementation of this kind of technology will be studied, its advantages and limitations, and conclusions reached through the project development will be presented.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

NOTACIÓN

IoT	Internet of Things
OS	Operating System
RTOS	Real Time Operating System
LR-WPAN	Low-Rate Wireless Personal Area Network
LPWAN	Low Power Wide Area Network
M2M	Machine-to-Machine
CSS	Chirp Spread Spectrum
ISM	Industrial, Scientific and Medical
GSM	Global System for Mobile communications
LTE	Long Term Evolution
IP	Internet Protocol
API	Application Programming Interface
USB	Universal Serial Bus
MPU	Multiple Process Unit
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
USART	Universal Synchronous and Asynchronous serial Reciever and Transmiter
UART	Universal Asynchronous serial Reciever and Transmiter
UML	Unified Modeling Language
VCP	Virtual COM Port

1. INTRODUCCIÓN

El desarrollo del internet de las cosas hace tiempo que dejó de impactar únicamente al ámbito industrial, donde ya se usaban máquinas y herramientas capaces de conectarse a la red y recopilar y compartir información mediante sensores y aplicaciones.

Ya se pueden encontrar también soluciones IoT en vehículos, el denominado *Smart Road*, sensores capaces de controlar las luces y las señales de tráfico (*Smart city*), frigoríficos capaces de conectarse a internet o a una aplicación móvil (*Smart home*) e incluso la monitorización remota de pacientes altamente infecciosos o con afecciones agudas o crónicas en el ámbito de la salud.

En este trabajo, se propone el desarrollo software de una librería o *driver* capaz de controlar un transceptor de la tecnología Sigfox que permite mandar datos obtenidos mediante sensores externos al *backend* o nube de la propia tecnología, siendo fácilmente accesibles. Además, se implementará en el sistema operativo FreeRTOS (*Real Time Operating System*), el cual permitirá comprobar que la librería es compatible con los OS (*Operating System*) y totalmente funcional.

1.1 Motivación

La motivación de este Trabajo de Fin de Grado resulta de la curiosidad por el auge de las redes inalámbricas (Figura 1-1), de las variadas posibilidades que ofrecen dentro del ámbito del internet de las cosas y de cómo permiten desarrollar el mundo electrónico con una facilidad más grande cada vez.

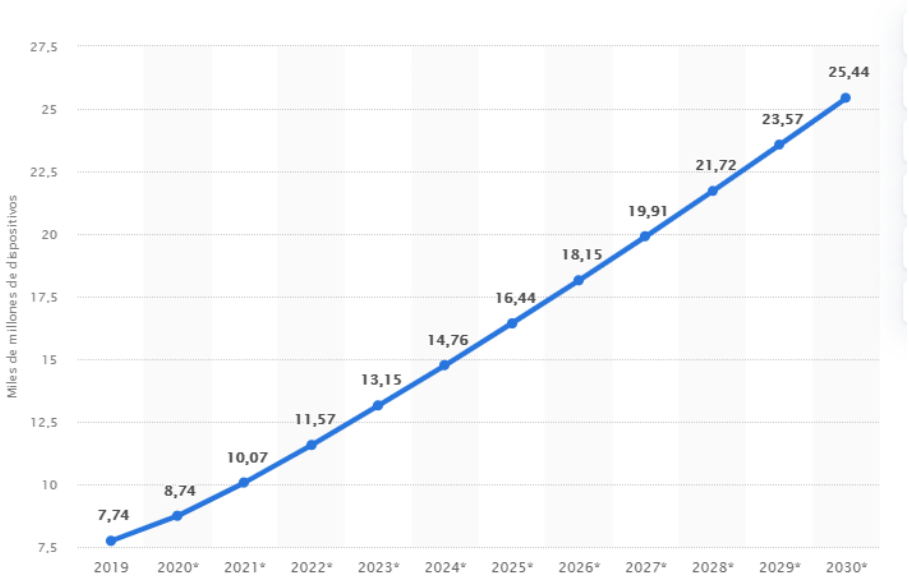


Figura 1-1: Estadística de dispositivos IoT conectados del 2019 al 2030 [1]

Dentro de las redes inalámbricas hay opciones muy variadas con distintas funcionalidades y utilidades de todo tipo, donde el estudio de todas ellas podría ocupar un proyecto completo. Sin embargo, la teoría de poco sirve sin una práctica en la que implementarla, por eso se decide, además de un estudio de las principales redes inalámbricas, desarrollar una aplicación de IoT que, basada en una de las principales redes inalámbricas de IoT, permita transmitir y recibir datos.

1.2 Objetivos

Este proyecto tiene, como objeto, el desarrollo de *driver* para el módem AX-SF10 ANT21-868 modelo AX-SIGFOX ANTSTAMP de la tecnología Sigfox, que permita recibir y enviar al *backend* de SigFox datos recibidos por sensores internos o externos.

Para poder llevar a cabo el desarrollo software se pretende conocer el funcionamiento de la antena de Sigfox, su capacidad de comunicación con el *backend*, y las variadas funcionalidades que presenta y que se pueden encontrar explicadas en el *datasheet*.

Como resultado, se verificará que existe un funcionamiento correcto en cada una de las funciones definidas dentro del driver, comprobando que se reciben los mensajes correspondientes a los datos enviados en el *backend* de SigFox.

Por último, se realizará la implementación de la aplicación en un sistema operativo en tiempo real, en concreto en FreeRTOS, con el fin de comprobar que la funcionalidad dentro de un entorno real es compatible.

1.3 Estructura del proyecto

La estructura de este proyecto está enfocada en adquirir los conocimientos necesarios para programar una aplicación para el *modem* Sigfox AX-SF10 ANT21-868 y poder integrarlo en un OS, de manera que proporcione la mejor solución posible.

Para ello, el desarrollo del proyecto, reflejado en esta memoria, tratará los siguientes aspectos, secuencialmente:

- En primer lugar, se ha hecho una introducción al proyecto, se han explicado la motivación y los objetivos a conseguir en el transcurso del desarrollo del proyecto.

- A continuación, se hará un estudio del estado del arte y se expondrán las ventajas y desventajas que implica el uso de redes inalámbricas en contraposición con redes alámbricas.

- Se presentarán distintas tecnologías actuales de redes inalámbricas, exponiendo sus características y usos principales.

- Se reflexionará sobre porqué Sigfox ofrece una solución mejor que otras tecnologías inalámbricas, como LoRaWAN y NB-IoT, según la aplicacion de IoT objetivo.

- Una vez escogida la tecnología, se profundizará en los componentes hardware, desde el estudio de su arquitectura, hasta las posibilidades que nos ofrece para su implementación; contando con comandos Hayes (AT) para la configuración del módulo Sigfox.

- A continuación, se diseña y desarrolla el *driver*, haciendo uso de los datos proporcionados por el *datasheet* de Sigfox, para implementar todas las funcionalidades que permita el módulo. El desarrollo se presentará mediante diagramas de flujo para facilitar su comprensión.

- Por último, se procederá a implementar el driver en el sistema operativo FreeRTOS y desarrollar tareas con él para comprobar que funciona correctamente dentro de un entorno real. Las tareas se encargarán de medir temperatura mediante un sensor incorporado en el módulo y enviarlo al *backend* de Sigfox, mediante el uso de colas o *queues*.

2. ESTADO DEL ARTE

En este apartado se pretende dar una visión general de las tecnologías de redes inalámbricas que existen actualmente, pero primero se debe conocer qué son las redes inalámbricas y los tipos que hay.

Se conoce como red inalámbrica a un tipo de conexión entre sistemas informáticos que se lleva a cabo mediante diferentes tipos de ondas del espectro electromagnético (radio, microondas, etc.) y por lo tanto no requiere de ningún tipo de cableado entre los sistemas.

Las redes inalámbricas pueden clasificarse de dos maneras distintas en base a los siguientes criterios:

1. Según su alcance:

- WLAN (*Wireless Local Area Network*, en español Red Inalámbrica de Área Local), es el estándar mediante el cual las tecnologías Wi-Fi se estructuran. Gracias al uso de repetidores es posible conectar diferentes dispositivos a grandes distancias mediante ondas de radio.

- WPAN (*Wireless Personal Area Network*, en español Red Inalámbrica de Área Personal), tiene un alcance de metros, por lo que sirve para uno o dos usuarios máximo, que se encuentren próximos entre sí. Este tipo de redes están basadas en una arquitectura similar a la de Bluetooth.

- WMAN (*Wireless Metropolitan Area Network*, en español Red Inalámbrica de Área Metropolitana), redes más complejas y de un alcance mucho mayor que las WLAN, capaces de cubrir hasta 20 kilómetros gracias al uso de ondas de radio o luz infrarroja.

- WWAN (*Wireless Metropolitan Area Network*, en español Red Inalámbrica de Área Amplia). Al igual que las redes WLAN, permite conectar a una gran cantidad de usuarios a la vez, y mediante el uso de tecnologías de telefonía móvil y microondas puede transmitir datos a través de grandes distancias. Las tecnologías principales de este tipo son los conocidos 3G/4G/5G.

2. Según su rango de frecuencias:

- Microondas. Existen dos tipos, en primer lugar, las terrestres, capaces de transmitir señales en un rango de kilómetros haciendo uso de antenas parabólicas en un rango de frecuencias de 1 a 300 GHz. En segundo lugar, las satelitales, donde haciendo uso de un satélite se pueden enviar señales entre varias estaciones, lo que permite un alcance mucho mayor que las terrestres a una velocidad mayor.

- Infrarrojos. Transmisión de datos mediante ondas infrarrojas, que pueden alcanzar frecuencias de entre 300 GHz y 400 THz de velocidad de transmisión de datos, pero necesitan estar en un radio muy cercano, del orden de centímetros a un metro.

- Ondas de radio. Permite emitir y recibir señales mediante ondas en diferentes frecuencias (AM, FM, H, etc.), lo que hace que conforme aumente la distancia, se reduzca su capacidad de transporte.

Algunos de los factores que a tener en cuenta cuando seleccionar el tipo de red que se desea usar son:

- El alcance de la red, si basta con un alcance cercano, o si, por el contrario, se necesita un alcance del orden de kilómetros.
- El ancho de banda de la red, que determina la tasa de envío de datos.
- El uso de energía si se desea implementar una solución de bajo consumo.
- El coste, se verá reflejado en el tipo de infraestructura y de su mantenimiento.
- La seguridad que proporciona cada red al envío de datos.

Las redes alámbricas han existido durante muchos años y seguirán existiendo, incluso las redes inalámbricas suelen estar conectadas a una red alámbrica en algún momento; por lo tanto, la red más utilizada es una híbrida entre redes alámbricas e inalámbricas.

Si se comparan las oportunidades que ofrecen ambas redes, se pueden observar las ventajas y desventajas que cada una presenta.

En el caso de las redes alámbricas, conexiones como Ethernet existen desde mucho antes que el Wi-Fi, lo que hace que la fiabilidad de estas aumente, ya que la probabilidad de perder la conexión es menor.

Además, en lo que respecta a la seguridad, en la conexión alámbrica no habrá datos de transmisión que puedan ser interferidos por terceros ya que suele estar detrás del cortafuegos de la red de área local (LAN), por lo tanto, la seguridad es alta.

Otra ventaja es la velocidad, ya que la conexión mediante cables evita que la comunicación se vea afectada por factores como paredes, techos, distancia, o por interferencias de otros dispositivos electrónicos, permitiendo que la conectividad alámbrica sea más rápida que la conexión inalámbrica. Un ejemplo es el caso de la fibra óptica, la cual alcanzó una velocidad máxima en el 2020 de 5.5 TBps [2].

En contraposición, la mayor desventaja de una red cableada es su propia infraestructura.

El coste de implementación, mantenimiento y reparación de una red cableada es mayor comparada con las redes inalámbricas, cuya implementación y mantenimiento son más sencillo y por lo tanto más barato.

Además, la accesibilidad y movilidad de una red alámbrica es más difícil, las cuales suelen estar soterradas u ocultas en paredes y techos para llegar a los sensores que necesitan conectarse a ellas. Teniendo en cuenta que los sensores son pequeños, pueden colocarse en cualquier lugar de una instalación, lo que resultaría en la imposibilidad de alcanzarlos.

La capacidad de desarrollo de la propia red, y por ende su escalabilidad se ven reducidas ya que necesitan que el hardware sea adquirido, instalado y configurado para cada uno de los usuarios antes de que pueda ser completamente operativo, aumentando los costes y dificultando la planificación.

Sin embargo, la escalabilidad de las redes alámbricas no necesita ninguna instalación hardware, sino una configuración actualizada, permitiendo que esté en funcionamiento de manera rápida y sencilla.

Así mismo, el coste ha ido disminuyendo en los últimos años gracias al desarrollo de la tecnología inalámbrica y al número, cada vez mayor, de fabricantes.

Por lo que se refiere a la conectividad, los dispositivos inalámbricos son más susceptibles a interferencias provenientes de dispositivos electrónicos cercanos que pueden causar pérdidas de conexión o reducir la calidad de la misma.

Hay que mencionar, además, que estas interferencias de señal pueden afectar a la velocidad y consistencia de los datos, y como consecuencia los datos transmitidos pueden no estar disponibles a la velocidad necesaria.

Como la mayoría de las redes cableadas tienden a ser voluminosas y costosas, las implementaciones de IoT inalámbricas son la solución más común. La configuración de una red inalámbrica es un proceso sencillo que permite el funcionamiento de esta en un abrir y cerrar de ojos.

Actualmente, las tecnologías inalámbricas más importantes y cuyas características pueden resultar más interesantes son las siguientes.

2.1 WI-FI

La conexión Wi-Fi es la conexión inalámbrica más conocida mundialmente y la más utilizada por la mayoría de usuarios gracias a su presencia en todos los ámbitos de la vida, desde el ámbito doméstico y educativo hasta el empresarial, etc. Durante muchos años, esta red se ha ido desarrollando, permitiendo la transferencia de datos a una velocidad más alta cada vez y el manejo de cantidades de datos cada vez más grandes (del orden de cientos de megabits por segundo). La principal desventaja de las redes Wi-Fi como solución IoT es el gran consumo de potencia que presentan.

- Estándar: Basado en 802.11n
- Frecuencia: 2,4-5GHz
- Alcance: Aproximadamente 50m.
- Velocidad de transferencia: 150-200Mbps, pero puede alcanzar los 600Mbps.

2.2 Bluetooth

Bluetooth es conocida por ser una de las primeras tecnologías de transmisión de datos de corto alcance y por ser también una de las más establecidas. Muy importante en el ámbito de la electrónica de consumo, gracias principalmente a Bluetooth LE (*Low Energy*), que ofrece un consumo de energía significativamente reducido, y gracias al cual, desde hace algunos años se han podido desarrollar equipos inteligentes de pequeño tamaño que requieran un consumo de energía bajo.

- Estándar: Bluetooth 4.2
- Frecuencia: 2,4GHz (ISM)
- Alcance: 50-150m (Smart)
- Velocidad de transferencia: 1Mbps (Smart)

2.3 ZigBee

ZigBee es una tecnología inalámbrica la cual se centra más en aplicaciones domóticas e industriales. Versiones como ZigBee PRO o ZigBee *Remote Control* se basan en el protocolo IEEE 802.15.4, el cual opera en bandas ISM, concretamente 868 MHz en Europa, 915 MHz en

Norteamérica y 2.4 GHz en todo el mundo y define el acceso a redes LR-WPAN (*Low-Rate Wireless Personal Area Network*).

ZigBee tiene algunas ventajas que pueden interesar a los usuarios como el bajo consumo en sistemas complejos, robustez, alta seguridad, alta escalabilidad y capacidad para soportar un gran número de nodos. Por estas características, es una tecnología bien posicionada para marcar el camino del control inalámbrico y las redes de sensores en aplicaciones IoT y M2M (Machine to machine).

- Estándar: ZigBee basado en IEEE 802.15.4
- Frecuencia: 868MHz (Europa) y 2.4GHz (todo el mundo)
- Alcance: 15-20km
- Velocidad de transferencia: 250kbps

2.4 LoRaWAN

LoRaWAN es el protocolo de comunicación basado en LoRa, estandarizado por LoRa-Alliance, que utiliza bandas ISM (*Industrial, Scientific and Medical*) sin licencia, es decir, 868 MHz en Europa, 915 MHz en Norteamérica y 433 MHz en Asia. La longitud máxima de la carga útil de cada mensaje es de 243 bytes.

LoRa utiliza seis factores de propagación para adaptar la velocidad de datos y el alcance. Un mayor factor de dispersión permite un mayor alcance a costa de una menor velocidad de datos, y viceversa. La velocidad de datos de LoRa oscila entre 300 bps y 50 kbps, dependiendo del factor de dispersión y del ancho de banda del canal. Además, las estaciones base LoRa pueden recibir simultáneamente mensajes transmitidos con diferentes factores de propagación y la comunicación bidireccional [Figura 2-1] la proporciona la modulación de espectro disperso chirp (CSS o *Chirp Spread Spectrum*).

Utilizando LoRaWAN, cada mensaje transmitido por un dispositivo es recibido por todas las estaciones base en el rango. Al explotar esta recepción redundante, LoRaWAN mejora la ratio de mensajes recibidos con éxito. Sin embargo, para conseguirlo se necesitan múltiples estaciones base en el área, lo que puede aumentar el coste de despliegue de la red.

Las recepciones duplicadas resultantes de esta técnica se filtran en el *backend* (servidor de red) que también es capaz de comprobar la seguridad, enviar acuses de recibo al dispositivo emisor y enviar el mensaje al servidor de aplicaciones correspondiente. Además, LoRaWAN aprovecha las múltiples recepciones del mismo mensaje por parte de diferentes estaciones base para localizar los dispositivos finales, evitando así el traspaso en la red LoRaWAN (es decir, si un nodo es móvil o se mueve, no es necesario el traspaso entre las estaciones base).

- Estándar: LoRaWAN
- Frecuencia: 868MHz (Europa)
- Alcance: 2-5km (entorno urbano), 15km (entorno rural)
- Velocidad de transferencia: 0,3-50 kbps (en la banda ISM de 868MHz).

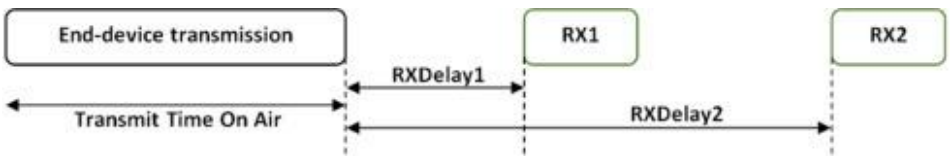


Figura 2-1: Ejemplo de comunicación de LoRaWAN [3]

2.5 NB-IoT

NB-IoT, o *Narrow Band* IoT, es la apuesta de 3GPP para dar respuesta a las necesidades de comunicación IoT y aparece como respuesta al auge de las LPWAN como lo son tecnologías como LoRaWAN o Sigfox entre otras. NB-IoT puede coexistir con GSM (*Global System for Mobile communications*) y LTE (*Long Term Evolution*) en bandas de frecuencia con licencia (por ejemplo, 700 MHz, 800 MHz y 900 MHz). NB-IoT ocupa un ancho de banda de frecuencia de 200 KHz, que corresponde a un bloque de recursos en la transmisión de GSM y LTE.

El protocolo de comunicación NB-IoT se basa en el protocolo LTE, reduciendo sus funcionalidades al mínimo y mejorándolas según las necesidades de las aplicaciones IoT. Por ejemplo, el sistema *backend* LTE se utiliza para enviar información a todos los dispositivos dentro de una célula. Como el sistema *backend* obtiene recursos y consume energía de la batería de cada dispositivo, se mantiene al mínimo. Se ha optimizado para mensajes de datos pequeños y poco frecuentes y evita las características no necesarias para el propósito del IoT, por ejemplo, las mediciones para supervisar la calidad del canal, la incorporación de operadores y la conectividad dual. Por lo tanto, los dispositivos solo requieren una pequeña cantidad de batería, lo que hace que sea rentable.

NB-IoT permite la conectividad de hasta cien mil dispositivos por célula, con la posibilidad de aumentar la capacidad añadiendo más portadoras NB-IoT. La velocidad de datos está limitada a 200 kbps en *downlink* y a 20 kbps en *uplink*. El tamaño máximo de la carga útil de cada mensaje es de 1600 bytes. Además, la tecnología NB-IoT puede alcanzar 10 años de duración de la batería cuando se transmiten 200 bytes al día de media [4].

- Estándar: LTE Cat NB1/2
- Frecuencia: 180KHz
- Alcance: 2km (entorno urbano), 15km (entorno rural)
- Velocidad de transferencia: Hasta 200 kbit/s (*Downlink*) y 20 kbit/s (*Uplink*)

2.6 Sigfox

Sigfox se presenta como un operador de redes LPWAN (*Low Power Wide Area Network*) que ofrece una solución de conectividad IoT de extremo a extremo basada en su propia tecnología patentada.

Sigfox despliega sus propias estaciones equipadas con radios y las conecta a los servidores de *backend* mediante una red basada en IP. Los dispositivos finales se conectan a estas estaciones base utilizando la banda ISM ultra estrecha (*ultra-narrow band*). En la Figura 2-2 se puede ver una representación de la infraestructura de Sigfox.

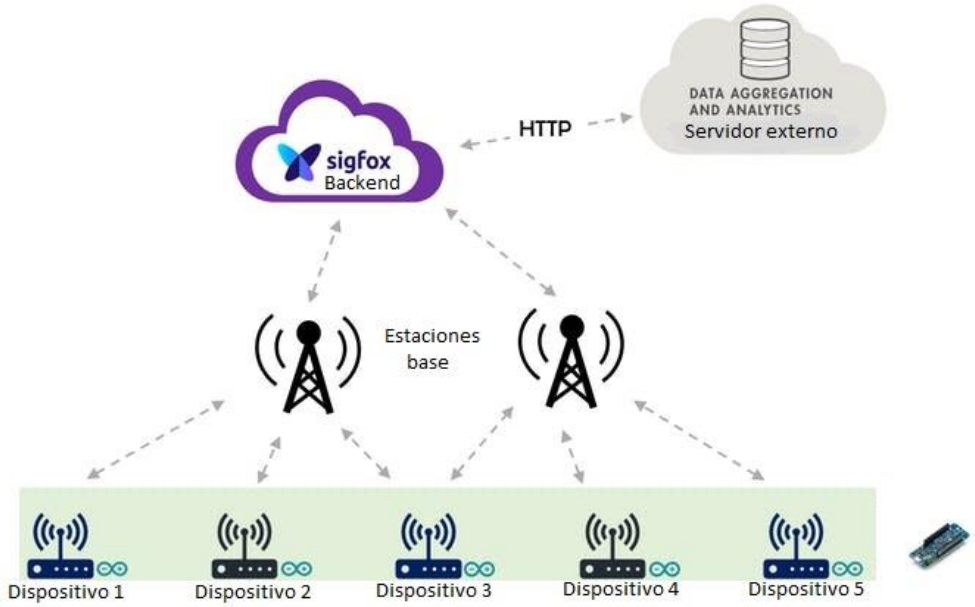


Figura 2-2: Infraestructura de Sigfox [5]

Sigfox utiliza bandas ISM sin licencia, al igual que LoRaWAN, 868 MHz en Europa, 915 MHz en Norteamérica y 433 MHz en Asia. Al emplear *ultra-narrow band*, Sigfox utiliza el ancho de banda de frecuencias de forma eficiente y experimenta niveles de ruido muy bajos, lo que permite que el consumo de energía sea muy bajo, la sensibilidad del receptor sea alta y que el diseño de antena sea de bajo coste, pero permitiendo un rendimiento máximo de sólo 100 bps.

Además, responde a la necesidad de aquellas aplicaciones que funcionan con baterías pequeñas, ya que solo consume 500 nA y es capaz de mantenerse en *stand-by* hasta 15 años alimentado por una batería de 2400 mAh [6].

Inicialmente, Sigfox sólo admitía la comunicación *uplink*, pero posteriormente evolucionó a una tecnología bidireccional con una importante asimetría de enlace. La comunicación *downlink*, es decir, los datos que se envían desde el *backend* a los dispositivos, sólo es posible si se ha producido en primer lugar una comunicación *uplink.*

El número de mensajes que Sigfox permite enviar está limitado a 140 por día. La longitud máxima de la carga útil de cada mensaje que se envía es de 12 bytes. Sin embargo, el número de mensajes que puede recibir, es decir, *downlink*, está limitado a 4 mensajes por día, y longitud máxima de la carga útil es de 8 bytes.

La fiabilidad de la comunicación se garantiza mediante la diversidad de tiempo y frecuencia, así como la duplicación de la transmisión. Cada mensaje se transmite varias veces (tres por defecto) por diferentes canales de frecuencia. Para ello, en Europa, por ejemplo, la banda entre 868,180 MHz y 868,220 MHz se divide en 400 canales ortogonales de 100 Hz (entre ellos, 40 canales están reservados y no se utilizan). Como las estaciones base pueden recibir mensajes simultáneamente en todos los canales, el dispositivo final puede elegir aleatoriamente un canal de frecuencia para transmitir sus mensajes. Esto simplifica el diseño del dispositivo final y reduce su coste.

Todas estas características la convierten en una candidata ideal para aplicaciones M2M como: contadores inteligentes, monitores médicos, dispositivos de seguridad, alumbrado público y sensores ambientales.

- Estándar: Sigfox
- Frecuencia: 868 MHz (Europa)
- Alcance: 40km (ambientes rurales), 10km (ambientes urbanos) [7]
- Velocidad de transferencia: 10-100bps

2.7 ¿Porqué Sigfox?

Dentro de las tecnologías inalámbricas, existen las denominadas LPWAN (*Low Power Wide Area Network*, en español Red de Bajo Consumo y Área Extensa), que como su nombre indica, son redes con poco volumen de datos capaz de transmitirse a kilómetros de distancia. Entre ellas se encuentran Sigfox, LoRaWAN y NB-IoT.

Este tipo de redes permiten conectar decenas o cientos de nodos a cada base, cubriendo territorios muy extensos. Además, son caracterizadas principalmente por un bajo consumo energético y una baja potencia, de manera que los dispositivos finales (sensores) puedan ser alimentados mediante baterías que pueden alcanzar años de vida útil.

Este tipo de tecnologías se desarrollan en torno a tres pilares que las caracteriza:

- Bajo consumo energético.
- Gran alcance o amplia distancia entre los nodos y la antena de recepción.
- Reducida capacidad de trasmisión de datos.

Ha de tenerse en cuenta muchos factores a la hora de elegir la tecnología LPWAN apropiada para una aplicación IoT, incluyendo calidad de servicio, duración de la batería, latencia, escalabilidad, longitud de la carga útil, cobertura, alcance, despliegue y coste.

En el apartado anterior se han mencionado algunas de las muchas tecnologías de redes inalámbricas que existen. Aunque quizás para la mayoría de la sociedad tecnologías como Wi-Fi y Bluetooth resultan más conocidas, en este caso interesa concentrarse en aquellas más adecuadas a las soluciones IoT.

Wi-Fi y Bluetooth no se emplean como soluciones por dos razones principales, el consumo de potencia y el radio de cobertura. Por un lado, el consumo de potencia del Wi-Fi sobrepasa con creces el deseado, ya que se busca un consumo muy pequeño que permita alargar la vida útil de la batería. Por otro lado, Bluetooth permite conexiones de área personal, mientras que para soluciones IoT interesa contar con redes WAN (*Wide Area Network*).

Por último, ambas tecnologías presentan un inconveniente común, no son capaces de conectarse a internet a menos que haya otro dispositivo o *gateway* presente (por ejemplo, en el caso del Wi-Fi el *router*) y en un rango en el que pueda crear una conexión a internet, lo que en muchas aplicaciones podría no ser útil.

En el caso de Zigbee, su principal desventaja es que depende en gran medida del factor humano. El mantenimiento de la red no es llevado a cabo por un operador, si no por quien la explota y, además, es necesario un *gateway* con conexión a internet, por ejemplo, un *router* como en el caso del Wi-Fi.

Los factores de IoT, desarrollados en la Tabla 2-1 y representados visualmente en la Figura 2-3, junto a las diferencias técnicas entre Sigfox, LoRa y NB-IoT determinarán su viabilidad para aplicaciones específicas. Como se discute a lo largo de este apartado, una tecnología no puede servir por igual para todas las aplicaciones de IoT.

Sigfox y LoRa servirán como dispositivos de menor coste, con gran alcancey alta cobertura, una tasa de comunicación de baja frecuencia y una duración de la batería muy larga. A

diferencia de Sigfox que está enfocada para despliegue de redes de área amplia, representada en la Figura 2-4, LoRa también servirá para despliegue de redes de red local y una comunicación fiable cuando los dispositivos se muevan a gran velocidad.

Además, a diferencia de LoRaWAN, Sigfox es una red operada, es decir, tras la suscripción a sus servicios, el usuario no debe preocuparse por su conexión a la red ni el mantenimiento de la infraestructura, ya que de estos se encargan los operadores que la explotan, permitiendo también que sea una solución muy económica, ya que el coste del despliegue de la red es pagado por todos los usuarios suscritos a ella.

Por el contrario, NB-IoT servirá a los mercados de IoT de mayor valor que están dispuestos a pagar por una latencia muy baja y una alta calidad de servicio, mientras que Sigfox ofrece una suscripción a su red de en torno 3€ al año [8], mucho inferior al coste que supone NB-IoT, cuya suscripción se encuentra en 16€ y 25€ anuales, dependiendo de la asignación mensual de datos [9]. LoRa ofrece sus servicios de manera gratuita, pero el mantenimiento de la red y el *gateway* con conexión a internet corren a expensas del usuario.

	Sigfox	**LoRaWAN**	**NB-IoT**
Modulation	BPSK	CSS	QPSK
Frequency	Unlicensed ISM bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia)	Unlicensed ISM bands (868 MHz in Europe, 915 MHz in North America, and 433 MHz in Asia)	Licensed LTE frequency bands
Bandwidth	100 Hz	250 kHz and 125 kHz	200 kHz
Maximum data rate	100 bps	50 kbps	200 kbps
Bidirectional	Limited	Yes	Yes
Maximum messages/day	140 (UL), 4 (DL)	Unlimited	Unlimited
Maximum payload length	12 bytes (UL), 8 bytes (DL)	243 bytes	1600 bytes
Range	10 km (urban), 40 km (rural)	5 km (urban), 20 km (rural)	1 km (urban), 10 km (rural)
Interference immunity	Very high	Very high	Low
Authentication & encryption	Not supported	Yes (AES 128b)	Yes (LTE encryption)
Adaptive data rate	No	Yes	No
Handover	End-devices do not join a single base station	End-devices do not join a single base station	End-devices join a single base station
Allow private network	No	Yes	No

	Sigfox	**LoRaWAN**	**NB-IoT**
Standardization	Sigfox company is collaborating with ETSI on the standardization of Sigfox-based network	LoRa-Alliance	3GPP
Spectrum cost	Free	Free	>500M€/MHz
Deployment cost	>4000€/base station	>100€/gateway >1000€/base station	>15000€/base station
End-device cost	<2€	3-5€	5-6€

Tabla 2-1: Resumen de tecnologías LPWAN (Sigfox, LoRaWAN y NB-IoT) [3]

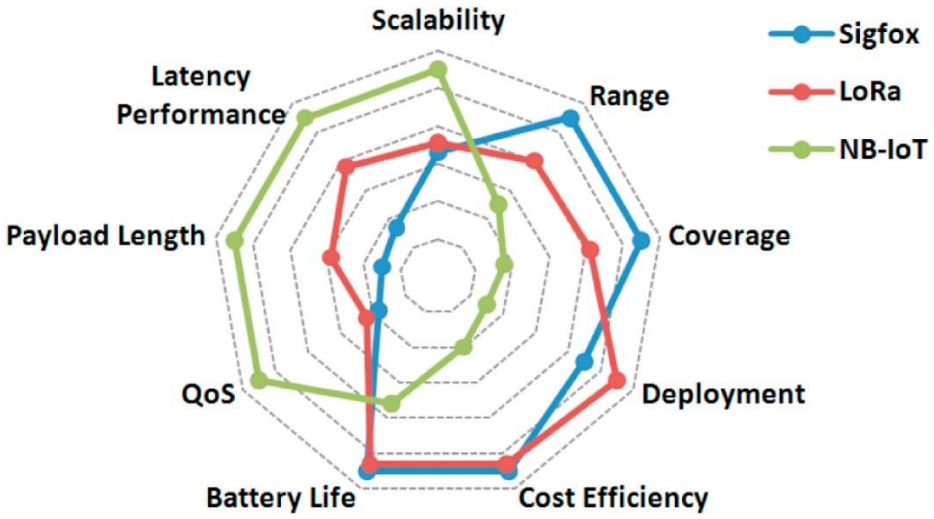


Figura 2-3: Comparativa de los factores IoT para Sigfox, LoRaWAN y NB-IoT [5]

Si se observa la Figura 2-3, Sigfox proporciona una solución mejor en aquellos aspectos en las cuales NB-IoT se queda corto, como pueden ser el rango y la cobertura o el coste. Por ello, podemos entender que Sigfox se presenta como una opción alternativa a lo que hasta ahora ofrecía NB-IoT, dando solución a zonas de sombra de esta última, y permitiendo la convivencia entre ambas.

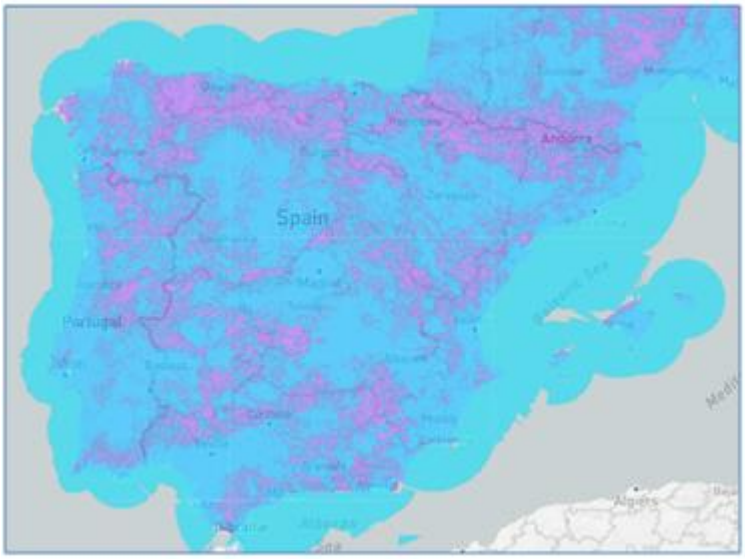


Figura 2-4: Cobertura de Sigfox en España [7]

En conclusión, se decide utilizar Sigfox, ya que supone la solución ideal para proyectos en zonas rurales y al aire libre, no solo por su alcance y cobertura (Figura 2-4), que permite la comunicación con el *backend* incluso en zonas más alejadas de los núcleos urbanos, sino también por su independencia del factor humado. Además, gracias a su infraestructura y a su bajo consumo, permite reducir costes de mantenimiento.

3. IMPLEMENTACIÓN DE DISPOSITIVOS HARDWARE

Para poder llevar a cabo el desarrollo del driver que aquí nos ocupa, se ha hecho uso de un *setup* compuesto por dispositivos hardware que ha permitido el desarrollo y posterior implementación de todo lo relativo al desarrollo software de este proyecto.

Por estas razones se considera oportuno profundizar en cada uno de los tres módulos que conforman este *setup*, la conexión de todos ellos, la funcionalidad que cada uno desempeña y sus características.

3.1 Placa de desarrollo Núcleo STM32

En primer lugar, el microcontrolador que se usará durante el desarrollo del proyecto y en el cual se cargarán todos los programas es el STM32L152RE del fabricante ST, cuya placa de desarrollo es la NUCLEO-L152RE, la cual se muestra en la Figura 3-1.

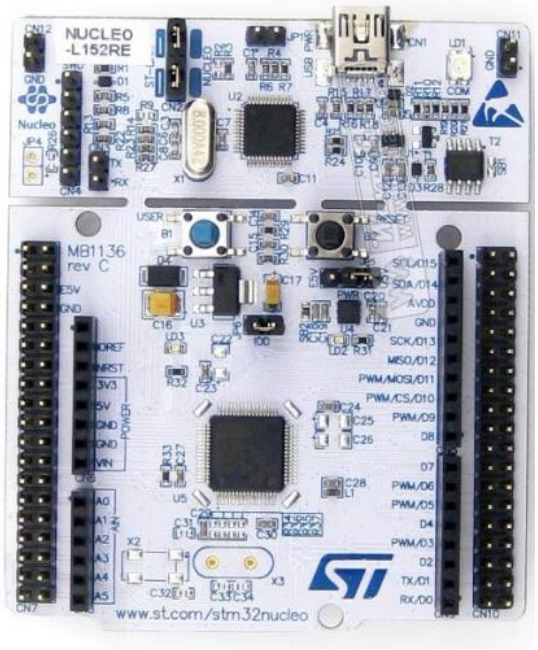


Figura 3-1: Vista de planta de la placa de desarrollo STM32L152RE [8]

Caracterizado por su ultra bajo consumo, incorpora las siguientes características [9]:

- Bus serie universal (USB)
- Núcleo procesador (CPU) de 32 bits que funciona a una frecuencia de 32 MHz
- Una unidad de protección de memoria (MPU)
- Memorias integradas de alta velocidad (memoria Flash de hasta 512 Kbytes y RAM de hasta 80 Kbytes)
- Alimentación de 3.3V y 5V
- Una amplia gama de I/O y periféricos conectados a dos buses APB.

- Dos amplificadores operacionales
- Un CAD de 12 bits
- Dos CDA
- Dos comparadores de muy bajo consumo
- Un temporizador de 32 bits
- Seis temporizadores de 16 bits
- Dos temporizadores *watch-dog*
- Dos temporizadores básicos, que pueden utilizarse como referencia de tiempos.

En lo referente a las características de comunicación de periféricos, contiene interfaces de comunicación estándar y avanzadas [9]:

- Dos I2C
- Tres SPI de 16Mbit/s
- Dos I2S
- Tres USART
- Dos UART
- Un USB.

En las figuras que se muestran a continuación se pueden observar los canales 7 (Figura 3-2), 8 (Figura 3-3), 9 (Figura 3-4) y 10 (Figura 3-5), con sus respectivos pines y las funcionalidades que cada uno de ellos es capaz de soportar.

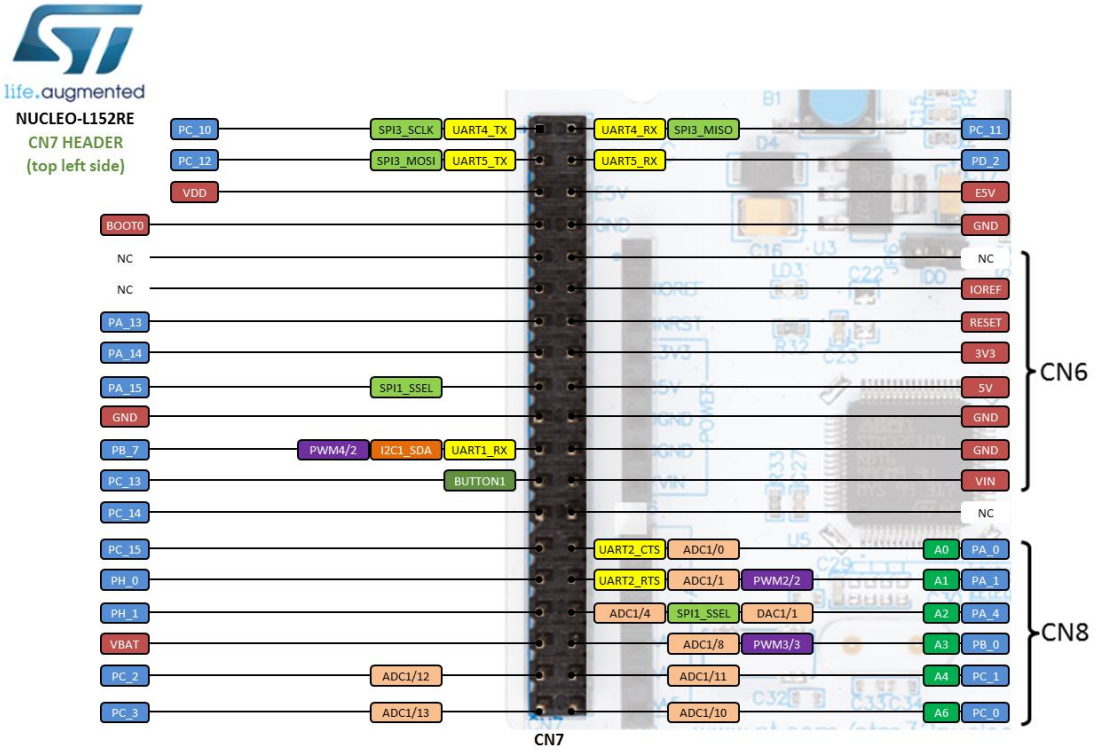


Figura 3-2: Pin Out CN7 placa STM32L152RE [9]

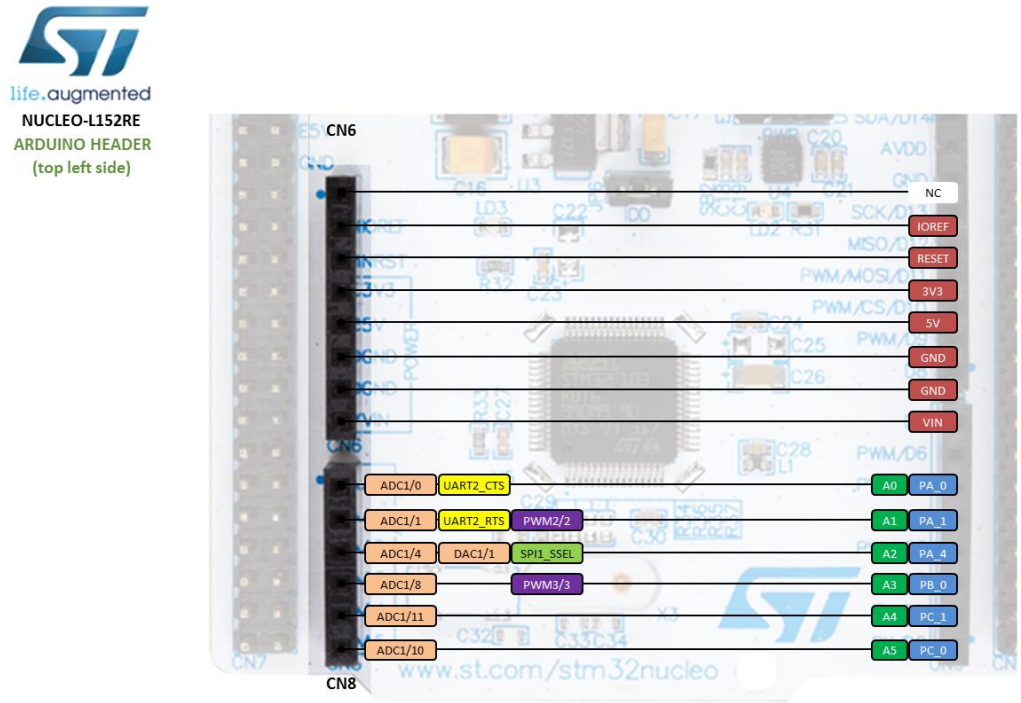


Figura 3-3: Pin Out CN8 placa STM32L152RE [9]

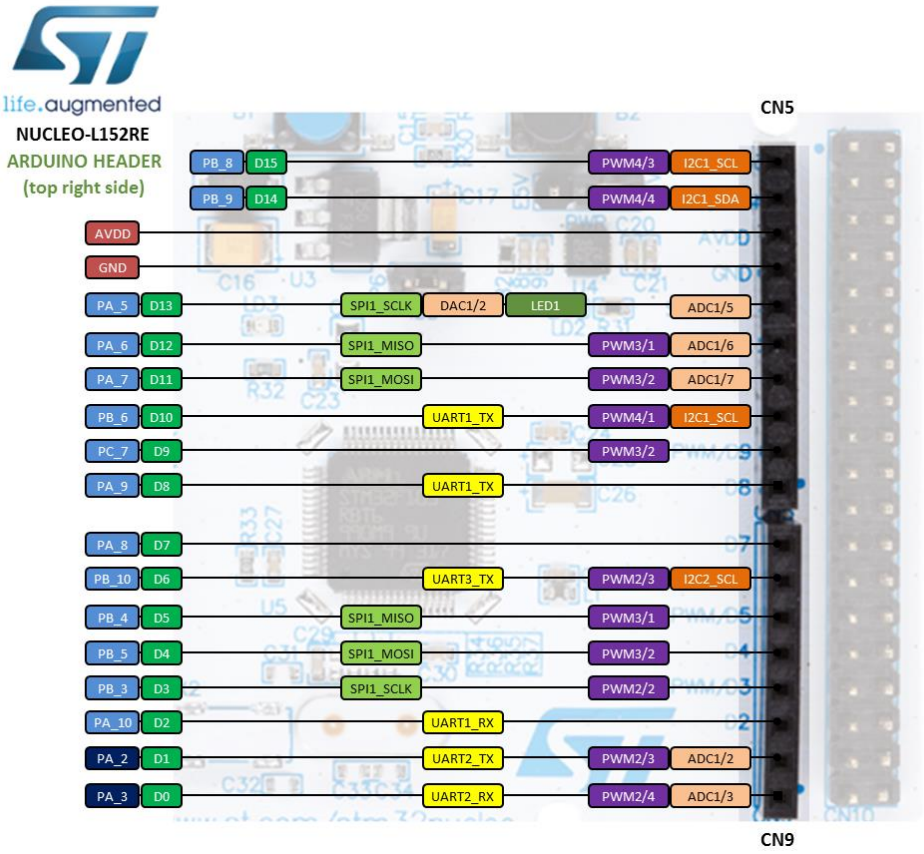


Figura 3-4: Pin Out CN9 placa STM32L152RE [9]

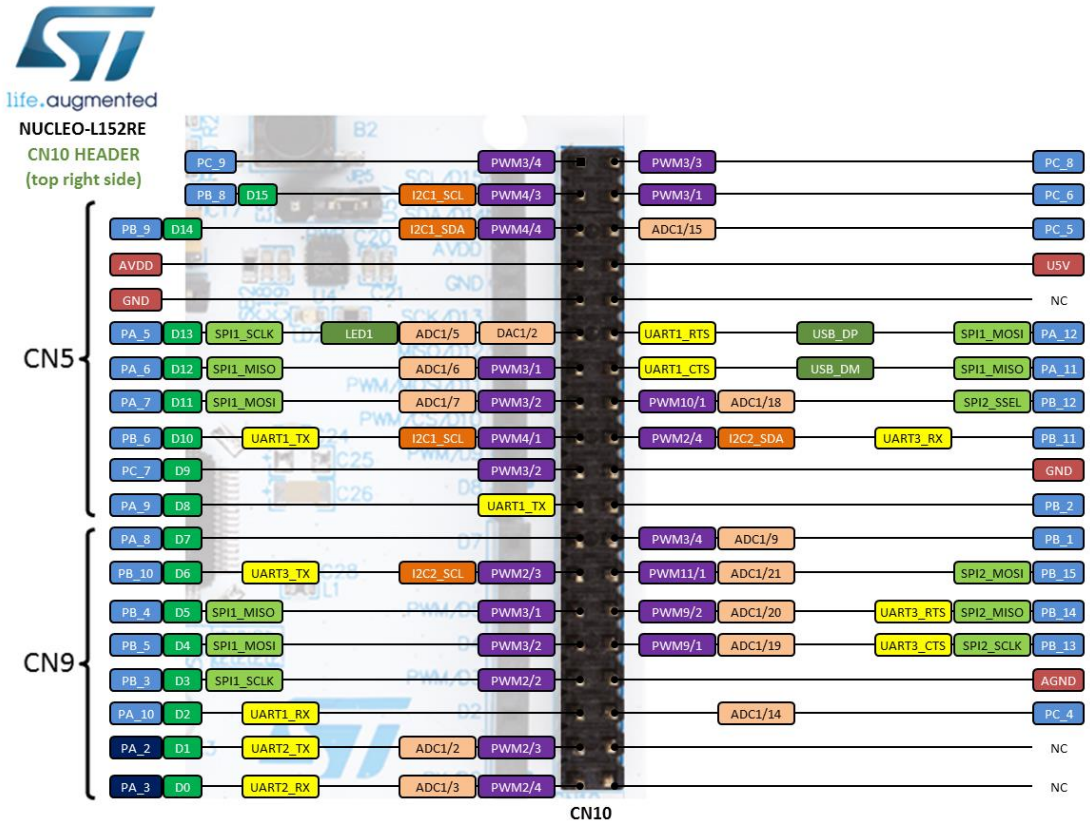


Figura 3-5: Pin Out CN10 placa STM32L152RE [9]

El microcontrolador es indispensable a la hora de desarrollar un proyecto de estas características, ya que es el encargado de soportar y ejecutar tanto la API como el sistema operativo que se implemente, de manera que las funciones descritas en él se desarrollen correctamente.

En este caso, se ha desarrollado la API en el entorno IAR Embedded Workbench IDE en lenguaje C, y el sistema operativo utilizado será FreeRTOS.

3.2 Transceptor Sigfox AX-SIGFOX ANTSTAMP

El módulo de Sigfox que se utilizará durante todo el proyecto será el AXSF-10 ANT21-868, modelo AX-SIGFOX ANTSTAMP con un chip antena de -5dB integrado, representado en la Figura 3-6.

Las principales características de este módulo antena son:

- Bidireccionalidad de comunicación, que permite enviar y recibir datos hacia y desde el *backend*.
- Control mediante comandos Hayes (AT), permiten la comunicación entre el usuario y el módulo.
- Consumo ultra-bajo, siendo el consumo de 0.5mA en modo *Standby*, 1.6uA en modo *Sleep* y 500nA en modo *Deep Sleep.*
- Suscripción Sigfox de dos años incluida en la compra del módulo
- Alimentación de 1.8V a 3.3V.
- Sensor de temperatura, se utiliza para solicitar datos de temperatura al módulo y enviarlos posteriormente al *backend*, como se puede ver más adelante en la implementación de FreeRTOS.

- Sensor de tensión.
- Comunicación dentro de la banda de frecuencia de 868MHz.
- 10 pines GPIO, de los cuales, el pin GPIO9 (pin 18), permite despertar al módulo del modo *Deep Sleep.*
- Velocidad de transmisión de datos de 100bps.

Como ya se ha mencionado anteriormente, los mensajes que Sigfox soporta tendrán un máximo de 12 bytes y solo permite transmitir 140 mensajes al día.

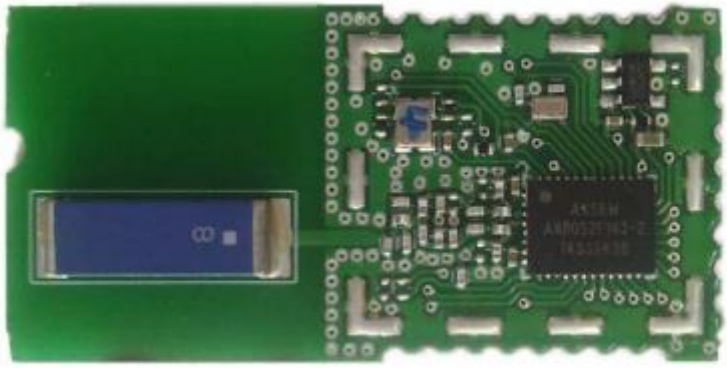


Figura 3-6: Módulo AX-SIGFOX ANTSTAMP [10]

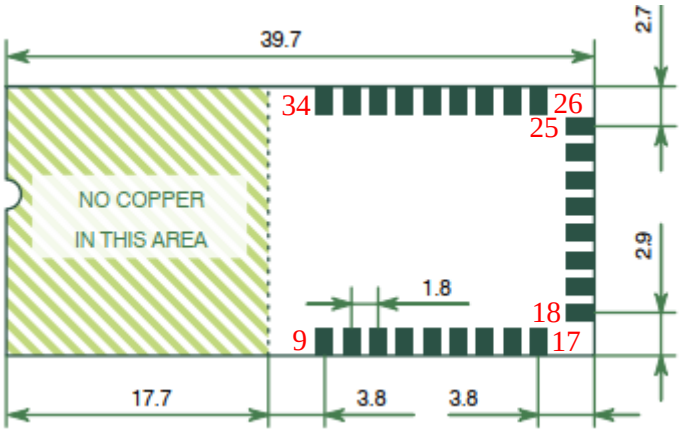


Figura 3-7: Dimensiones y pines módulo AX-SIGFOX ANTSTAMP [10]

9	NC	Do not connect
10	GPIO8	General purpose IO
11	GPIO7	General purpose IO, selectable SPI functionality (MISO)
12	GPIO6	General purpose IO, selectable SPI functionality (MOSI)
13	GPIO5	General purpose IO, selectable SPI functionality (SCK)
14	GPIO4	General purpose IO, selectable ∑Δ DAC functionality, selectable clock functionality
15	CPU_LED	Module activity status, enabled whenever the module is running
16	RADIO_LED	Radio activity status
17	VTCXO	TCXO enable (used to control the on-board TCXO)
18	GPIO9	General purpose IO and wake-up from deepsleep
19	UART_TX	UART used to communicate with the module at a bitrate of 9600 baud, no parity, 8 data bits and one stop bit.

20	UART_RX	UART used to communicate with the module at a bitrate of 9600 baud, no parity, 8 data bits and one stop bit.
21	RX_LED	Radio receive activity status
22	TX_LED	Radio transmit activity status
23	NC	Do not connect
24	NC	Do not connect
25	VDD	Power Supply
26	GND	Ground
27	RESET_N	Optional reset (active low). Do not connect the pin if not used.
28	GND	Ground
29	GPIO0	General purpose IO, selectable ADC functionality, selectable ∑Δ DAC functionality, selectable clock functionality
30	GPIO1	General purpose IO, selectable ADC functionality
31	GPIO2	General purpose IO, selectable ADC functionality
32	NC	Do not connect
33	NC	Do not connect
34	GPIO3	General purpose IO, selectable ADC functionality

Tabla 3-1: Pines del módulo AX-SIGFOX ANTSTAMP [10]

Además, se realizó el diseño de una placa para acceder a los pines del transceptor (Figura 3-8). Este diseño se hizo con el fin de implementarlo en este proyecto, pero se encuentra fuera del contenido del trabajo.

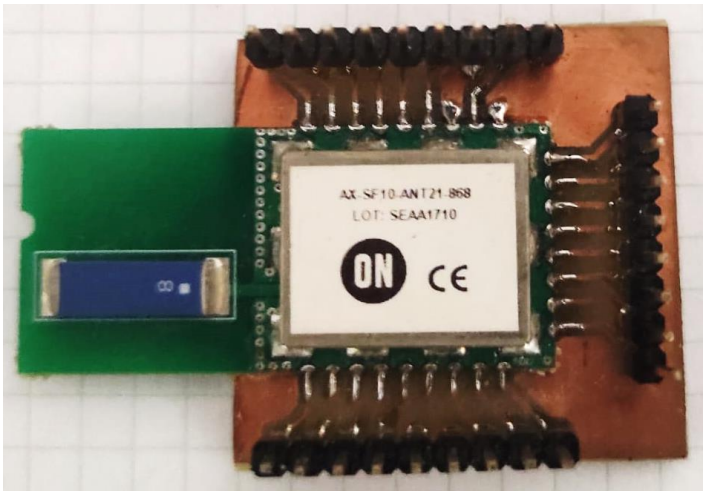


Figura 3-8: Vista final, pines soldados de módulo AX-SIGFOX ANTSTAMP

Haciendo uso del esquema de pines del *datasheet* (Figura 3-7) y de las funcionalidades de cada uno enumerados en la Tabla 3-1, se pueden identificar fácilmente en el módulo de pines ya soldados (Figura 3-8).

Es importante mencionar también el uso del *backend* de Sigfox como punto final de la comunicación, que recibirá los datos enviados por el módem y los presentará a través de su página web. Estos mensajes se pueden reenviar, mediante *callback*, a servicios externos para la generación de estadísticas y alarmas en tiempo real.

3.3 Módulo TTL FT4232H-56Q Mini Module

Como último componente del *setup* se tiene el módulo FT4232H-56Q Mini, representado en la Figura 3-9.

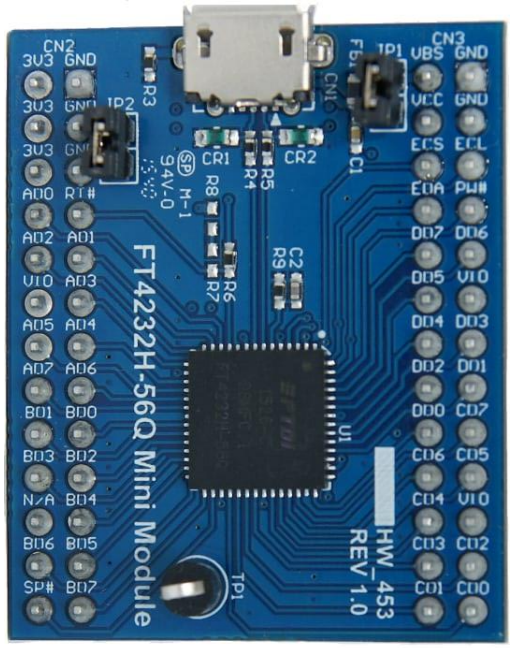


Figura 3-9: Vista superior, pines y electrónica, módulo FT4232H-56Q Mini Module [11]

Este módulo es un módulo de desarrollo *USB-serial/FIFO* de la gama de productos FTDI que utiliza el chip puente FT4232H-56Q USB *Hi-Speed* de cuatro puertos que maneja toda la señalización y protocolos USB 2.0 de alta velocidad (480Mb/s) a UART/MPSSE IC.

Una vez conectado el módulo al ordenador, este será reconocido como 4 puertos COM virtuales (VCP), cuyas interfaces podrán ser configuradas independientemente como interfaces serie asíncronas o síncronas (Figura 3-10). Los pines de cada puerto se reconocen fácilmente en el grabado de la placa de la Figura 3-9, de manera que cada uno de los puertos se identifica con las letras A, B, C y D respectivamente.

Figura 3-10: Funcionamiento del móduloFT4232H-56Q Mini [12]

De todos los pines que presenta cada puerto, interesa quedarse con X00 y X01, ya que son los pines de transmisión (TX) y recepción (RX), respectivamente, de cada puerto.

Además, que interese conocer, el módulo presenta tres pines de alimentación a 3.3V y tres pines a tierra Figura 3-9.

La finalidad de este módulo es principalmente la visualización de la comunicación existente. En primer lugar, comprobar que el módulo Sigfox funciona correctamente, permitiendo una comunicación entre bidireccional entre el usuario y la placa, de manera que el usuario puede mandar distintos comandos y observar la respuesta

del módulo. En segundo lugar, interceptar los mensajes compartidos entre el microcontrolador y el módulo una vez implementada la aplicación.

Para acceder a estos mensajes, se utiliza el software Docklight, el cual es una herramienta de prueba, análisis y simulación de protocolos de comunicación en serie. Permite monitorizar la comunicación entre dos dispositivos serie o probar la comunicación serie de un único dispositivo, el cual funciona como monitor de puerto serie.

3.4 Interconexión de módulos hardware

A continuación, se realiza una breve explicación de como se ha desarrollado el *setup* hardware que permite la comunicación entre los distintos módulos desarrollados anteriormente.

3.4.1 Setup inicial

Un *setup* secundario fue necesario al comienzo del desarrollo del proyecto. Este permitía una comunicación únicamente entre el usuario y el transceptor de Sigfox mediante el uso de la placa TTL, conectando las placas como se muestra en la Figura 3-11: Conexión entre los pines de los módulos del setup inicialFigura 3-11:

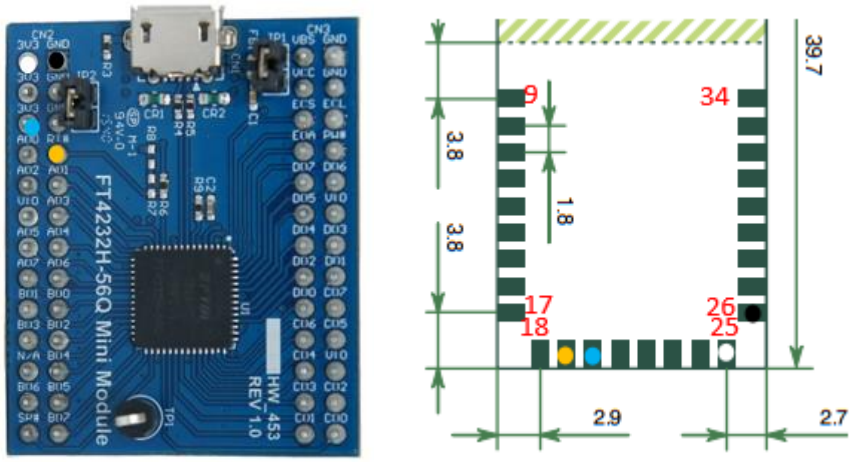


Figura 3-11: Conexión entre los pines de los módulos del setup inicial

Siendo el pin blanco alimentación 3.3V, el pin negro GND, el pin amarillo TX en el módulo Sigfox – RX en el módulo TTL y el pin azul RX en el módulo Sigfox – TX en el módulo TTL.

Figura 3-12: Diagrama de la implementación del setup inicial

La implementación mostrada en la Figura 3-12 se utiliza para comprobar que el transceptor funciona correctamente y para poder familiarizarse con los comandos AT y con los formatos de comunicación de este módulo, con el fin de implementar correctamente la API.

3.4.2 Setup principal

El *setup* principal que se ha llevado a cabo para el desarrollo de la aplicación es aquel compuesto por las tres placas. El microcontrolador como núcleo encargado del procesamiento de lo que se envía a y se recibe del módulo de Sigfox, el módulo Sigfoxencargado de recibir y enviar datos al *backend* y el TTL interceptando todos los mensajes como modo de depuración.

Esta conexión se realiza como se muestra en la Figura 3-13:

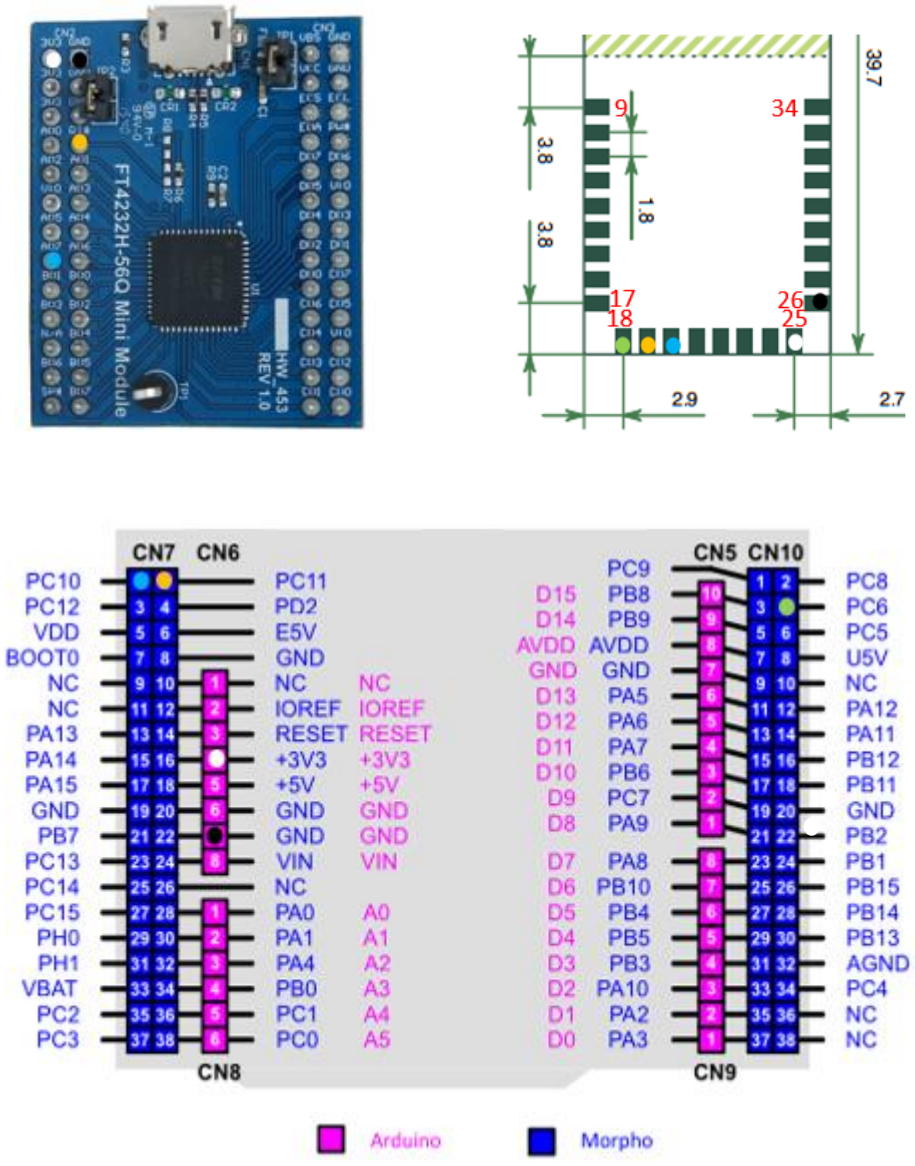


Figura 3-13: Conexión entre los pines de los módulos del setup primario

Con la disposición mostrada en la Figura 3-13, las tres placas comparten alimentación de 3.3V (blanco) y tierra (negro). Además, el pin GPIO9 de la placa Sigfox que se encarga de despertarla está conectado con el pin PC6, GPIO del microcontrolador (verde). Por último, los canales de transmisión (TX) (amarillo) y recepción (RX) (azul) de la placa Sigfox están conectados con el receptor (RX) (amarillo) y el transmisor (TX) (azul) del microcontrolador respectivamente. El módulo TTL se conecta en paralelo a la conexión entre el microcontrolador y el transceptor independientemente a dos pines X01 de dos puertos diferentes, como se muestra en la Figura 3-14, ya que interesa recibir todos

los mensajes, tanto los enviados por el microcontrolador como aquellos enviados por el módulo Sigfox.

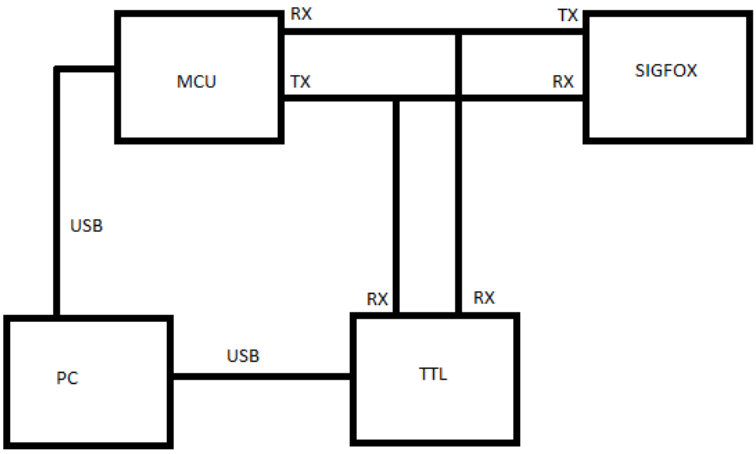


Figura 3-14: Diagrama de la implementación del setup principal

Aquí se puede observar la imagen del *setup* real y las conexiones entre cada módulo.

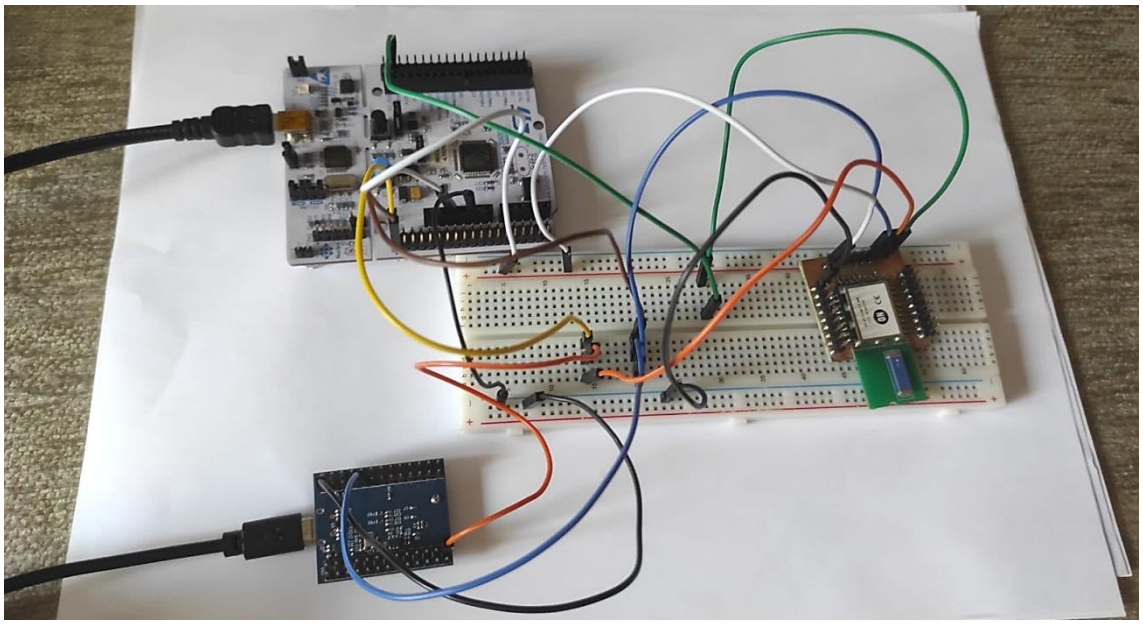


Figura 3-15: Setup principal implementado en la vida real

Nota: El módulo Sigfox que se observa en la Figura 3-15 no está “pinchado” en la *protoboard*, si no que se ha colocado sobre ella para que la visión del *setup* sea mejor.

4. DISEÑO Y DESARROLLO DEL DRIVER AXSF

Haciendo uso de los dispositivos hardware introducidos en la sección anterior y de las aplicaciones utilizadas, como el monitor puerto serie Docklight y el entorno de desarrollo IAR Embedded Workbench se pretende cumplir uno de los objetivos marcados en la realización de este proyecto: programar una aplicación para el módulo AXSF-10 ANT21-868.

Esta sección pretende incidir en la metodología seguida para encontrar la solución que se presentará finalmente, en base a los siguientes pasos a seguir:

- **Definición del problema** a resolver y definición de las especificaciones de la solución que se desea obtener.

- **Diseño de la solución**, valorando todas las posibilidades extraídas del estudio de la definición del problema, y los datos obtenidos del *datasheet* del módulo.

- **Desarrollo de la solución**, una vez llegue a una visión final del resultado a obtener.

- **Pruebas** y **verificación de funcionamiento**, que se tratarán en la siguiente sección.

4.1 Definición del problema

Se desea conseguir un *driver* para el módulo AX-SF10 ANT21-868 de Sigfox, capaz de controlar las principales funcionalidades de este, con el fin de poder integrarlo en futuros proyectos de recepción y envío de datos inalámbrico.

4.1.1 Especificaciones

Durante el desarrollo del software se tienen en cuenta algunas especificaciones entorno al método de desarrollo:

- El código estará escrito en lenguaje C e implementado en el entorno de desarrollo IAR Embedded Workbench.

- Se hará uso de las librerías proporcionadas previamente por el GIE, como es el caso de la librería de la UART o los GPIO. Además, se hará uso si es necesario de las librerías “string.h”, “stdarg.h” y “stdio.h”.

- El código estará integrado por un fichero .c en el que se desarrollen íntegramente las funciones y un fichero .h en el que se declaren las funciones, los mensajes de error, comandos y estructuras varias.

- La solución propuesta debe tener en cuenta que el límite de mensajes permitidos por día, en base a la suscripción elegida, es de 140 mensajes, y el tamaño máximo es de 12 bytes.

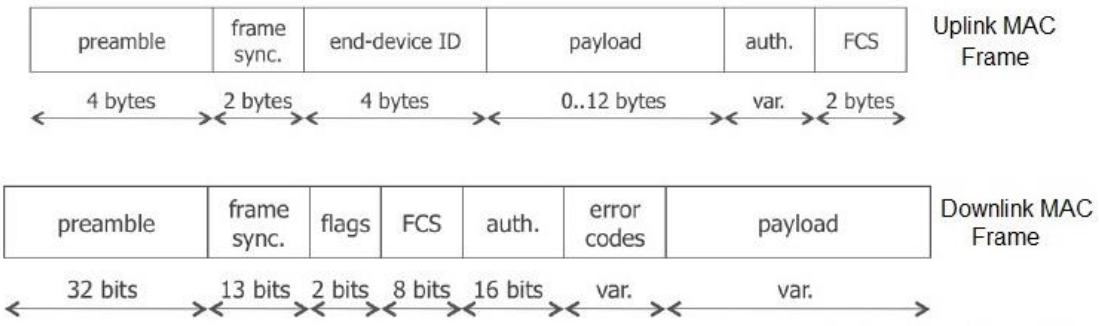


Figura 4-1: Estructura de los mensajes de Sigfox [5]

- La comunicación será bidireccional, entre el módulo AXSF-10 ANT21-868 y el *backend* de Sigfox, de manera que permita recibir actualizaciones de registros, siempre que se especifique o el usuario lo requiera.

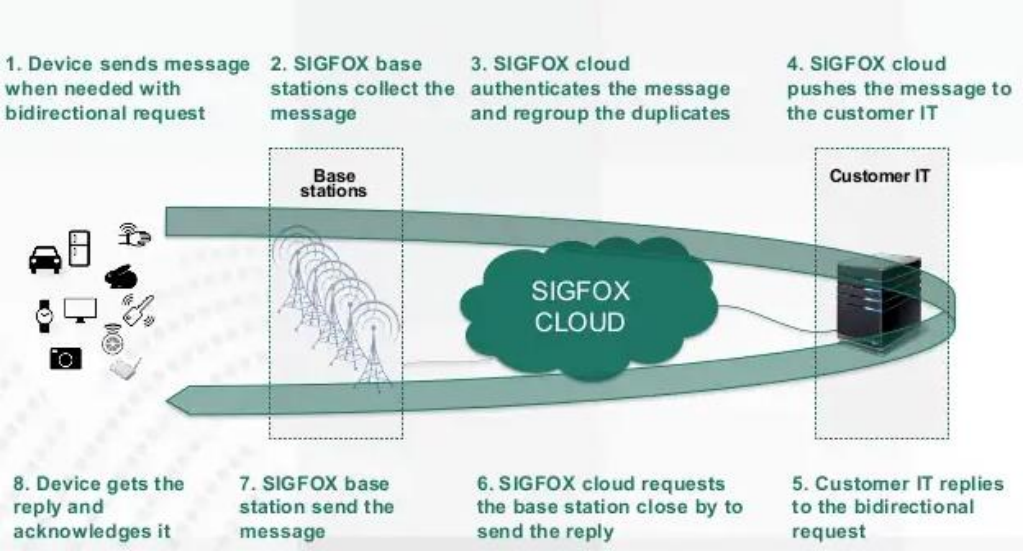


Figura 4-2: Funcionamiento de la comunicación bidireccional de Sigfox [5]

4.2 Diseño de la solución

Si se tiene en cuenta lo planteado en la definición del problema, una de las soluciones más importantes debe ser la durabilidad de vida útil de la batería. Las principales soluciones IoT presentan consumo de energía bajo, entre ellas se encuentra Sigfox, en concreto nuestro módulo que proporciona tres estados de funcionamiento, cada uno con un consumo de energía diferente [10]:

- Deep sleep mode current: 500 nA
- Sleep mode current: 1.6 µA
- Standby mode current: 0.5 mA

Para ello debe ser posible cambiar los modos de funcionamiento de manera que mientras el módulo no esté en funcionamiento, es decir, recibiendo o enviando datos o realizando alguna otra funcionalidad, espere en el modo de *Deep Sleep*, donde el consumo es mucho inferior a los otros dos, y cuando sea necesario despierte y pase al modo *Stand-by* durante el tiempo que tarda en realizar la acción.

También debe ser posible implementar una comunicación bidireccional, como se ha comentado anteriormente en las especificaciones, de manera que sea capaz de mandar datos al *backend* y, además, poder solicitar información desde este cuando lo requiera, como, por ejemplo, una alarma cada medianoche.

En adición a esta funcionalidad de bidireccionalidad, debe ser posible acceder a los registros del módulo en caso de necesitar una actualización desde el *backend*.

Por otro lado, en caso de producirse un error debe ser posible resetear el módulo, para no perder la información de la configuración, también debe poderse guardar en la memoria flash del módulo.

Por último, puede ser necesario en ocasiones comprobar el correcto funcionamiento del módulo de Sigfox.

4.3 Desarrollo de la solución

Para tener una idea de cómo proceder con el desarrollo del driver AXSF, lo primordial es estudiar el *datasheet* del transceptor de Sigfox, entender todas las funcionalidades que permite implementar mediante comandos AT (Tabla 4-1) y considerar todos aquellos comandos que puedan permitirnos alcanzar la solución final.

Este apartado contiene detalles exclusivos del BSW (*Basic Software*), y no de la programación; utilizando diagramas UML (*Unified Modeling Language*) para describir la solución propuesta cuando se considere necesario.

Command	**Name**	**Description**
AT	Dummy Command	Just returns 'OK' and does nothing else. Can be used to check communication.
AT$SB=bit[,bit]	Send Bit	Send a bit status (0 or 1). Optional bit flag indicates if AX-SIGFOX module should receive a downlink frame.
AT$SF=frame[,bit]	Send Frame	Send payload data, 1 to 12 bytes. Optional bit flag indicates if AX-SIGFOX module should receive a downlink frame.
AT$SO	Manually send out of band message	Send the out-of-band message.
ATSuint?	Get Register	Query a specific configuration register's value. See chapter "Registers" for a list of registers.
ATSuint=uint	Set Register	Change a configuration register.
AT$IF=uint	Set TX Frequency	Set the output carrier macro channel for Sigfox frames.
AT$IF?	Get TX Frequency	Get the currently chosen TX frequency.
AT$DR=uint	Set RX Frequency	Set the reception carrier macro channel for Sigfox frames.
AT$CW=uint,bit[,uint_opt]	Continuous Wave	To run emission tests for Sigfox certification it is necessary to send a continuous wave, i.e. just the base frequency without any modula tion. Parameters: **Name** **Range** **Description** Frequency 800000000-999999999, 0 Continuous wave frequency in Hz. Use 868130000 for Sigfox or 0 to keep previous frequency. Mode 0, 1 Enable or disable carrier wave. Power 0-14 dBm of signal | Default: 14
AT$CB=uint_opt,bit	Test Mode: TX constant byte	For emission testing it is useful to send a specific bit pattern. The first parameter specifies the byte to send. Use '-1' for a (pseudo-)random pattern. Parameters: **Name** **Range** **Decsription** Pattern 0-255, -1 Byte to send. Use '-1' for a (pseudo-)random pattern. Mode 0, 1 Enable or disable pattern test mode

AT$T?	Get Temperature	Measure internal temperature and return it in 1/10th of a degree Celsius.
AT$V?	Get Voltages	Return current voltage and voltage measured during the last transmission in mV.
AT$I=uint	Information	Display various product information: 0: Software Name & Version Example Response: AX-Sigfox 1.0.6-ETSI 1: Contact Details Example Response: support@axsem.com 2: Silicon revision lower byte Example Response: 8F 3: Silicon revision upper byte Example Response: 00 4: Major Firmware Version Example Response: 1 5: Minor Firmware Version Example Response: 0 6: Firmware Revision Example Response: 3 7: Firmware Variant (Frequency Band etc. (EU/US)) Example Response: ETSI 8: Firmware VCS Version Example Response: v1.0.2-36 9: SIGFOX Library Version Example Response: DL0-1.4 10: Device ID Example Response: 00012345 11: PAC Example Response: 0123456789ABCDEF
AT$P=uint	Set Power Mode	To conserve power, the AX-SIGFOX module can be put to sleep manually. Depending on power mode, you will be responsible for waking up the AX-SIGFOX module again! 0: software reset (settings will be reset to values in flash) 1: sleep (send a break to wake up) 2: deep sleep (toggle GPIO9 or RESET_N pin to wake up; the AX-SIGFOX module is not running and all settings will be reset!)
AT$WR	Save Config	Write all settings to flash (RX/TX frequencies, registers) so that they survive reset/deep sleep or loss of power. Use AT$P=0 to reset the AX-SIGFOX module and load settings from flash.
AT:Pn?	Get GPIO Pin	Return the setting of the GPIO Pin *n*; *n* can range from 0 to 9. A character string is returned describing the mode of the pin, fol lowed by the actual value. If the pin is configured as analog pin, then the voltage (range 0…1 V) is returned. The mode characters have the following meaning: **Mode Description** 0 Pin drives low 1 Pin drives high Z Pin is high impedance input U Pin is input with pull-up A Pin is analog input (GPIO pin 0…3 only) T Pin is driven by clock or DAC (GPIO pin 0 and 4 only) The default mode after exiting reset is U on all GPIO pins.
AT:Pn=?	Get GPIO Pin Range	Print a list of possible modes for a pin. The table below lists the response. **Pin Modes** P0 0, 1, Z, U, A, T P1 0, 1, Z, U, A P2 0, 1, Z, U, A P3 0, 1, Z, U, A P4 0, 1, Z, U, T P5 0, 1, Z, U P6 0, 1, Z, U P7 0, 1, Z, U P8 0, 1, Z, U P9 0, 1, Z, U
AT:Pn=mode	Set GPIO Pin	Set the GPIO pin mode. For a list of the modes see the command AT:Pn?
AT:ADC Pn[-Pn[(1V|10V)]]?	Get GPIO Pin Analog Voltage	Measure the voltage applied to a GPIO pin. The command also allows measurement of the voltage difference across two GPIO pins. In differential mode, the full scale range may also be specified as 1V or 10 V. Note however that the pin input voltages must not exceed the range 0..VDD. The command returns the result as fraction of the full scale range (1 V if none is specified). The GPIO pins

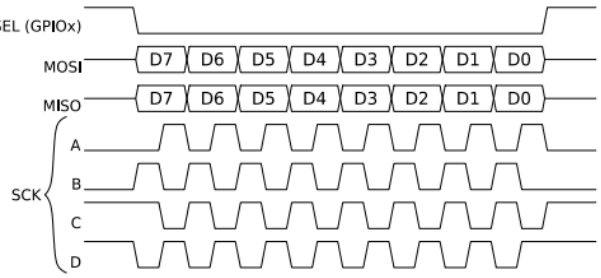
		referenced should be initialized to analog mode before issuing this command.
AT:SPI[(A|B|C|D)]=bytes	SPI Transaction	This command clocks out *bytes* on the SPI port. The clock frequency is 312.5 kHz. The command returns the bytes read on MISO during output. Optionally the clocking mode may be specified (default is A): **Mode** **Clock Inversion** **Clock Phase** A normal normal B normal alternate C inverted normal D inverted alternate  Note that SEL, if needed, is not generated by this command, and must instead be driven using standard GPIO commands (AT:Pn=0|1).
AT:CLK=freq,reffreq	Set Clock Generator	Output a square wave on the pin(s) set to T mode. The frequency of the square wave is (freq / 2^{16}) × reffreq. Possible values for reffreq are 20000000, 10000000, 5000000, 2500000, 1250000, 625000, 312500, 156250. Possible values if freq are 0…65535.
AT:CLK=OFF	Turn off Clock Generator	Switch off the clock generator
AT:CLK?	Get Clock Generator	Return the settings of the clock generator. Two numbers are returned, freq and reffreq.
AT:DAC=value	Set ∑Δ DAC	Output a ∑Δ DAC value on the pin(s) set to T mode. Parameter value may be in the range -32768…32767. The average output voltage is (1/2 + value / 2^{17}) × VDD. An external low pass filter is needed to get smooth output voltages. The modulation frequency is 20 MHz. A possible low pass filter choice is a simple RC low pass filter with R = 10 kΩ and C = 1µ F.
AT:DAC=OFF	Turn off ∑Δ DAC	Switch off the DAC
AT:DAC?	Get ∑Δ DAC	Return the DAC value

Tabla 4-1: Comandos AT soportados por el módulo AX-SIGFOX ANTSTAMP [10]

A continuación, se procede a desarrollar todas las funciones que proporcionen una solución al proyecto y que permitan la implementación del driver.

- uint8_t AXSF_Init(uint8_t UART_PORT)
- uint8_t AXSF_SendCommand(uint8_t UART_PORT, char *commando)
- uint8_t AXSF_SendAT(uint8_t UART_PORT)
- uint8_t AXSF_Reboot(uint8_t UART_PORT)
- uint8_t AXSF_SendFrame(uint8_t UART_PORT, uint8_t flag, char *mensaje)
- uint8_t AXSF_SendBit(uint8_t UART_PORT, uint8_t flag, uint8_t bit)
- uint8_t AXSF_Sleep(uint8_t UART_PORT, uint8_t Sleep)
- uint8_t AXSF_WakeUp(uint8_t UART_PORT, int8_t pin_number, char pin_port, uint8_t Sleep)
- uint8_t AXSF_SaveConfig(uint8_t UART_PORT)
- uint8_t AXSF_SetRegister(uint8_t UART_PORT, uint8_t Register, uint8_t Reg_value)
- uint8_t AXSF_Get(uint8_t UART_PORT, uint8_t command_id, char *Received_Info)
- void ascii2hex(char* ascii, char* hex)

Antes de diseñar la implementación de las soluciones que debe proporcionar el módulo de Sigfox, se debe incorporar también la propia inicialización del dispositivo.

4.3.1 uint8_t AXSF_Init (uint8_t UART_PORT)

La inicialización del módulo AX-SIGFOX ANTSTAMP se realiza mediante la inicialización de la UART que lo comunica con el microcontrolador, ya que cualquier comunicación entre el entorno/usuario y el módulo Sigfox se realizará mediante esa UART, es decir, los comandos AT se transmiten mediante UART.

Para implementar la inicialización del dispositivo Sigfox no es necesario el uso de ningún comando Hayes, sino que se declara una estructura del tipo *UART_Struct_Config*, definida en el driver "GIE_UART_DRIVER.c" y se configuran los parámetros de la UART, siendo los admitidos para el módulo de Sigfox: ***baud rate* 9600*, word length* 8 bits*, stop bits* 1 y *parity none***.

Esta función, además, hará uso de dos funciones que se desarrollarán más adelante, AXSF_Reboot, la cual reinicia el módulo con el fin de asegurar una inicialización sin valores previos, y AXSF_SendAT para comprobar que la inicialización se ha realizado correctamente, la cual hará que el módulo devuelva un "*OK*" en el caso de que se haya inicializado correctamente o un mensaje de error en el caso contrario.

A la función solo hay que pasarle por parámetros la UART que se utilizará.

A continuación, se explica el diseño de las funciones que permiten desarrollar el *driver*.

4.3.2 uint8_t AXSF_SendCommand (uint8_t UART_PORT, char *commando)

Esta función permite enviar cualquier comando al dispositivo de Sigfox y comprobar que se ha recibido correctamente, lo que la convierte en una función muy útil a lo largo del desarrollo software ya que este tipo de comprobaciones deben hacerse siempre.

Para utilizar esta función ha de conocerse el nombre de cada comando a enviar.

Para enviar el mensaje simplemente se ha de utilizar la función UART_WriteValue del *driver* GIE_UART_DRIVER.h. La dificultad se presenta a la hora de comprobar que el comando se haya enviado correctamente y que además la respuesta sea la adecuada y se procese de acorde al tipo de respuesta.

Como se puede ver en los comandos AT descritos en el *datasheet*, el dispositivo Sigfox puede responder de dos maneras dependiendo del comando que se haya empleado, un "OK" cuando el comando no requiere ninguna información de vuelta, o un mensaje con la información que el comando haya solicitado y nada más.

Es por ello que en este caso es esencial identificar y diferenciar aquellos comandos que, por simplicidad, son capaces de solicitar información al módulo de Sigfox.

Si se repasa el *datasheet* se aprecia que hay dos formatos de comandos que solicitan información, en primer lugar, aquellos acabados en "?" y, en segundo lugar, aquellos cuyo comando tiene la estructura "AT$I=" acompañados de un valor que se asocia con una información específica.

Para ello ha de comprobarse si el comando enviado cumple esas características, por lo tanto, es un comando que va a solicitar información al módulo Sigfox, o si, por el contrario, solo envía información al dispositivo.

Esta función puede ser llamada dentro de otras funciones ya que permite enviar un comando y comprobar que se ha recibido sin necesidad de implementar el mismo código varias veces.

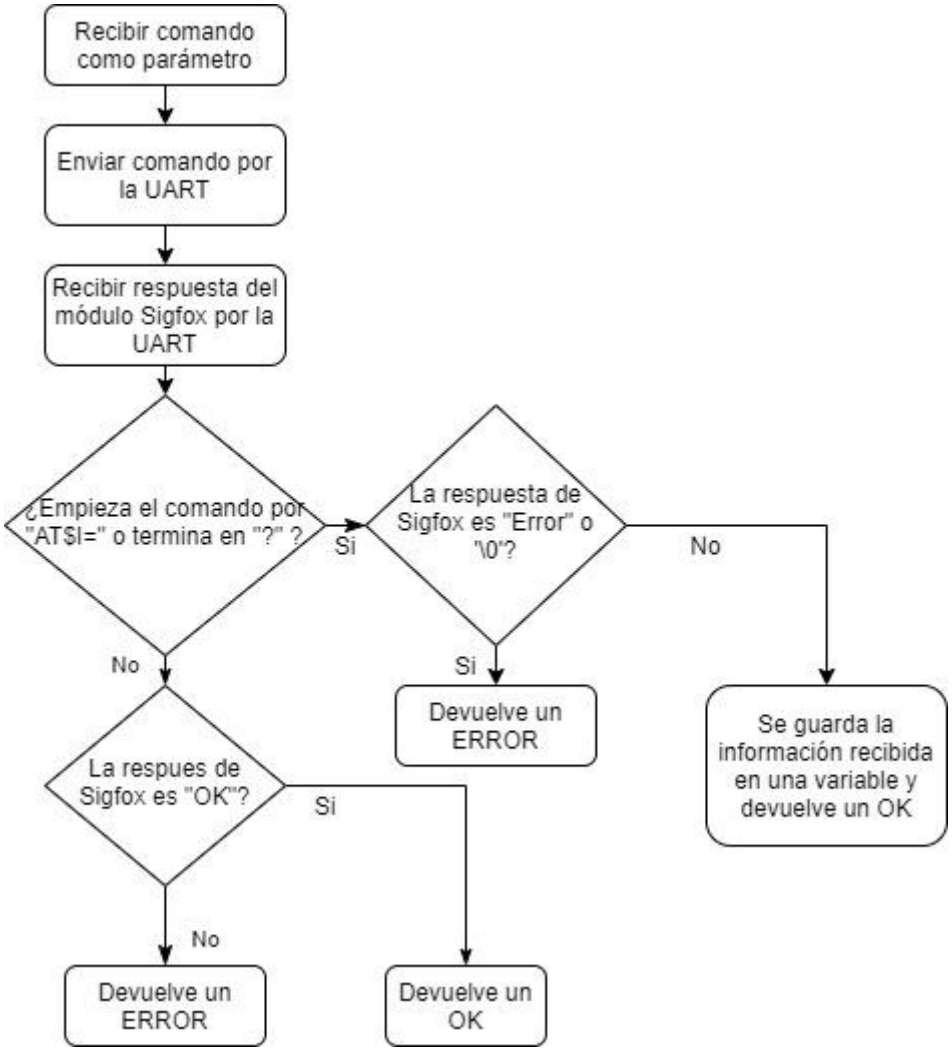


Figura 4-3: Diagrama diseño función AXSF_SendCommand

4.3.3 uint8_t AXSF_SendAT (uint8_t UART_PORT)

Permite enviar un comando AT (también llamado comando *dummy*, porque su única función es que el módulo Sigfox devuelva un mensaje "OK").

La implementación de esta función es sencilla, aunque no parece muy útil, es la manera más sencilla de comprobar que el dispositivo funciona correctamente después de despertarse o de inicializarse, por ejemplo, si se llama a la función y se recibe un "OK" de vuelta.

Para su implementación se utilizará la función AXSF_SendCommand.

Comando: AT

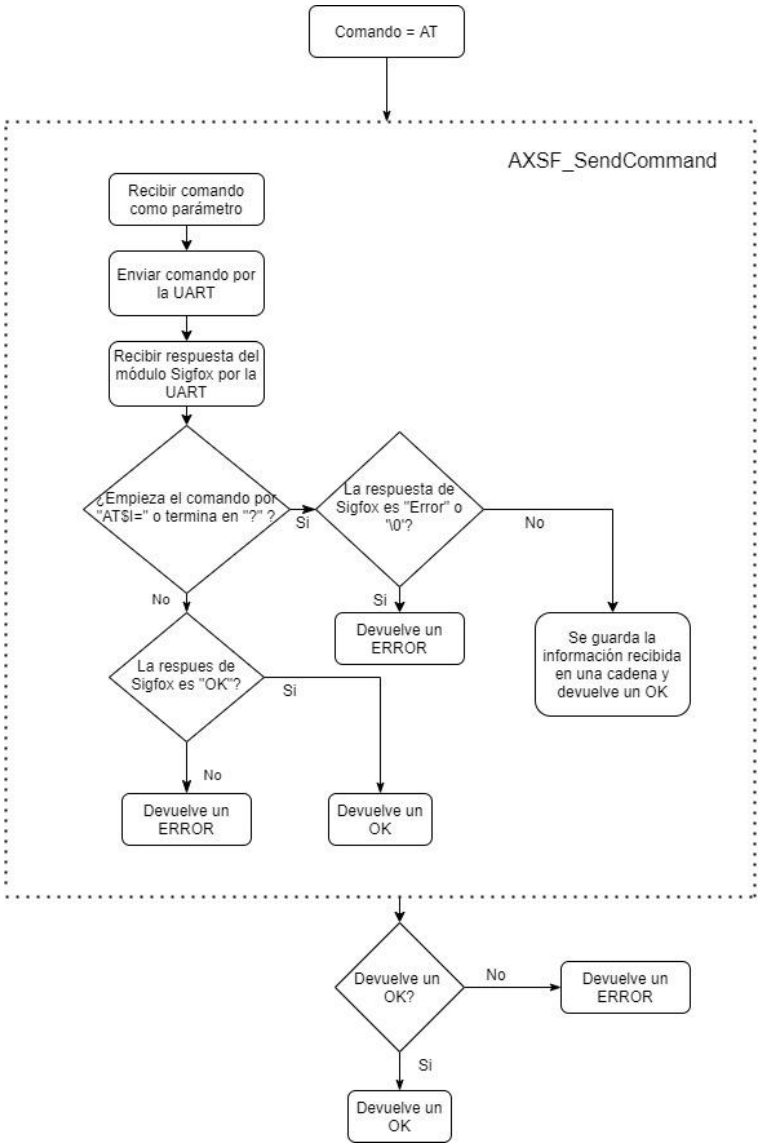


Figura 4-4: Diagrama diseño función AXSF_SendAT

4.3.4 uint8_t AXSF_Reboot (uint8_t UART_PORT)

Cuando se produce un *reset* en el módulo los valores de los ajustes se resetean con los valores almacenados en la memoria flash. Para ello utiliza la función AXSF_SendCommand.

Comando: AT$P=0

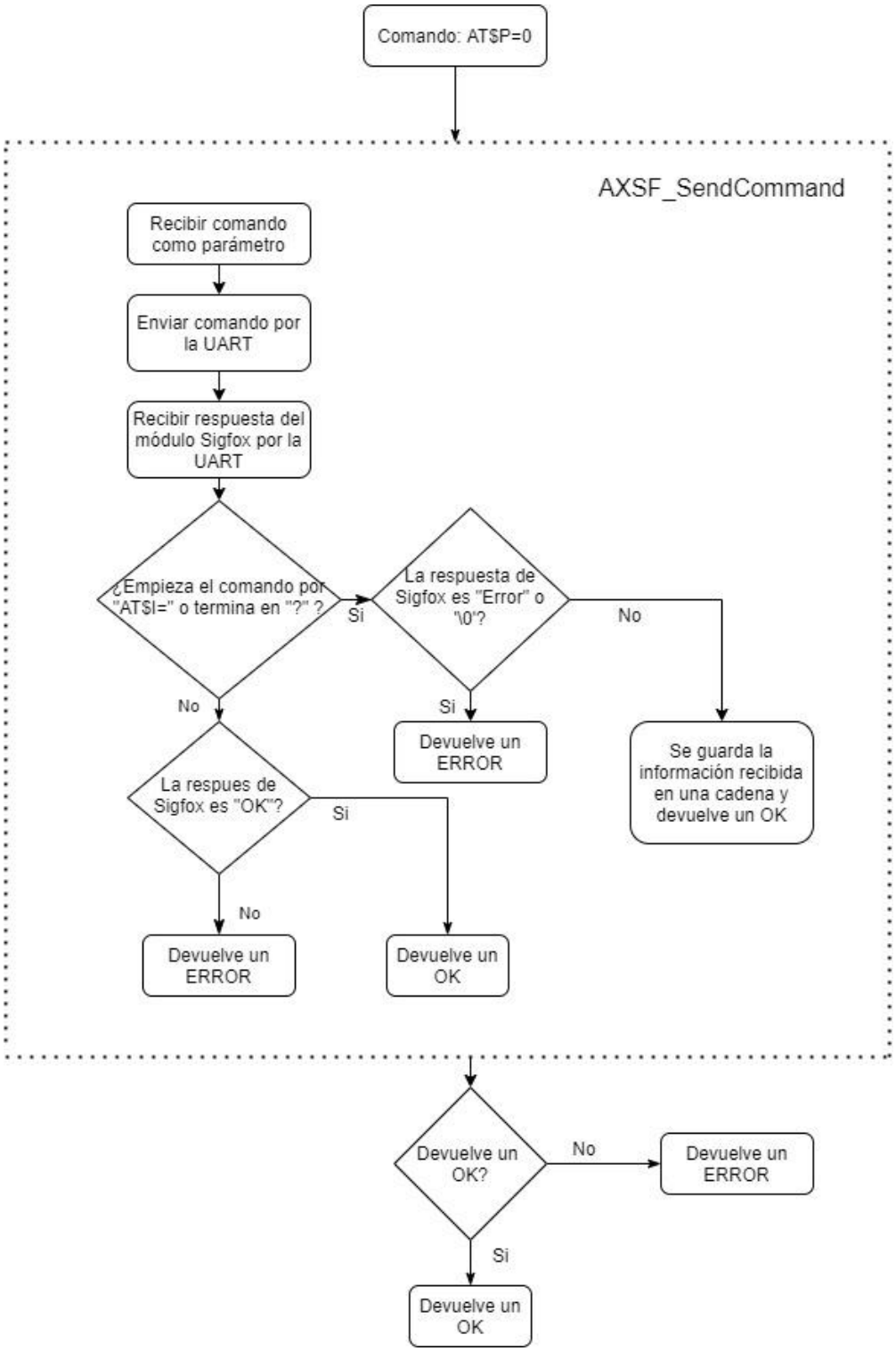


Figura 4-5: Diagrama diseño función AXSF_Reboot

4.3.5 uint8_t AXSF_SendFrame (uint8_t UART_PORT, uint8_t flag, char *mensaje)

Principal función de todo el proyecto. El módulo de Sigfox funciona principalmente como antena, lo que nos permite enviar mensajes de hasta 12 bytes, suficiente para el propósito de este trabajo. Además, mediante un bit de control (*flag*, predeterminado como 0) permite especificarle al módulo si debe esperar a recibir un

mensaje desde el *backend* (valor de *flag* 1). El último parámetro que se ha de pasar a la función es el mensaje/*frame* que se desea enviar.

Importante, se ha de tener en cuenta el formato de los mensajes que admite Sigfox son aquellos en formato hexadecimal, además deben ir siempre acompañados por los terminadores de cadena \r\n. En caso de que esto no se cumpla, Sigfox devolverá un error y no enviará ni recibirá datos.

Comando: AT$SF=frame[,bit] (bit = 0, 1 - bit *flag* para la recepción de mensajes).

A la función se le pasará como parámetros la UART, un bit *flag*, que determinará si espera un mensaje de vuelta (1) o no (0), y una cadena con el mensaje que se desee enviar.

En este caso la función no se implementa haciendo uso de la función AXSF_SendCommand, ya que no es eficiente al tener que diferenciar entre mensajes que esperan un mensaje desde el *backend* y los que no, sino que se hará uso de las funciones de escritura y lectura de la UART directamente.

El mensaje se envía a través de la UART mediante la función UART_WriteValue() a la cual se pasa como parámetros la UART y la variable char con el mensaje.

Para leer las respuestas de Sigfox se emplea la función UART_ReadValue() a la cual se le pasa como parámetros la UART y un char en el que se ha de almacenar la información recibida. Además, el tamaño máximo del mensaje y el *timeout*.

La respuesta dependerá en este caso del valor de la *flag* que se envió junto al mensaje, ya que, si esta es 0, no espera mensaje de vuelta por lo que si se ha enviado correctamente Sigfox responderá “OK”, sin embargo, si la *flag* es 1, se quedará esperando 25 segundos [13] hasta recibir el mensaje de vuelta. Cuando esto ocurra, se almacenarán los datos recibidos en una variable global a la que se puede acceder mediante puntero. Si alguna de estas dos opciones no ocurre, el microcontrolador devolverá un mensaje de error.

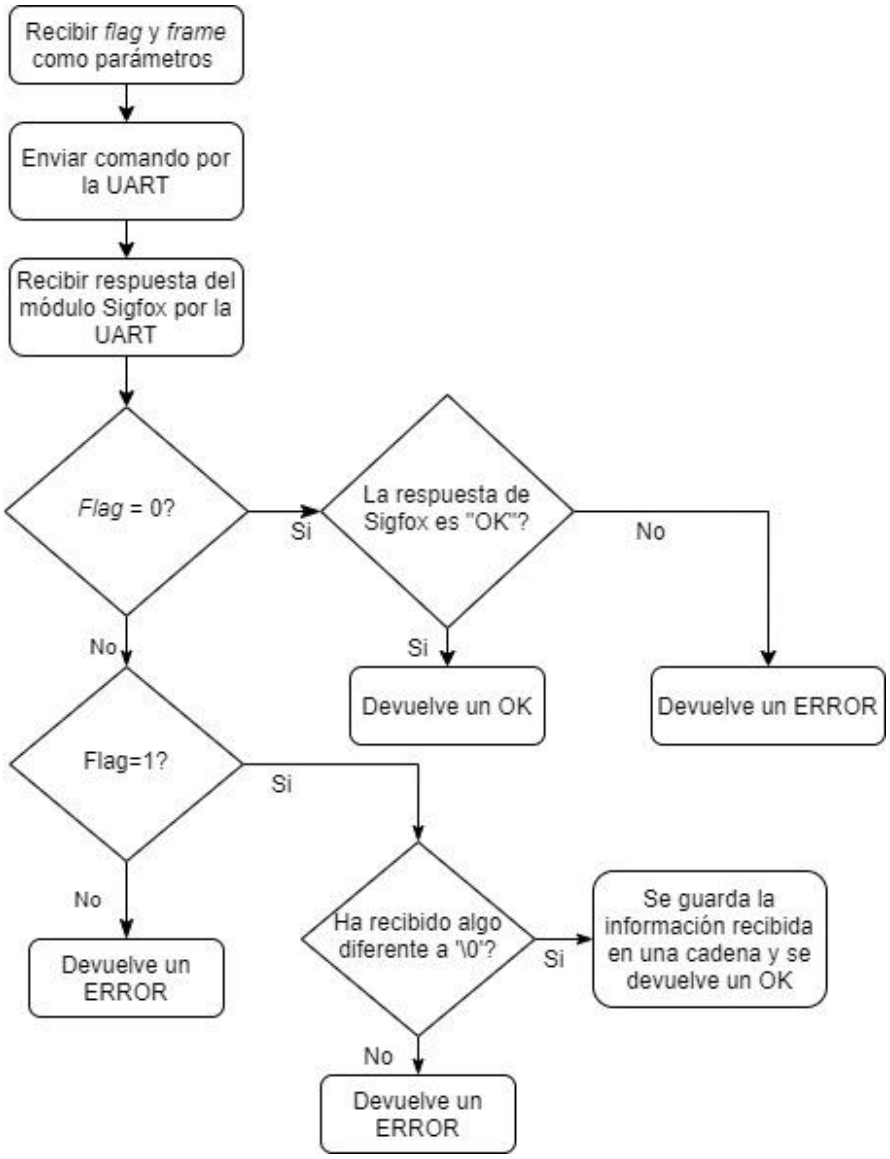


Figura 4-6: Diagrama diseño función AXSF_SendFrame

4.3.6 uint8_t AXSF_SendBit (uint8_t UART_PORT, uint8_t flag, uint8_t bit)

Su estructura es exacta a la de AXSF_SendFrame, la única diferencia es que esta solo soporta como mensaje un bit (0,1). También permite recibir mensajes desde la nube mediante el parámetro de *flag*.

Su implementación no proporciona una gran funcionalidad, pero tampoco implica una dificultad.

Comando: AT$SB=bit[,bit] (bit = 0, 1 – bit *flag* para la recepción de mensajes).

4.3.7 uint8_t AXSF_Sleep (uint8_t UART_PORT, uint8_t Sleep)

Los modos que permite el AX-SIGFOX ANTSTAMP son dos, *sleep* o *deep sleep*, la principal diferencia entre ambos es el consumo de corriente, siendo 1.6µA y 500nA respectivamente, permitiendo que la vida útil de la batería se prolongue durante años.

En el caso del modo *Sleep* sólo sigue funcionando el temporizador de activación de los mensajes *Out-of-Band* (fuera de banda). Cuando recibe un mensaje de fuera de banda (OOB), el módulo se despierta automáticamente para transmitir el mensaje y luego vuelve al modo de reposo.

A la función se le pasa como parámetros la UART y un entero (*Sleep*) que indica el modo, siendo 1 para *Sleep mode* o 2 para *Deep Sleep mode.*

Nota: Se recomienda encarecidamente poner los módulos AX-SIGFOX en modo de *Deep Sleep* cuando no se utilicen con el fin de mantener la vida útil de la batería el mayor tiempo posible.

Comando: AT$P=1

Cuando se utiliza el modo *Deep Sleep*, hay que tener en cuenta dos cosas: todo está apagado, los temporizadores no funcionan y se pierden todos los ajustes (para evitar esto se han de guardar los ajustes en la memoria flash previo a dormir el módulo).

Por lo tanto, los mensajes fuera de banda no se enviarán, los estados de los pines se congelan y el usuario debe asegurarse de que esto no dará lugar a condiciones que consuman mucha corriente.

Comando: AT$P=2

Para desarrollar esta función se vuelve a hacer uso de la función AXSF_SendCommand mediante la cual se envía el comando a través de la UART y se comprueba que Sigfox confirme la recepción.

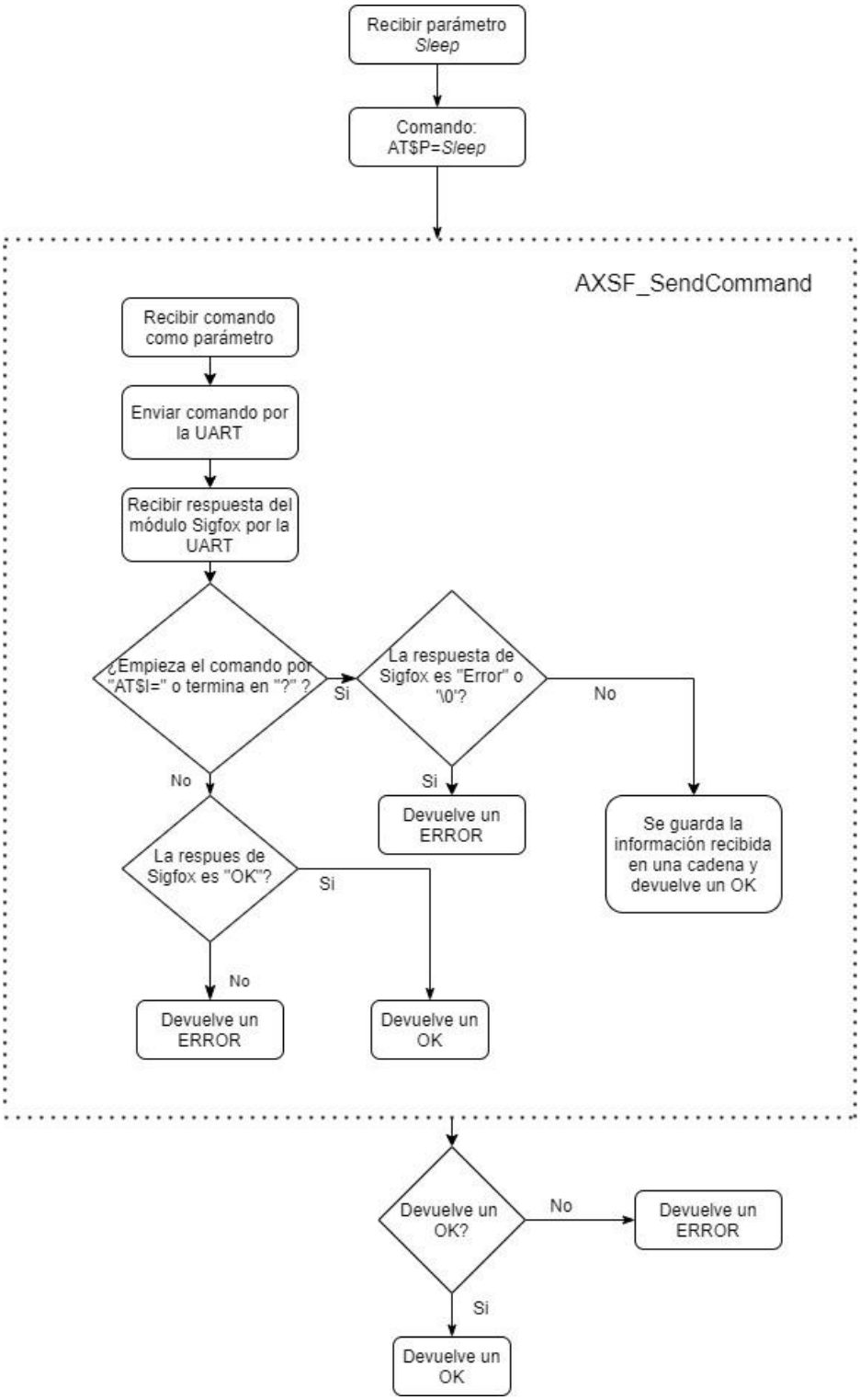


Figura 4-7: Diagrama diseño función AXSF_Sleep

4.3.8 uint8_t AXSF_WakeUp (uint8_t UART_PORT, uint8_t pin_number, char pin_port, uint8_t Sleep)

Esta función no implica el uso de comandos AT, sin embargo, ha de conocerse cómo puede despertarse el módulo de cada uno de los modos de sueño, ya que son diferentes maneras para cada uno (Figura 4-9).

Como el método de despertar de la placa es diferente dependiendo del modo en el que se encuentre dormida, el usuario debe especificar en qué modo se durmió la placa, mediante el parámetro *Sleep* de la función (1 para modo *Sleep* y 2 para modo *Deep Sleep)*.

En caso del modo *Sleep* (1), para despertar el módulo AX-SIGFOX hay que activar el pin UART_RX, por ejemplo, enviando un *break* (*break* es una violación del *framing* RS232, es decir, al menos 10 bits de duración a nivel bajo) a través de la UART.

Por otro lado, para el modo *Deep Sleep* basta con conectar el pin GPIO9 del módulo Sigfox a tierra. Como esto no se puede hacer manualmente ni se puede mantener siempre conectado a tierra se hará uso de uno de los pines GPIO del microcontrolador, en concreto el PC6.

Para ello, ha de hacerse uso del driver GIE_GPIO_DRIVER.h, el cual nos permite controlar los pines GPIO de nuestro microcontrolador.

El pin GPIO que se desee utilizar ha de inicializarse en primer lugar, haciendo uso de, las estructuras GPIO_PIN, para definir el número y el puerto del pin, y GPIO_Struct_Config para los modos, y de la función GPIO_Init (GPIO_PIN PIN, GPIO_Struct_Config config).

Para despertar al módulo del modo *Deep Sleep* se ha de conectar el pin GPIO9 del módulo Sigfox a tierra, para ello, el pin GPIO del microcontrolador, de valor predeterminado 1, debe tomar el valor 0 y volver al valor 1, es decir, un *toggle*, que permita más adelante volver a poner en modo *Deep Sleep* al módulo Sigfox.

Cuando se realiza el procedimiento de despertar al módulo es la única ocasión en la que Sigfox no devolverá ningún mensaje, ni de confirmación ni de error, por lo que se puede comprobar si se ha despertado correctamente si se envía un comando AT y la placa devuelve un “OK”.

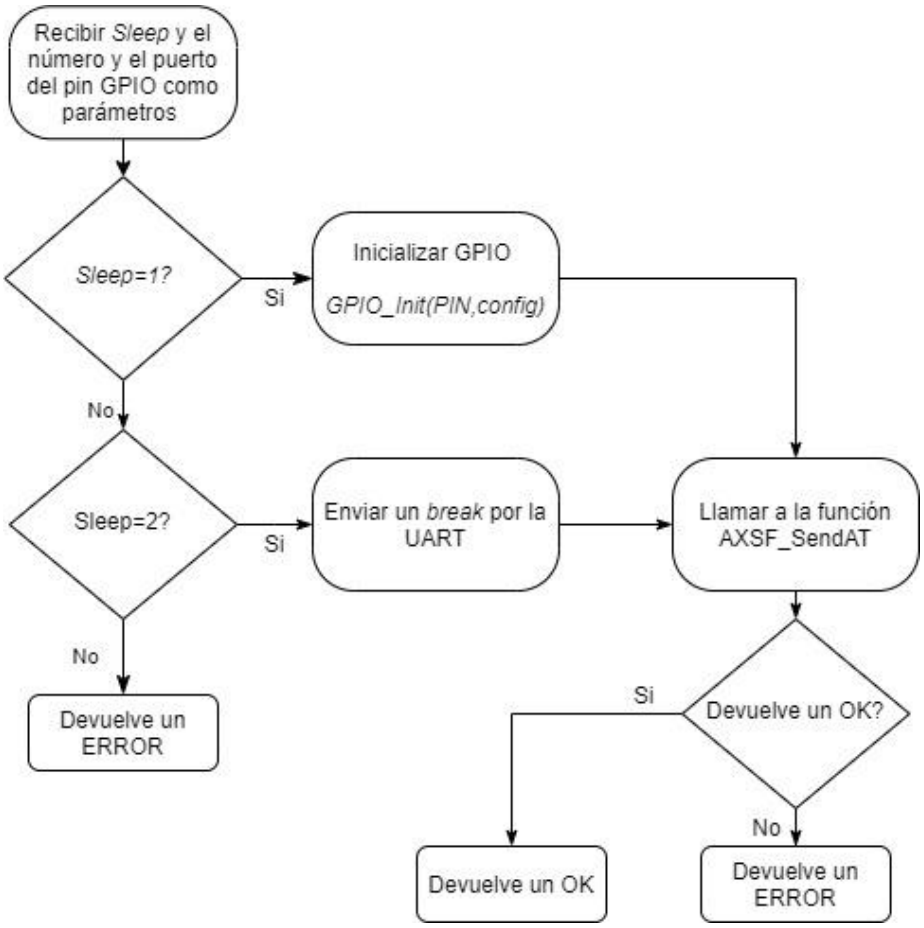


Figura 4-8: Diagrama diseño función AXSF_WakeUp

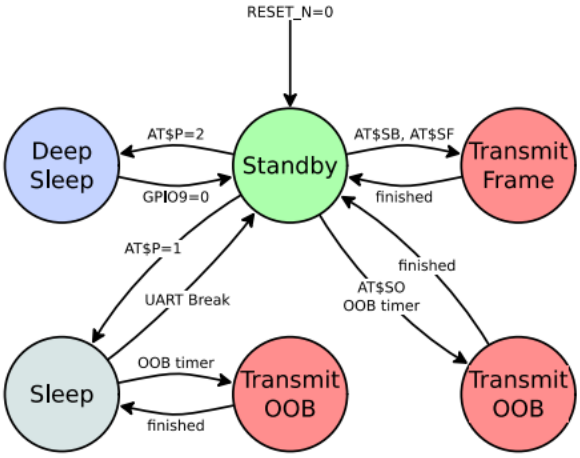


Figura 4-9: Diagrama de estado de los modos de energía de AX-SIGFOX ANTSTAMP [10]

4.3.9 uint8_t AXSF_SaveConfig (uint8_t UART_PORT)

Escribe todos los ajustes en la memoria flash (frecuencias RX/TX, registros) para que no se pierdan cuando se produce un *reset* o si entra en modo *Deep Sleep* o si se produce una pérdida de energía. Comprueba que el dispositivo Sigfox devuelve un mensaje de "OK".

Para esto se vuelve a hacer uso de la función AXSF_SendCommand, que permite enviar el comando y comprobar que se ha recibido correctamente por el módulo Sigfox.

Comando: AT$WR

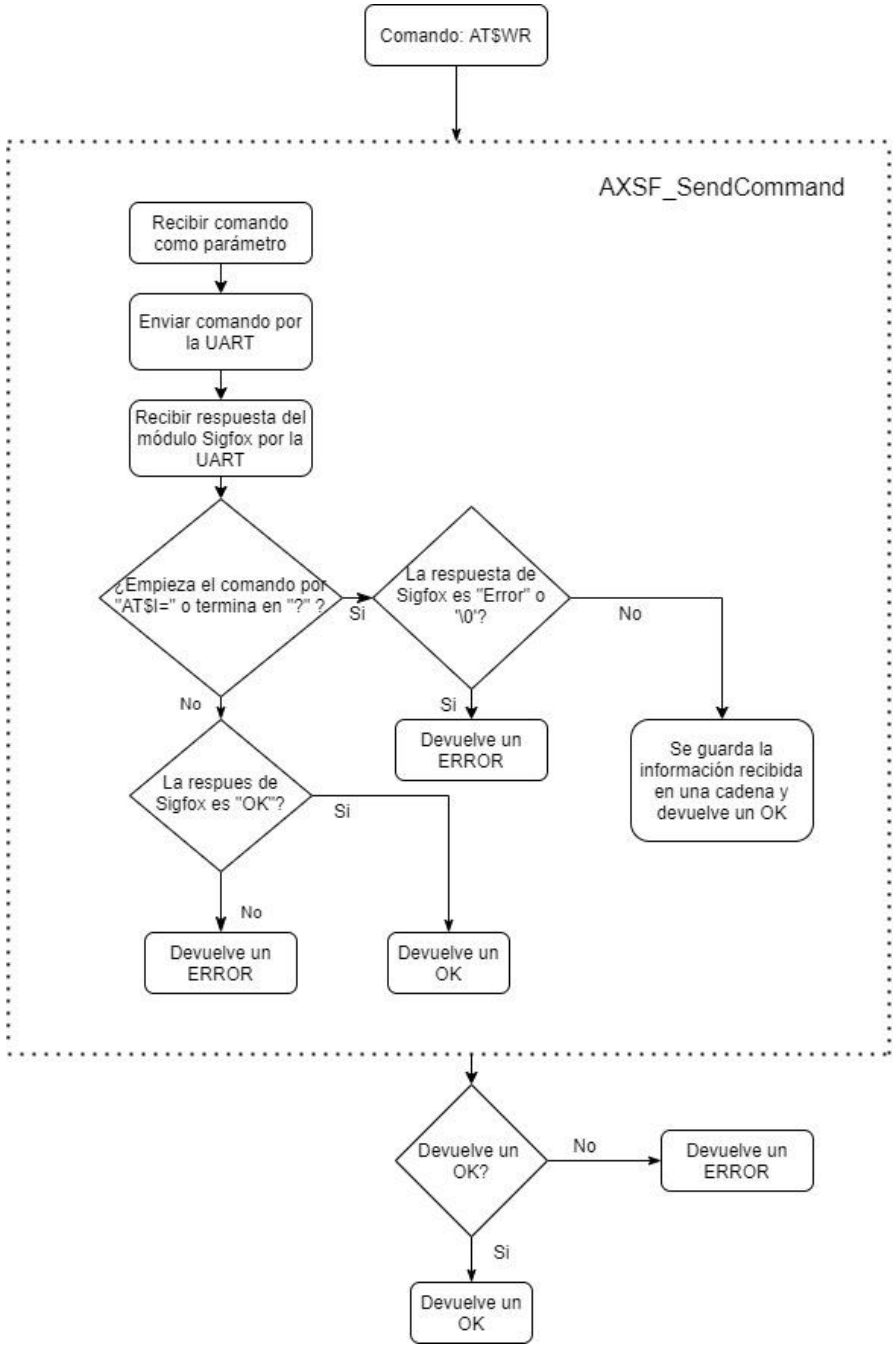


Figura 4-10: Diagrama diseño función AXSF_SaveConfig

4.3.10 uint8_t AXSF_SetRegister (uint8_t UART_PORT, uint8_t Register, uint8_t Reg_Value)

Los registros que soporta el módulo AX-SIGFOX ANTSTAMP son los que se presentan en la Tabla 4-2, conociendo estos datos, se puede implementar una función en la que se puedan configurar estos registros, a la cual se le pasará como parámetros la UART, el registro que se desee configurar y el nuevo valor de este.

Number	Name	Description	Default	Range	Units
300	Out Of Band Period	AX-SIGFOX module sends periodic static messages to indicate that they are alive. Set to 0 to disable.	24	0-24	hours
302	Power Level	The output power of the radio.	14	0-14	dBm

Tabla 4-2: Registros de AX-SIGFOX ANTSTAMP [10]

Primero se comprueba que el nuevo valor que se desea configurar está dentro del rango permitido y a continuación se envía la nueva configuración al dispositivo, el cual devolverá un “OK” si lo recibe correctamente.

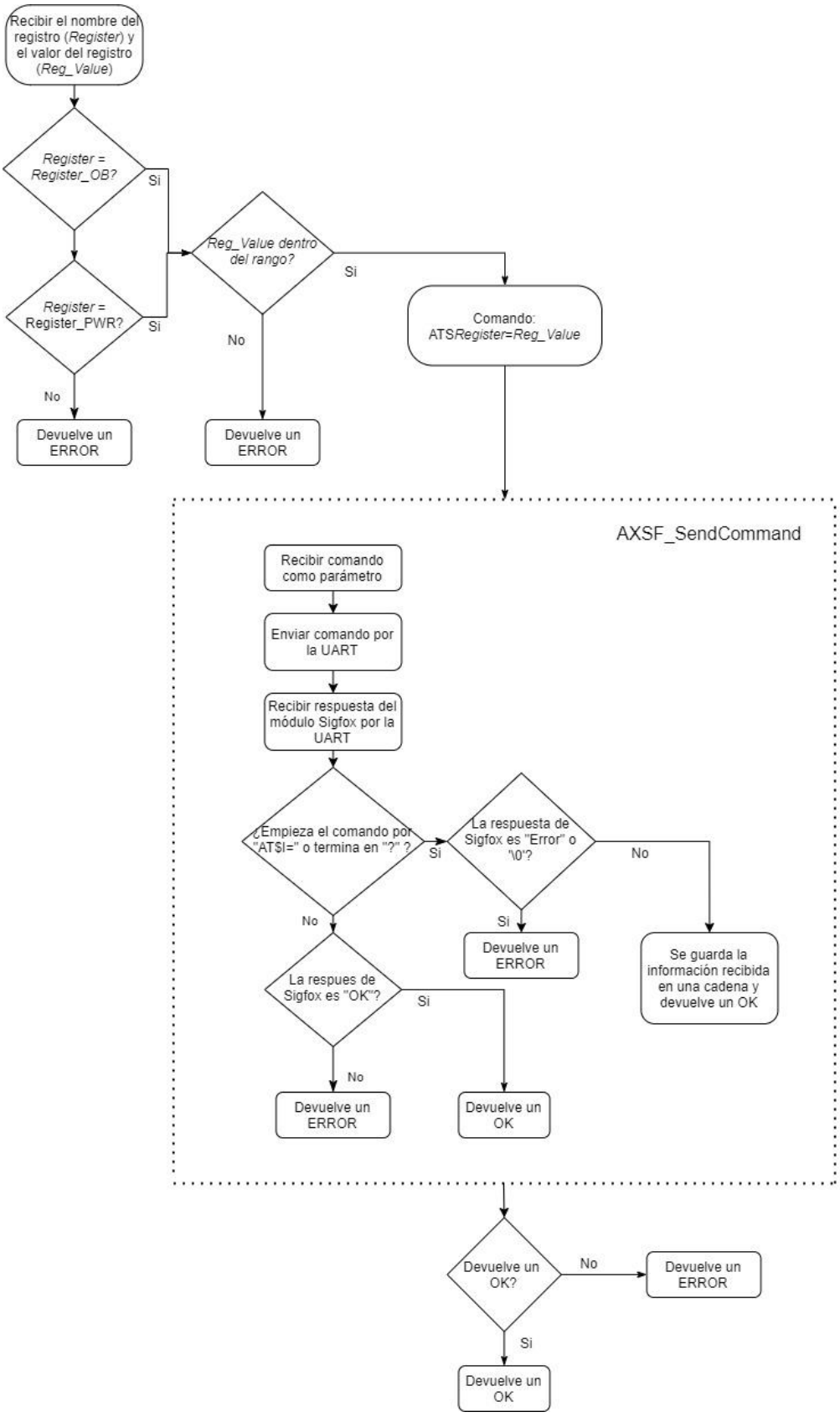


Figura 4-11: Diagrama diseño función AXSF_SetRegister

Además de estas funciones, se ha decidido implementar otras funciones más genéricas que permiten solicitar información o hacer comprobaciones de manera muy sencilla.

4.3.11 uint8_t AXSF_Get (uint8_t UART_PORT, uint8_t command_id, char *Received_Info)

Permite solicitar información haciendo uso de un puntero a array previamente declarado, donde los array son cadenas de caracteres con cada uno de los comandos que solicitan información al módulo Sigfox, para ello no hace falta conocer el comando previamente, simplemente pasar como parámetro a la función la información que se desea solicitar de todas las disponibles.

Se pasan como parámetros a la función el nombre del comando definido y una cadena donde se copiará la información de vuelta.

La función se encarga de relacionar el nombre del comando con su respectivo comando declarado en el puntero y enviar y recibir la información.

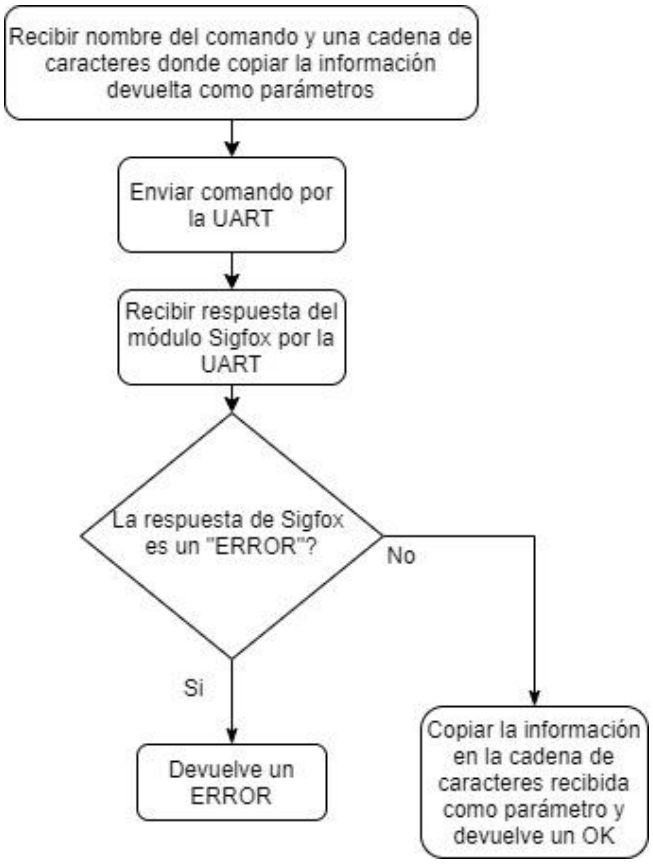


Figura 4-12: Diagrama diseño función AXSF_Get

4.3.12 void ascii2hex (char* ascii, char* hex)

Por último, se ha implementado una función que permite convertir cualquier cadena de caracteres ascii en hexadecimal. Como se ha explicado previamente, Sigfox solo admite *frames* en formato hexadecimal, el usuario puede elegir si usar esta función o no.

A la función debe pasarse como parámetros dos cadenas de caracteres, una en ascii y otra en la que se copiará el resultado en hexadecimal.

4.3.13 Código de cabecera de la aplicación

A continuación, se puede leer el código del archivo de cabecera (.h) para la implementación del *driver*.

En este se procede a definir los códigos de error de cada una de las funciones implementadas en el código fuente, junto a la definición de las propias funciones.

```
#ifndef GIE_SIGFOX_DRIVER_H
#define GIE_SIGFOX_DRIVER_H
#pragma once

#include "GIE_DEBUG_DRIVER.h"
#include "GIE_UART_DRIVER.h"
#include "GIE_IWDG_DRIVER.h"
#include "GIE_GPIO_DRIVER.h"
#include "GIE_RTC_DRIVER.h"
#include "stm32l1xx_hal.h"
#include "stm32l1xx_nucleo.h"
#include <string.h >
#include <stdarg.h>
#include <stdio.h>

// ***** INIT - GENERAL CONSTANTS TO DEFINE THE CONFIG ***** //
// ********************************************************************** //


/* Define if inside FreeRTOS */
#define SIGFOX_USE_FREERTOS
#define Command_id 50


/***************Private defines************/
/**
* @defgroup AXSF_INIT_CODES Códigos de inicialización
*
* @{
*/
#define AXSF_INIT_OK 0
#define AXSF_INIT_UART_ERROR 1
#define AXSF_INIT_REBOOT_ERROR 2
#define AXSF_INIT_CONFIG_ERROR 3
/** @} */

/**
* @defgroup AXSF_FLAG_CODES Códigos de obtención de parámetros adicionales
*
* @{
*/
#define AXSF_FLAG_CODE_WRONG 10
/** @} */

/**
* @defgroup AXSF_GET_CODES Códigos de obtención de parámetros adicionales
*
* @{
*/
#define AXSF_GET_OK 20
#define AXSF_GET_ERROR 21
/** @} */
```

```
/**
*  @defgroup  AXSF_RECEIVED_CODES  Códigos  de  obtención  de  parámetros
adicionales
*
* @{
*/
#define AXSF_RECEIVED_INFO_OK       30
#define AXSF_RECEIVED_INFO_ERROR    31
#define AXSF_RECEIVE_OK 32
#define AXSF_RECEIVE_ERROR 33
#define AXSF_RECEIVE_ACTUALIZATION_OK 34
#define AXSF_RECEIVE_ACTUALIZATION_ERROR 35


/** @} */

/**
* @defgroup AXSF_SEND_CODES Códigos de envío de comando
*
* @{
*/
#define AXSF_SEND_CMD_OK 40
#define AXSF_NO_CMD_SEND 41
#define AXSF_SEND_CMD_ERROR 42
#define AXSF_SEND_AT_OK 43
#define AXSF_SEND_AT_ERROR 44
#define AXSF_SEND_FRAME_OK 45
#define AXSF_SEND_FRAME_ERROR 46
#define AXSF_SEND_BIT_OK        47
#define AXSF_SEND_BIT_ERROR     48

/** @} */

/**
* @defgroup AXSF_REGISTER_CODES Códigos de modos de consumo del módulo
*
* @{
*/
#define AXSF_SET_REGISTER_OK 50
#define AXSF_SET_REGISTER_ERROR 51
#define AXSF_SET_WRONG_REGISTER 52
#define AXSF_SET_WRONG_REGISTER_VALUE 53
/** @} */

/**
* @defgroup AXSF_SAVE_CONFIG_CODES Códigos de modos de consumo del módulo
*
* @{
*/
#define AXSF_SAVE_CONFIG_OK 60
#define AXSF_SAVE_CONFIG_ERROR 61
/** @} */

/**
* @defgroup AXSF_SLEEP_CODES Códigos de modos de consumo del módulo
*
* @{
*/
#define AXSF_SLEEP_OK 80
#define AXSF_SLEEP_ERROR 81
#define AXSF_SLEEP_CODE_WRONG 82

/** @} */

/**
* @defgroup AXSF_REBOOT_CODES Códigos de reinicio
*
* @{
*/
#define AXSF_REBOOT_OK 90
#define AXSF_REBOOT_ERROR 91
/** @} */
```

```
/**
* @defgroup AXSF_WAKEUP_CODES Códigos de cierre de transmisión
*
* @{
*/
#define AXSF_WAKEUP_OK 100
#define AXSF_WAKEUP_ERROR 101
/** @} */

/* Comandos*/
//Info
#define Software 0
#define Contact 1
#define Rev_LB 2
#define Rev_UP 3
#define M_Firm 4
#define m_Firm 5
#define Rev_Firm 6
#define Var_Firm 7
#define Ver_Firm 8
#define SIGFOX 9
#define ID 10
#define PAC 11

//Get
#define Voltage 12
#define Temperature 13
#define Register_OB 14
#define Register_PWR 15

////Registers
//#define OB 300
//#define PWR 302


/*****FUNCTIONS*****/
uint8_t AXSF_Init(uint8_t UART_PORT);
uint8_t AXSF_Reboot(uint8_t UART_PORT);
uint8_t AXSF_SendAT(uint8_t UART_PORT);
uint8_t AXSF_SendBit(uint8_t UART_PORT, uint8_t flag, uint8_t bit);
uint8_t AXSF_SendCommand(uint8_t UART_PORT, char *command);
uint8_t  AXSF_SendFrame(uint8_t  UART_PORT,  uint8_t  flag,  char  *mensaje);
//Solo admite hexadecimal. Minusculas.
uint8_t AXSF_SaveConfig(uint8_t UART_PORT);
uint8_t AXSF_Get(uint8_t UART_PORT, uint8_t command_id, char *Received_Info);
uint8_t  AXSF_SetRegister(uint8_t  UART_PORT,  uint8_t  Register,  uint8_t
Reg_value);
uint8_t AXSF_Sleep(uint8_t UART_PORT, uint8_t Sleep);
uint8_t  AXSF_WakeUp(uint8_t  UART_PORT,  int8_t  pin_number,  char  pin_port,
uint8_t Sleep);
void ascii2hex(char* ascii, char* hex);
#endif
```

Con el desarrollo software terminado, se necesita una manera de comprobar que es posible utilizarse en un entorno real, para ello es esencial la implementación en un sistema operativo en tiempo real capaz de manejar las tareas del *driver* con independencia.

5. INTEGRACIÓN EN FREERTOS

FreeRTOS es un sistema operativo en tiempo real líder en el mercado para microcontroladores y pequeños microprocesadores. Desarrollado en colaboración con las principales empresas fabricantes de chips del mundo y distribuido libremente bajo la licencia de *open-source* del MIT, FreeRTOS incluye un núcleo y un conjunto creciente de bibliotecas de IoT adecuadas para su uso en todos los sectores industriales. [14]

La implementación de una aplicación en un sistema operativo permite verificar la funcionalidad de este, bajo las limitaciones que supone un sistema operativo real, manejar los recursos del sistema de manera más eficiente, lo que permite disminuir el consumo de energía. Además, a nivel de arquitectura del software ayuda a organizar las funcionalidades de manera más efectiva gracias a la división en tareas.

Figura 5-1: Principales empresas colaboradoras con FreeRTOS [14]

5.1 Arquitectura de tareas y colas

FreeRTOS permite a las aplicaciones organizarse como un conjunto de tareas de ejecución independiente. Para ello, se hace uso de distintos mecanismos como las colas, las colas por interrupción, los semáforos, las definiciones de prioridad, etc.

En concreto, en este proyecto se han implementado tareas y una cola que permite la comunicación entre ellas.

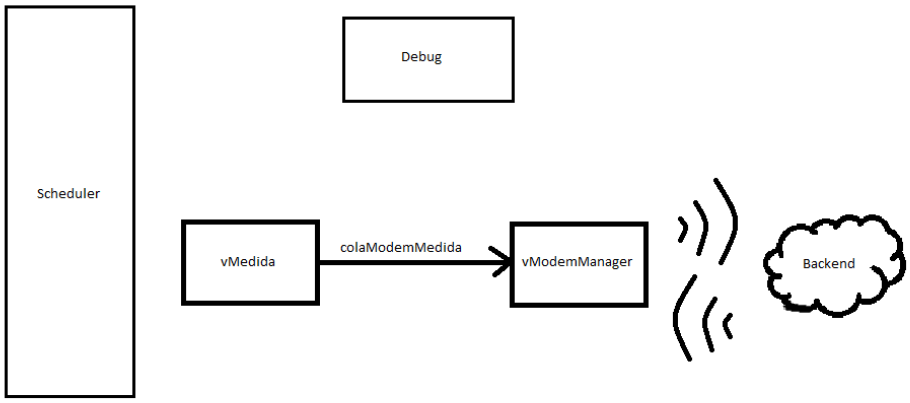


Figura 5-2: Representación funcionamiento de tareas en FreeRTOS

Donde las tareas *Scheduler* y *Debug* son tareas que se utilizan en el desarrollo del funcionamiento del RTOS, pero su desarrollo no se contempla en este trabajo.

De esta forma, la comunicación entre las dos tareas principales se realiza con el uso de la cola, mediante las funciones predefinidas *xQueuSend*, para enviar datos, y *xQueuReceive*, para recibir datos.

5.2 void vMedida ()

La tarea vMedida() se encargará de medir la temperatura de un sensor de la placa, esta solicita al módulo Sigfox el valor de la temperatura que mide el sensor que este tiene e introducir el dato en una cola y a continuación, procede a dormir al módulo (Figura 5-3).

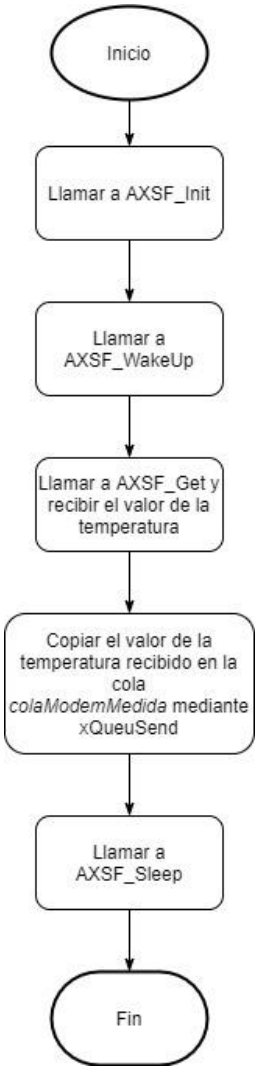


Figura 5-3: Diagrama diseño tarea vMedida

5.3 void vModemManager ()

La tarea *vModemManager* permite que el módulo se comunique con el *backend*, y pueda transferir un dato.

5.3.1 Enviar datos recibidos desde la cola

Para llevar a cabo esta implementación, se hace uso de la tarea *vModemManager*, cuya funcionalidad será la de recibir el dato de la cola enviado desde *vMedida* y ser capaz de enviarlo al *backend*.

Para ello se accede a la cola haciendo uso de la función *xQueuReceive* y mediante las funciones definidas en el driver se envía.

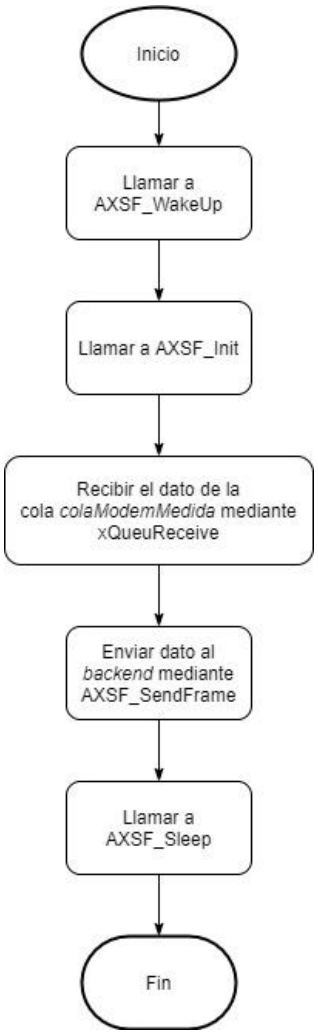


Figura 5-4: Diagrama diseño tarea vModemManager para enviar datos de una cola

5.3.2 Enviar un dato random

Por último, se explica la implementación de la tarea *vModemManager*, cuya funcionalidad es realizar las acciones de inicializar, despertar el módulo, enviar al *backend* un valor *random* y volver a dormir el módulo mediante las funciones del *driver* de Sigfox anteriormente descrito.

Esta tarea no hace uso de la cola. De esta manera se comprueba que las funciones principales declaradas dentro del *driver* son compatibles con el RTOS, y se puede implementar un sistema más complejo, antes de realizar pruebas más complejas como la explicada anteriormente.

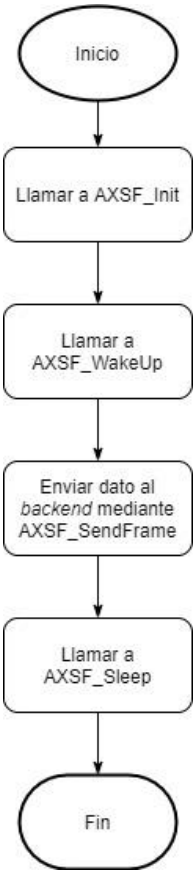


Figura 5-5: Diagrama diseño tarea vModemManager para enviar un dato random

6. PRUEBAS DE VALIDACIÓN DEL SISTEMA

Para llevar a cabo la validación del sistema, se ha de implementar una rutina que permita al módulo enviar un dato, por ejemplo, la temperatura medida desde el sensor de temperatura del dispositivo, al *backend*.

Para ello, en primer lugar, se realiza una prueba desde el entorno de desarrollo IAR Embedded Workbench y se monitoriza la comunicación entre dispositivos con el Docklight, y, en segundo lugar, se implementará la rutina en FreeRTOS y se comprobará que funciona y que es independiente al usuario.

6.1 Prueba desde el entorno de desarrollo IAR

Para comprobar que la aplicación se ha desarrollado correctamente y cumple las funciones especificadas en el diseño, se implementa una secuencia similar a la que se realiza en el sistema operativo, de manera que se llama a las siguientes funciones y mediante Docklight se puede observar la comunicación entre el microcontrolador y el módulo Sigfox, siendo el puerto COM8 el microcontrolador y el COM10 Sigfox.

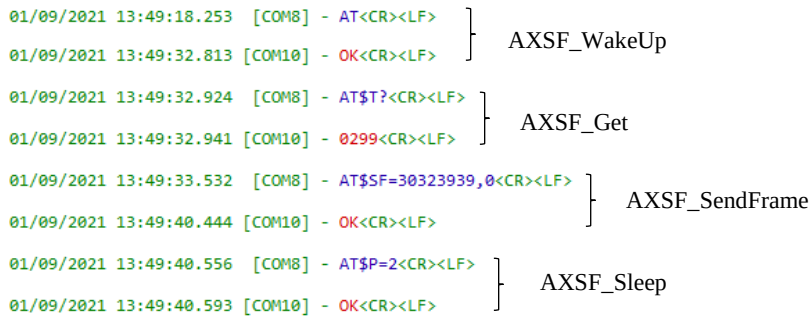


Figura 6-1: Comunicación prueba 1 en Docklight

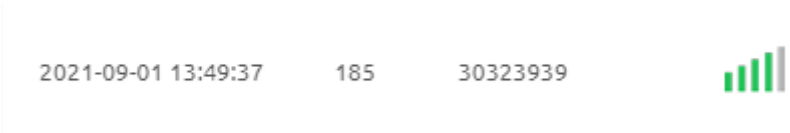


Figura 6-2: Mensaje recibido en el Backend para la prueba 1

Como se puede observar en la Figura 6-1, cada una de las funciones envían los comandos correspondientes y se recibe la respuesta desde Sigfox, en primer lugar, se despierta el módulo mediante AXSF_WakeUp, y se envía un AT para comprobar que se ha despertado correctamente, a lo que el módulo Sigfox responde con un OK.

A continuación, se solicita la información del sensor de temperatura mediante AXSF_Get y el módulo Sigfox la devuelve como décimas de grados centígrados. Se envía esta información al *Backend* mediante AXSF_SendFrame, a lo que el módulo Sigfox responde un OK.

Por último, se duerme el módulo mediante AXSF_Sleep, donde Sigfox vuelve a responder un OK.

6.2 Prueba desde el entorno de FreeRTOS

En el caso de la implementación en FreeRTOS, el proceso es bastante similar al anterior, pero en este caso, el propio sistema operativo implementa una rutina entre las tareas vMedida y vModemManager, las cuales se comunican mediante una cola. La comunicación entre el microcontrolador y el módulo de Sigfox se intercepta y observa en el Docklight, como muestra la Figura 6-3.

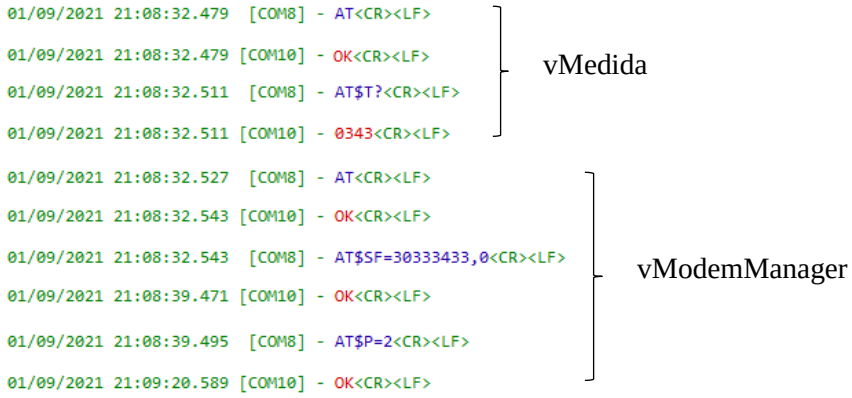


Figura 6-3: Comunicación prueba 2 en Docklight

Figura 6-4: Mensaje recibido en el Backend para la prueba 2

En la Figura 6-4, se observa como el mensaje se recibe correctamente en el *Backend.*

Con este apartado, se comprueba que el desarrollo que se ha llevado a cabo durante el proyecto ha conseguido cubrir los requisitos especificados para el funcionamiento de la aplicación, y se espera que la integración en futuros proyectos sea posible. De esta manera, se presentan en el siguiente apartado las conclusiones obtenidas a lo largo del desarrollo.

7. CONCLUSIONES Y LÍNEAS FUTURAS

A nivel personal, la posibilidad que me ofrecieron mis tutores para desarrollar este proyecto me ha permitido conocer en mayor profundidad el ámbito de los sistemas inalámbricos y de las redes de comunicación, además de obtener una nueva perspectiva sobre cómo influye la electrónica en el continuo desarrollo de las comunicaciones inalámbricas. Además, me ha permitido adquirir conocimientos de programación en C de *Basic Software* mucho más amplios, ya que solo conocía las nociones básicas impartidas en la asignatura de Informática de primero de carrera, y que pueden resultar muy útiles en el mercado laboral actual enfocado a la electrónica.

A nivel conceptual, durante el desarrollo de este Trabajo de Fin de Grado se ha comprobado que la llegada de las redes inalámbricas LPWAN permite implementar soluciones 100% monitorizadas remotamente, lo que permite prescindir del factor humano y el coste que ello supone. Además, dentro de las características de las tecnologías inalámbricas, se ha conseguido dar solución a aquellas donde las actuales tecnologías se quedan cortas, como son el rango y el consumo energético.

Tras la justificación del trabajo a realizar se ha realizado un estudio del estado actual de las redes inalámbricas, enfocándose en las redes LPWAN, como LoRaWAN, NB-IoT y Sigfox, analizando sus características, comparándolas y optando por Sigfox para la solución buscada.

Posteriormente, se han desarrollado los módulos seleccionados para llevar a cabo la implementación del trabajo. En primer lugar, el microcontrolador STM32L152RE del fabricante ST, caracterizado por su ultra bajo consumo, una CPU de 32 bits y 32 MHz, memoria Flash de hasta 512 Kbytes y RAM de hasta 80 Kbytes, alimentación de 3.3V y 5V, CADs y CDAs, 11 periféricos de comunicación y 11 temporizadores. La función principal del microcontrolador será soportar los programas desarrollados y comunicarse con el módulo Sigfox.

En segundo lugar, el transceptor Sigfox, modelo AX-SF10 ANT21-868, caracterizado por su ultra bajo consumo en modo *Deep Sleep*, su comunicación bidireccional con el *backend* y la implementación de sensores de tensión y temperatura en el propio módulo. Este dispositivo es controlado mediante comando Hayes (AT), y será el encargado de permitir la comunicación entre el usuario y la nube, de manera que su principal función será enviar hacia el *backend*, desde donde el usuario puede accederlos o redirigirlos a un servidor externo.

Por último, se hace uso de un módulo TTL, en concreto el modelo FT4232H-56Q Mini Module. Este módulo, caracterizado por poseer 4 canales de comunicación que serán reconocidos por el ordenador como 4 puertos COM virtuales (VCP) independientes, permite interceptar los mensajes que se transmiten entre el microcontrolador y el módulo Sigfox, y es posible mostrarlos por pantalla haciendo uso del programa Docklight.

Para la implementación del trabajo, se ha realizado el montaje de dos *setups* diferentes. En primer lugar, se realizó la conexión únicamente entre el módulo Sigfox y el TTL, de manera que se pudiese implementar una comunicación directa entre usuario y transceptor, como se muestra en la Figura 3-12 en la página 36. Mediante esta configuración y el uso del Docklight, fue posible comunicarse con el transceptor de Sigfox sin necesidad de utilizar el microcontrolador, con el fin de comprender mejor las funcionalidades que ofrecía y cómo utilizarlas haciendo uso de comandos AT.

Una vez se ha completado este primer contacto con el transceptor, se incluye el microcontrolador en la implementación anterior, de manera que ahora la placa TLL ya no funciona como interfaz de comunicación entre el usuario y el dispositivo Sigfox, sino que se limita a interceptar la comunicación entre transceptor y microcontrolador. Para ello, se hace uso del *setup* mostrado en la Figura 3-14 en la página 38. Con este *setup*, la comunicación se produce entre el microcontrolador, ejecutando el programa, y el transceptor.

Posteriormente, se ha definido y desarrollado el programa que debe ejecutarse en el microcontrolador, en este caso, un driver de control en el cual se implementan las funcionalidades que permiten dar solución a los objetivos planteados, como la transmisión y recepción de datos hacia y desde el *backend*, el uso de los distintos modos de bajo consumo, la configuración del módulo y de los registros, o el reseteo de este.

Finalmente, se ha hecho uso del entorno de FreeRTOS para proporcionar una validación al desarrollo del driver y se han expuesto los resultados obtenidos tanto en el entorno de desarrollo como una vez implementado en FreeRTOS, de manera que se ha demostrado la viabilidad de la integración del driver en una aplicación IoT con las garantías de éxito suficientes.

Cabe mencionar que el desarrollo software se ha realizado con herramientas industriales, como son el entorno de desarrollo IAR Embedded Workbench o Docklight, y haciendo uso de librarías ya desarrolladas para el control de periféricos como la UART y los pines GPIO.

Como posibles futuras líneas de investigación relacionadas con este trabajo, los futuros proyectos se basarán en un desarrollo más amplio de las funcionalidades ofrecidas por Sigfox, como, por ejemplo, la implementación del servidor externo en el *backend* de Sigfox para la recepción de datos, y posterior visualización de estos, con el fin de hacer un correcto tratamiento de datos, posiblemente enfocado dentro del ámbito del *Data Engineering*.

Otro ejemplo es el desarrollo de un software que permita transmitir tramas de más de 12 bytes haciendo uso de tramas seccionadas. También resulta interesante profundizar en la localización de dispositivos proporcionada por Sigfox, y estudiar qué funcionalidades puede proporcionar en futuros proyectos.

Como puede apreciarse, el auge de las redes inalámbricas, en concreto las LPWAN, implica un estudio continuo de las funcionalidades que ofrecen, con el fin de poder desarrollar aplicaciones y sistemas más eficientes y que proporcionen soluciones a los problemas más actuales de las comunicaciones inalámbricas.

ANEXO A. PROCESO DE SUSCRIPCIÓN A SIGFOX

Para la conexión del módulo FT4232H con el ordenador se utiliza el programa Docklight.

Primero se han de configurar los ajustes del proyecto desde la ventana mostrada en la Figura A-1:

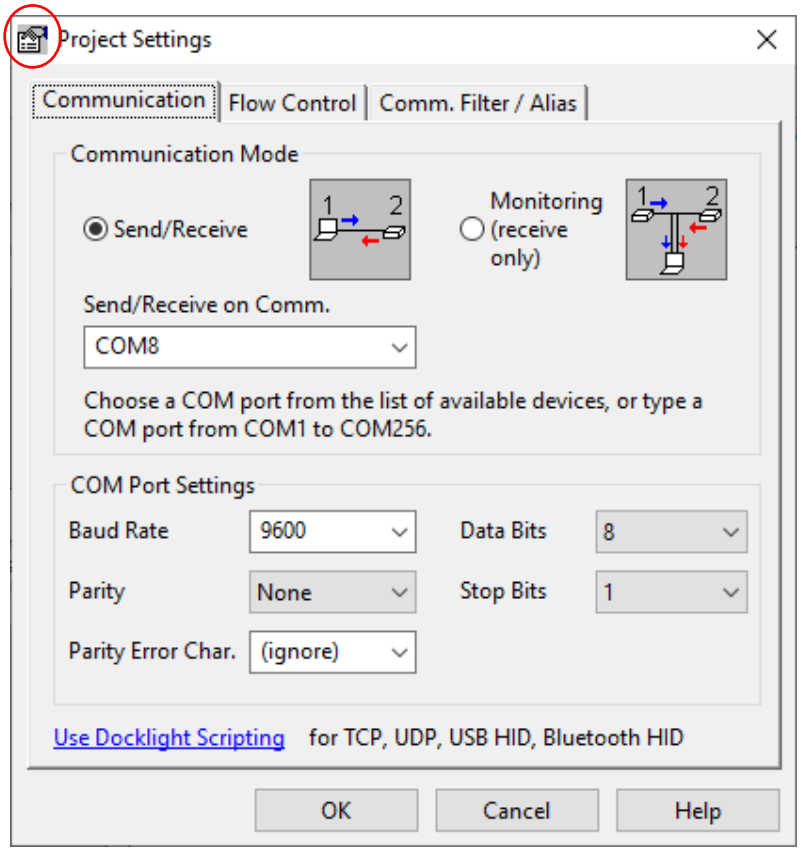


Figura A-1: Configuración ajustes de proyecto en Docklight

Todos los ajustes del puerto serie (Figura A-1) se pueden encontrar en el *datasheet* del módulo FT4232H [12], la selección del puerto depende de en qué puerto de la placa TTL se hagan las conexiones, en este caso se hace en el puerto A, y por lo tanto corresponde al puerto de menor número que aparezca, en este caso COM8 (el resto son 9, 10 y 11).

Una vez conectada la placa se comienza la comunicación, para ello se usarán comandos AT. Para comprobar que la conexión está bien hecha, se puede utilizar el comando "AT", también conocido como comando *dummy* porque no realiza ninguna acción y solo devuelve un "OK" (Figura A-2).

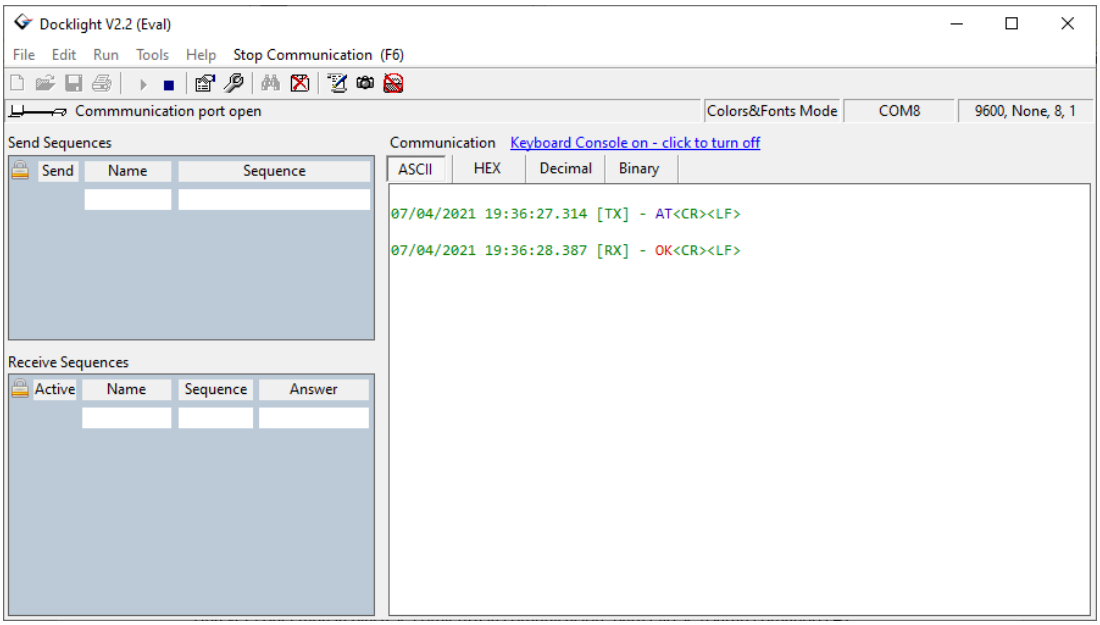


Figura A-2: Mensajes en Docklight- AT y respuesta

A continuación, se necesita obtener el número ID y el número PAC que caracterizan al módem Sigfox, para ello se utilizan los comandos AT$I=10 Y AT$I=11 [10] respectivamente (Figura A-3) y se obtiene lo siguiente:

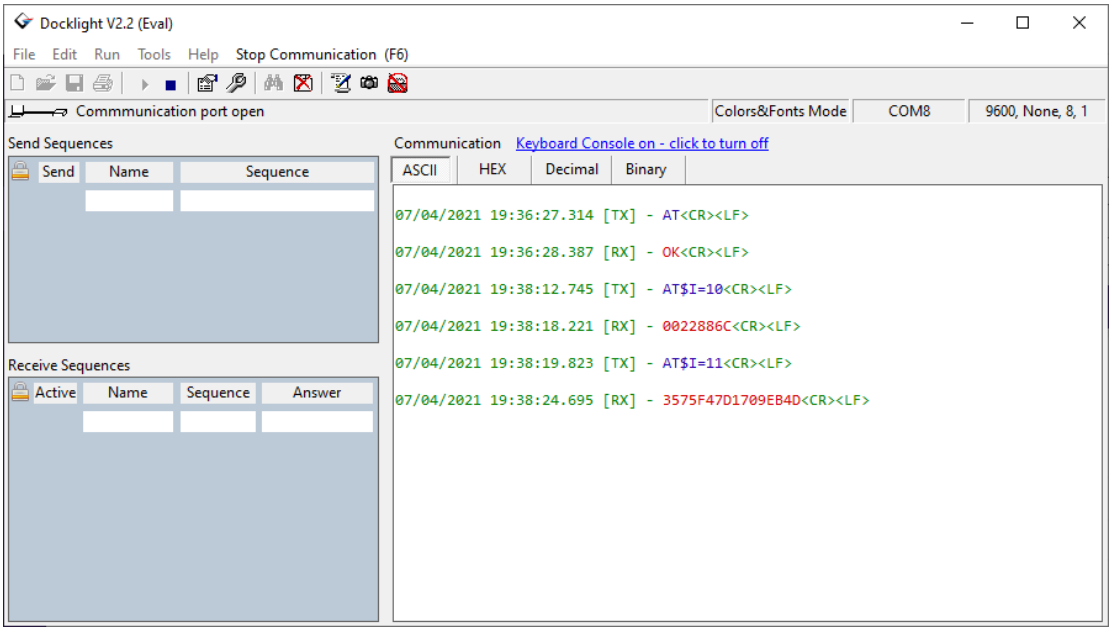


Figura A-3: Mensajes en Docklight - ID y PAC y respuestas

Una vez se conocen el ID y PAC del módem se debe dar de alta en la página de Sigfox. [15]

Primero se selecciona el país en el que se va a registrar, en este caso *Spain*, y se hace *click* en *next*. A continuación, se introducen el ID y el PAC (Figura A-4) que ha devuelto el módulo Sigfox, se introducen también los datos de la cuenta y se registrado.

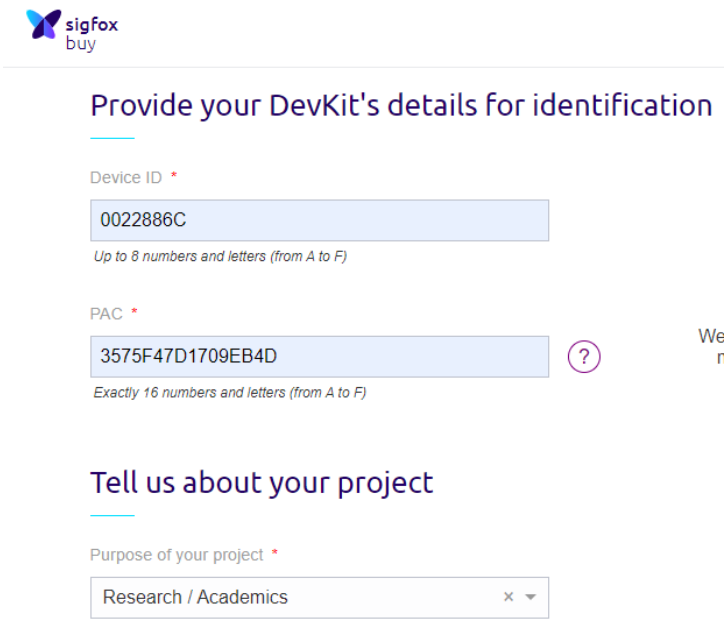


Figura A-4: Identificación del módulo en la página de Sigfox

En algunos casos es posible que haya que contratar el servicio antes de registrar el módulo, esto se puede hacer contactando con subscribe@sigfox.com

Una vez completado todo lo anterior se debe registrar el dispositivo en la página del Backend de Sigfox [7], donde se accede con la cuenta proporcionada anteriormente.

Primero se debe hacer un registro en el apartado *DEVICE TYPES*, haciendo *click* arriba a la derecha en *new* (Figura A-5).

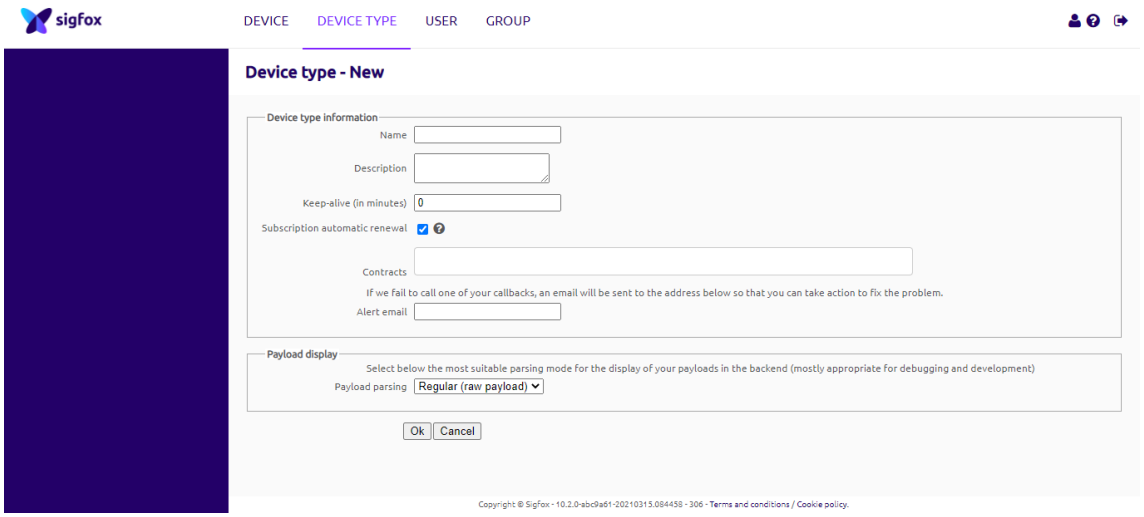


Figura A-5: Registro tipo de dispositivo Sigfox en la página de Sigfox

Se completa una vez seleccionado el contrato que se ha realizado y que está asociado a la cuenta y se termina haciendo *click* en OK (Figura A-6).

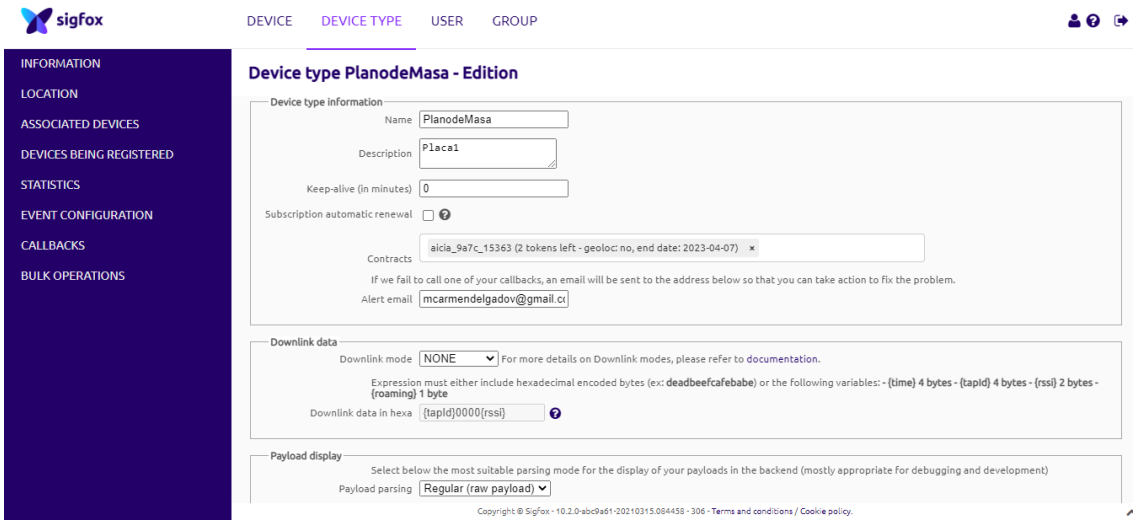


Figura A-6: Registro completo del tipo de dispositivo Sigfox en la página de Sigfox

A continuación, se hace *click* en *DEVICE* y arriba a la derecha en se hace *click* de nuevo en la pestaña *new*, donde habrá que rellenar la pestaña que aparece (Figura A-7).

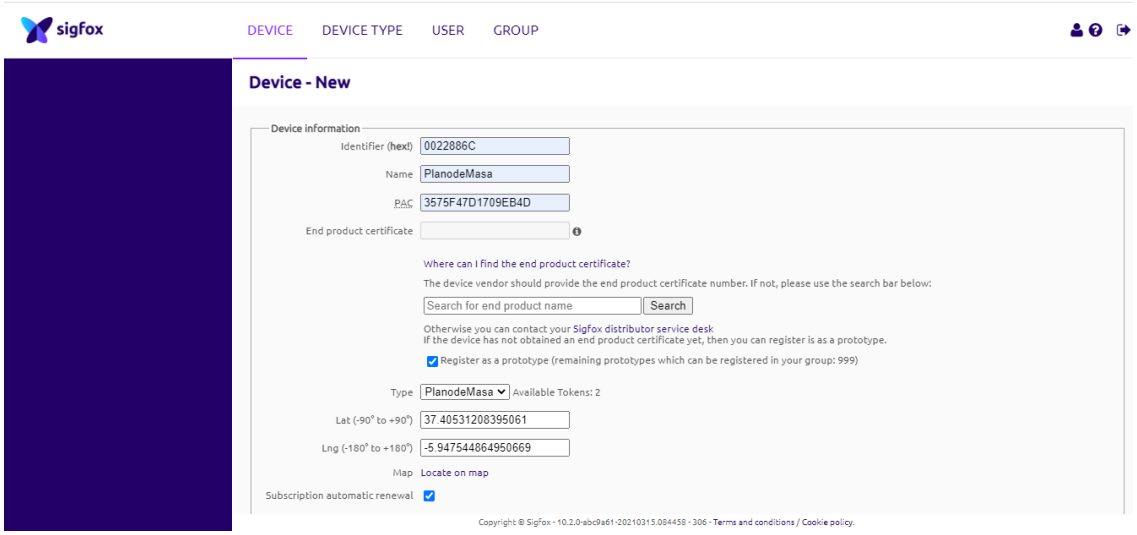


Figura A-7: Registro completo del dispositivo Sigfox en la página de Sigfox

Una vez completado el registro se puede comenzar la comunicación con el *backend*. Para esto se utiliza el puerto serie Docklight y mediante comando AT (en este caso AT$SF para mandar mensajes de hasta 12 bytes o AT$SB para mandar bits), es posible enviar un mensaje al *backend* de Sigfox (Figura A-8).

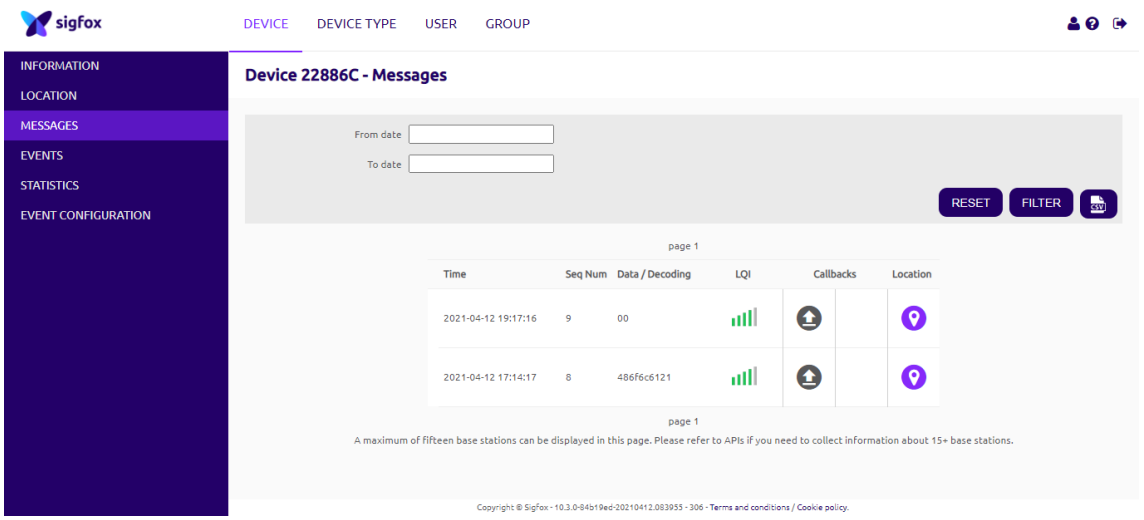


Figura A-8: Mensajes recibidos en el backend de Sigfox

REFERENCIAS

[1] «Dispositivos conectados (Internet de las cosas) a nivel mundial de 2019 a 2030, en Statista,» [En línea]. [Último acceso: Agosto 2021].

[2] S. Fernandez, «Xataka móvil,» 22 Mayo 2020. [En línea]. Available: https://www.xatakamovil.com/conectividad/nuevo-record-velocidad-fibra-optica-44-2tbps-wikipedia-5-3-segundos.

[3] K. Mekki, E. Bajic, F. Chaxel y F. Meyer, «A comparative study of LPWAN technologies for large-scale IoT deployment,» *ICT Express*, pp. 1-7, March 2019.

[4] A. Montes y L. Camacho, «Telemetría a través de redes de área extensa de baja potencia (LPWA) y en coexistencia con el internet de las cosas (IoT),» p. 2.

[5] J. E. Crespo, «Aprendiendo Arduino,» 7 Marzo 2018. [En línea]. Available: https://www.aprendiendoarduino.com/tag/backend-sigfox/. [Último acceso: Agosto 2020].

[6] C. Gomez, J. C. Veras, R. Vidal, L. Casals y J. Paradells, «A Sigfox Energy Consumption Model,» *Sensors*, 2019.

[7] «Sigfox Backend,» [En línea]. Available: https://backend.sigfox.com/auth/login.

[8] «Zephyr Project,» [En línea]. Available: https://docs.zephyrproject.org/2.5.0/boards/arm/nucleo_l452re/doc/index.html.

[9] «STM32 Datasheet,» [En línea]. Available: https://os.mbed.com/platforms/ST-Nucleo-L152RE/.

[10] «AXSF10 ANT21868 Datasheet,» [En línea]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/763729/ONSEMI/AX-SF10-ANT21-868.html.

[11] «FTDI Chip,» [En línea]. Available: https://ftdichip.com/products/ft4232h-56-mini-module/.

[12] «FT4232H Mini Module Datasheet,» [En línea]. Available: https://www.farnell.com/datasheets/1915281.pdf.

[13] «Disk91 - The IoT Blog,» 20 Diciembre 2014. [En línea]. Available: https://www.disk91.com/2014/technology/internet-of-things-technology/sigfox-downlink-howto/. [Último acceso: 7 Septiembre 2021].

[14] «FreeRTOS,» [En línea]. Available: https://www.freertos.org/.

[15] «Activate Sigfox,» [En línea]. Available: https://buy.sigfox.com/activate.

[16] «Narrow Band IoT,» [En línea]. Available: https://en.wikipedia.org/wiki/Narrowband_IoT.

[17] J. Salazar, «Redes Inalámbricas,» [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/100918/LM01_R_ES.pdf.

[18] «Semantic Web Builder,» 2020. [En línea]. Available: http://www.semanticwebbuilder.org.mx/es_mx/swb/Sistemas_Embebidos_Innovando_hacia_los_Sistemas_Inteligentes_.