UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UN REGISTRADOR COMPACTO BASADO EN TECNOLOGÍA DE ULTRA BAJO CONSUMO Y COMUNICACIÓN SIGFOX PARA INTERNET DE LAS COSAS (IOT) CON APLICACIÓN EN PROYECTOS DE CIUDADES INTELIGENTES

AUTOR: JUAN RAMÓN SÁNCHEZ VICEDO

TUTOR: JULIO GOMIS-TENA DOLZ

COTUTOR: MARCOS ANTONIO MARTÍNEZ PEIRÓ

Curso Académico: 2019-20

Agradecimientos

Aprovecho esta ocasión para hacer constar mi agradecimiento a todas aquellas personas, sin las que su apoyo, no hubiera sido posible la realización de este Trabajo Final de Máster.

En primer lugar, mi agradecimiento más sincero a mi tutor Julio Gomis-Tena, por su apoyo y paciencia en las largas jornadas de consultas que hemos realizado. También a Jesús Sandia, técnico del laboratorio del Departamento de Ingeniería Electrónica, quien siempre se ha mostrado atento y amigable a mis peticiones. También a mi cotutor Marcos Antonio Martínez, quien también ha participado en muchas de nuestras reuniones, aportando grandes ideas y consejos.

No puedo olvidarme de mis padres y familia, quiénes me han apoyado en todo momento, dándome las fuerzas que en ciertos momentos creía que no tenía. Pues ellos siempre han confiado en mí y en mis posibilidades.

Por último, quiero también aquí nombrar a mi pareja, Laura, pues ha sido ella quien, en mayor medida, ha tenido que soportarme en mis momentos de estrés y ha sido ella quien me ha dado las fuerzas necesarias para continuar.

A todos y cada uno de ellos, gracias.

Resumen

El trabajo pretende abordar el diseño electrónico de un registrador de datos compacto, de pequeño tamaño y muy bajo consumo que pueda ser utilizado en entornos de ciudad inteligente. Para ello, debe registrar una variable analógica y almacenarla hasta el momento de comunicación con la plataforma IoT.

Para mantener el bajo consumo se diseñará con un procesador de bajo consumo y con un modem SigFox, que permite una baja tasa de bits y un número limitado de paquetes. El objetivo es mantener el equipo operativo sin cambios de baterías en más de un año. Finalmente, se pretende alcanzar una solución de bajo coste que permita realizar miles de instalaciones económicamente viables en redes de suministro de agua en la ciudad.

Palabras clave: Registrador; Microcontrolador; Internet de las Cosas; Ultra bajo Consumo; SigFox; Smart City; Monitorización; Redes de distribución de agua.

Abstract

The work aims to address the electronic design of a compact data logger, with small size and very low power consumption that can be used in smart city environments. The small logger will have an only analog variable, storing it until the time of communication with the IoT platform.

To maintain the low consumption, it will be designed with a low consumption processor and a SigFox modem, which allows a low bit rate with a limited number of sent packages. The goal is to keep the equipment operating without battery changes in more than a year. Finally, it is intended to achieve a low-cost solution that allows thousands of economically viable installations in water supply networks in the city.

Keywords: Logger; Microcontroller; Internet of Things; Ultra low power; SigFox; Smart City, Monitoring, Water supply networks.

Resum

El treball pretén abordar el disseny electrònic d'un registrador de dades compacte, de dimensions reduïdes i molt baix consum que es pot utilitzar en entorns de la ciutat intel·ligent. Per tant, ha de registrar una variable analògica i emmagatzemar-la fins al moment de comunicació amb la plataforma IoT.

Per mantenir el consum es dissenyarà amb un processador de consum baix i amb un mòdul SigFox, que permet una baixa taxa de bits i una quantitat limitat de paquets. L'objectiu és mantenir l'equip operatiu sense canvis de bateries durant més d'un any. Finalment, es pretén assolir una solució de baix cost que permet realitzar milers d'instal·lacions econòmicament viables en xarxes de subministre d'aigua a la ciutat.

Paraules clau: Registrador; Microcontrolador; Internet de les Coses; Ultra baix consum; SigFox; Smart City; Monitorització; Xarxes de distribució d'aigua

ÍNDICE DE DOCUMENTOS

ÍNDICE MEMORIA

ÍNDICE PRESUPUESTO

ÍNDICE ANEXOS

DOCUMENTO I:

MEMORIA

1. OBJETIVO Y JUSTIFICACIÓN

El objetivo principal de este Trabajo Fin de Máster es el desarrollo de un sistema para la monitorización continua e inteligente de redes de distribución de aguas.

Actualmente con el desarrollo de las nuevas tecnologías de comunicación, han surgido nuevos conceptos como el IoT, la industria 4.0, las Smart-cities etc. El desarrollo ha sido impulsado por el reducido precio y consumo de las nuevas tecnologías, incluso se ha llegado a considerar tales avances como la Cuarta Revolución Industrial. La gran reducción del precio y del consumo ha permitido la creación de grandes de redes de sensores inteligentes, conectados entre sí.

El proyecto está impulsado por la empresa Global Omnium encargada de llevar el mantenimiento de la red de distribución para Aguas de Valencia. Actualmente, la monitorización se realiza en puntos clave de la instalación, estaciones de bombeo y zonas conflictivas. El sistema actual es incapaz de detectar la mayoría de los problemas, como fugas en determinados sectores, centrándose únicamente en los de mayor importancia. Esta incapacidad viene determinada por la falta de sensórica actualmente instalada, ya que no permite la modelización de la red completa.

El escaso número de sensores actualmente empleados viene justificado por el coste actual de los sistemas comerciales. Se trata de sistemas con comunicaciones convencionales como la red móvil 3G, los que además de tener un precio elevado, su elevado consumo los imposibilita para su empleo mediante baterías, obligándose a conectarlos a la red eléctrica.

Debido a la necesidad expuesta, se plantea el desarrollo de un sistema de adquisición (Datalogger) de entrada genérica para su empleo en una red de monitorización inteligente del sistema de distribución de aguas. Como aspectos fundamentales para su desarrollo se tendrán en cuenta el consumo y el coste. Por eso, se evaluarán las nuevas tecnologías de comunicación que permiten enlaces a larga distancia con un consumo y precio reducido.

En primer lugar, se han definido las características a imponer en el sistema propuesto. Una vez, definidas sus características se han evaluado las distintas tecnologías, seleccionando la más apropiada para la aplicación en cuestión. Finalmente, se han seleccionado los dispositivos necesarios mediante un criterio técnico-económico y se ha procedido a su ensamblaje en conjunto.

Con el dispositivo desarrollado el siguiente paso ha sido la definición de su funcionamiento y posterior programación. A partir del prototipo inicial se han propuesto una serie de mejoras para una futura versión final del dispositivo.

2. ANÁLISIS DE REQUISITOS DE APLICACIÓN

El sistema propuesto debe ser capaz de adaptarse a las condiciones de trabajo que requiere la aplicación de estudio. Los sistemas de distribución de agua se tratan de grandes canalizaciones subterráneas de tuberías. Los aspectos más importantes que controlar son el caudal y la presión.

Atendiendo a las características de instalación se pueden definir los siguientes requisitos:

- Escalable a un gran número de dispositivos.
- Permitir la comunicación a grandes distancias.
- Bajo coste para así no incrementar el coste de la instalación completa.
- Protección suficiente para soportar ambientes húmedos e incluso inmersiones en agua.
- El consumo debe ser reducido, de forma que pueda ser alimentado a través de baterías para los puntos en los que no se disponga de alimentación eléctrica.

Los sensores de caudal y presión empleados en el sector se tratan de sensores industriales con salidas normalizadas, habitualmente analógicas de 0-10 V o 4-20 mA. Estos sensores destacan por su elevada precisión, pero requieren de calibraciones periódicas.

Por otra parte, las redes de distribución de agua se tratan de sistemas con grandes inercias y constantes de tiempo, lo que se traduce a sistemas lentos. Los cuales no requieren de una monitorización de sus variables a frecuencias altas, sino que para su monitorización es completamente válido realizar mediciones puntuales.

- Adaptable a la salida del sensor: 0-10 V o 4-20 mA.
- Ajustable a las características del sensor empleado: ganancia, offset…
- Facilidad de la calibración periódica del sensor.
- Debe realizar mediciones periódicas.
- Capaz de detectar la desconexión, rotura o problemas en el sensor.
- Debe controlar la alimentación del sensor para reducir su consumo.

En cuanto a su manipulación debe ser fácilmente programable, pudiéndose ajustar sin necesidad de cambiar ningún tipo de conexión ni exponer el circuito electrónico al ambiente de trabajo.

- Facilidad de uso.
- Ajustable sin realizar modificaciones físicas en el dispositivo.
- Ajustable a distancia remota, para impedir el desplazamiento de un operario a campo.

3. DESCRIPCIÓN DE LA SOLUCIÓN

Atendiendo a los requisitos expuestos en el apartado anterior, se ha procedido a definir las características técnicas que debe poseer la solución. El sistema propuesto puede ser dividido en diversos subsistemas

- Subsistema de comunicación.
- Subsistema de adquisición del sensor.
- Subsistema de interfaz con el operador.
- Subsistema de alimentación.
- Subsistema controlador.

3.1. Subsistema de comunicación

El subsistema de comunicación permite el envío de las mediciones de presión o caudal tomadas, hasta un centro de control o directamente a la nube. Se trata del aspecto más relevante del sistema ya que es lo que le va a permitir la diferenciación con productos comerciales ya existentes en el mercado. Para su selección, en primer lugar, se han estudiado las distintas tecnologías existentes, seleccionando la tecnológica más adecuada atendiendo a los siguientes requisitos:

- Largo alcance.
- Bajo consumo.
- Bajo coste.
- Escalable.
- Fácilmente ampliable.
- Permita una comunicación bidireccional.

Con estos requisitos se ha seleccionado la tecnología a emplear y tras ello se ha realizado un estudio técnico-económico para la selección del dispositivo final.

3.2. Subsistema de adquisición

El subsistema de adquisición es el encargado de realizar la interfaz entre el controlador del sistema y el sensor. El subsistema debe incluir:

- Un convertidor analógico-digital para la digitalización de la señal analógica del sensor.
- Circuito de adaptación de la señal (0-10 V o 4-20 mA) a la entrada del convertidor.
- El control de la alimentación del sensor.

3.3. Subsistema de interfaz con el operario

El subsistema de interfaz con el operario ha de permitir configurar el equipo, calibrar el sensor y comprobar que las mediciones son correctas de una forma rápida, sencilla y sin requerir de sistemas complejos.

Para tal objetivo, se ha definido un subsistema doble. Por una parte, un display que permita rápidamente comprobar que las lecturas del sensor son correctas. Y, por otra parte, una conectividad Bluetooth que permita la configuración del dispositivo desde cualquier dispositivo móvil a partir de una aplicación. Para dotar al sistema de seguridad, así como estanqueidad se ha evitado el uso de interruptores o botones en el exterior, sustituyéndose por interruptores magnéticos protegidos en el interior. Esto también supone una reducción del coste de producción, ya que los interruptores/botones con protección exterior IP65 o superior, tienen un elevado coste.

3.4. Subsistema de alimentación

El subsistema de alimentación se encarga de proveer la energía necesaria al resto de subsistemas, como se ha mencionado anteriormente el sistema puede situarse en lugares en los que no hay conexión a la red eléctrica en sus proximidades. Por lo que se ha diseñado un sistema capaz de funcionar con baterías, incluyéndole un cargador de baterías de Litio.

3.5. Subsistema controlador

El subsistema controlador, es el cerebro del dispositivo, es el encargado de comunicarse con los distintos subsistemas para que cada uno desarrolle su función cuando es requerido. Debido a la aplicación a desarrollar se ha optado por el empleo de un microcontrolador, ya que se requiere que el consumo sea lo más reducido posible y no se requiere ningún tipo de algoritmo complejo de procesado.

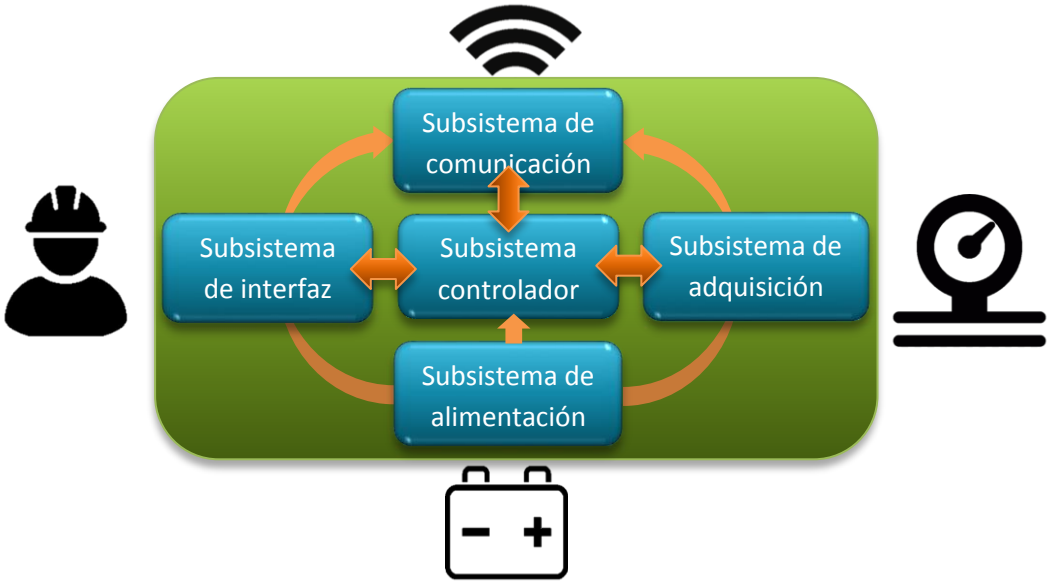


Figura 1. Esquema relación entre los distintos subsistemas. Fuente: elaboración propia.

4. ESTUDIO DE LAS TECNOLOGÍAS DE COMUNICACIÓN

Existen distintas tecnologías de comunicación inalámbrica, atendiendo al alcance de estas se pueden clasificar en:

- **WPAN (Wireless Personal Area Network):** Se encuentran dentro de la norma 802.15 del IEEE. Su empleo se limita a la interconexión de dispositivos personales, posee un alcance limitado a decenas de metros. Entre las distintas tecnologías destaca el Bluetooth y el ZigBee.
- **WLAN (Wireless Local Area Network):** quedan definidas dentro de la norma IEEE 802.11. La norma recoge las distintas versiones de Wi-Fi existentes, se trata de una tecnología de alcance medio (hasta 1 km de distancia) con un ancho de banda y consumo elevado.
- **WNAN (Wireless Network After Next): se** encuentran dentro del estándar IEEE 802.16. La más común es la tecnología WiMAX, capaz de proporcionar altas velocidades de transmisión de datos a grandes distancias.
- **WWAN (Wireless Wide Area Network):** se caracterizan por alcanzar distancias de hasta 50 Km de distancia, suelen requerir frecuencias con licencias.

Dentro del último grupo (WWAN), con el desarrollo de nuevas tecnologías y el auge de los dispositivos inteligente conectados (IoT) ha aparecido un nuevo grupo de tecnologías clasificadas como **LP-WAN (Low Power Wide-Area Networks).**

LP-WAN son capaces de mantener comunicaciones a larga distancia, pero reduciendo drásticamente su consumo permitiendo además un elevado número de dispositivos interconectados en la red. La reducción del consumo se consigue mediante una reducción del ancho de banda por lo que también son conocidas como tecnologías *NarrowBand*.

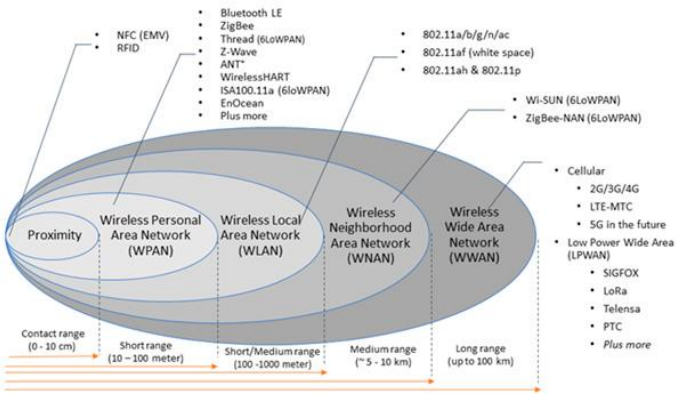


Figura 2. Esquema de distintas tecnologías de comunicación de acuerdo con el alcance de transmisión. Fuente: (1).

Como se ha descrito anteriormente, el sistema propuesto requiere de conexiones a largas distancias, de elevados números de nodos conectados y de una tasa de envío reducida. Por lo tanto, las tecnologías más acordes con el sistema son las tecnologías recogidas en el grupo LP-WAN. Para una correcta elección de la tecnología concreta se ha profundizado en el estudio de las más importantes. (2) (3).

4.1. LoraWAN

Se basa en técnicas de modulación de espectro extendido derivadas de la tecnología CCS. Emplea bandas libres de radiofrecuencia por debajo del Giga-Hertzio 433 MHz, 868 MHz (Europa), 915 MHz (América del norte y Australia). Permite alcances de hasta 10 km en áreas rurales, con un bajo consumo de potencia. La tecnología LoRa cubre la capa física mientras que el protocolo LoraWAN cubre las capas superiores.

Utiliza una modulación de espectro ensanchado patentada, esto permite que LoRa pueda ajustar la velocidad de datos a costa de la sensibilidad. Además, emplea una codificación *Forward Error Correction* para mejorar la resistencia frente a interferencias. Alcanza sensibilidades de 150 a 170 dB.

LoraWAN es un protocolo de capa de control de acceso al medio (MAC) basado en la nube. Pero actúa principalmente como protocolo de capa de red para administrar la comunicación entre puertas de enlace LPWAN (*Gateways*) y dispositivos finales como un protocolo de enrutamiento. Por lo tanto, una red LoraWAN estaría formada por un conjunto de nodos finales que enviarían los datos a un conjunto de Gateways, que finalmente los subirían a la nube a través de una conexión a internet.

4.2. SigFox

SigFox utiliza 200 kHz de las bandas disponibles públicamente y sin licencia para intercambiar mensajes de radio por aire (868 a 869 MHz en Europa y 902 a 928 MHz en América). SigFox utiliza la tecnología Ultra Narrow Band (UNB) combinada con la modulación DBPSK y GFSK. Cada mensaje tiene 100 Hz de ancho y se transfiere a una velocidad de datos de 100 o 600 bits por segundo, según la región.

SigFox no es únicamente una tecnología o protocolo de comunicación, sino que ha creado una red similar a las redes de telefonía celular. Con la diferencia de que los dispositivos que estén conectados a la red no están vinculados a una única estación base, esto significa, que cualquier estación puede recibir la información y transmitirla hacia la nube. En caso de recibirse el mensaje en varias estaciones base, este se filtra mediante su identificador para evitar duplicidades.

SigFox proporciona un servicio web: www.backend.sigfox.com, donde se pueden consultar los mensajes recibidos por cada uno de los dispositivos o crear *callbacks* para enviarlos a otras páginas webs, servidores o a un correo electrónico.

Actualmente está presente en la mayoría de los países de Europa, en su página web se puede comprobar la cobertura real de la red. En caso de no disponer cobertura, SigFox permite la instalación de repetidores de señal o incluso la disposición de nuevas estaciones base. Los principales inconvenientes que presenta SigFox son: que para poder operar en la red SigFox se requiere de una subscripción anual por dispositivo y la limitación de mensajes diarios, ya que está limitada a 140 mensajes de subida (*uplink*) y 4 mensajes de bajada (*downlink*). Ese límite viene definido por términos legales, ya que al tratarse de una banda libre ISM, se permite un uso inferior al 1% diario.

$$\%Uso = \frac{T_{uso}}{T_{diario}} = \frac{144\ msg \cdot 6\ s/msg}{24\ h \cdot 60\ min/h\ \cdot 60\ s/min} = 1\% \qquad \text{Ec.1}$$

4.3. Tecnologías celulares: LTE-M y NB-IoT

LTE-M es la abreviación de LTE Cat-M1, se trata de una tecnología reciente de LPWAN, desarrollada para dispositivos IoT. Es una tecnología celular de ancho de banda estrecho, que permite la transmisión de pequeñas cantidades de datos a internet con un consumo de energía reducido. Esta tecnología emplea la misma red que las tecnologías LTE convencionales, tiene un ancho de banda grande en comparación con el resto de LPWAN (1.4 MHz), lo que permite velocidades de datos de hasta 1 Mbps.

NB-IoT, emplea una ampliación de la red LTE estándar, pero limitando su ancho de banda a 200 kHz, emplea una modulación OFDM en el caso de un número reducido de dispositivos o la modulación SC-FDMA cuando hay un número mayor de dispositivos involucrados. Al tener un ancho de banda más reducido, la tasa de datos se ve reducida a un máximo de 62.5 kbps, aumentándose también la latencia.

4.4. Comparativa diferentes tecnologías

Para la selección de la tecnología más apropiada se han resumido las características, atendiendo a alcance de transmisión, velocidad de transferencia de datos, consumo, coste de mantenimiento y despliegue.

Tabla 1. Comparativa entre las distintas tecnologías de comunicación estudiadas. Fuente: elaboración propia.

	LoraWAN	SigFox	LTE-M	NB-IoT
Rango	< 10 km	< 13 km	<10 km	< 15 km
Banda (MHz)	433	868/915	700-900	700-900
Ancho de banda	<500 kHz	100 kHz	1.4 MHz	200 kHz
Tasa de envío	<10 kbps	< 100 kbps	< 1Mbps	< 150 kbps
Coste anual	Medio	Medio	Alto	Alto
Coste inicial	Alto	Bajo	Medio	Medio
Consumo	Bajo	Bajo	Alto	Medio

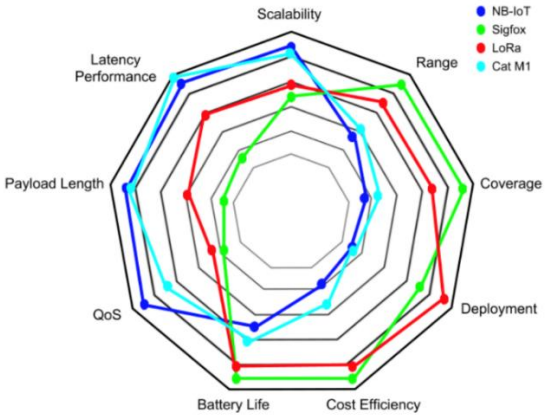


Figura 3. Comparativa entre las distintas tecnologías LP-WAN. Fuente: (4).

Teniendo en cuenta la aplicación a la que va destinada el proyecto, redes de distribución de agua, como se he expuesto anteriormente se requiere de una tecnología con largo alcance, una tasa de envío baja y un consumo reducido. Por lo tanto, las tecnologías celulares quedan descartadas ya que suponen un mayor consumo que las tecnologías SigFox y LoraWAN.

Finalmente, se ha seleccionado la tecnología SigFox ya que, para el escenario estudiado, Valencia, la red SigFox proporciona cobertura prácticamente en la totalidad del territorio. Por lo que el coste de despliegue de la red sería proporcional al número de dispositivos, mientras que en una red LoraWAN supone un coste inicial de creación de la red. Además de reducir el coste inicial, se evitan los costes de mantenimiento de la red y los costes de mantenimiento del servidor para el almacenamiento de datos ya que ambos costes son asumidos por la empresa SigFox.

5. ESTUDIO TÉCNICO-ECONÓMICO Y SELECCIÓN DE COMPONENTES

El sistema propuesto ha sido dividido en los siguientes subsistemas:

- Subsistema de comunicación: módulo SigFox.
- Subsistema de adquisición del sensor: convertidor analógico-digital.
- Subsistema de interfaz con el operador: display numérico y módulo Bluetooth.
- Subsistema de alimentación: cargador de batería de polímero/ion Litio y regulador.
- Subsistema controlador: microcontrolador.

5.1. Subsistema de comunicación: módulo SigFox

Para la selección del módulo SigFox se han estudiado distintos módulos:

Tabla 2. Comparativa entre distintos módulos SigFox. Fuente: elaboración propia

	RC1682 RadioCrafts	AX-SFEU On-Semiconductor	OL2385 (NXP)	eRIC-SIGFOX	ATA8520E (Atmel)
Interfaz	UART	UART	UART/SPI	UART	SPI
Modos Uplink/Dowlink	Si	Sí	Sí	Sí	Sí
Consumo recepción	31 mA	10 mA	10 mA	10 mA	10 mA
Consumo envío	59 mA	49 mA	33 mA	49 mA	32.7 mA
Salida de antena	Sí	Requiere filto RF	Requiere filtro RF	Sí	Requiere filtro RF
Consumo modo Sleep	2 µA	0.5mA/1.3 µA/100nA	20 µA	0.5mA/1.3 µA/100nA	5nA
Alimentación	2.8-3.6 V	1.8-3.6 V	1.8-3.6 V	1.8-3.6 V	1.9-3.6 V o 2.4-5.5V
Sensibilidad	-126 dBm	-126 dBm	-124 dBm	-126 dBm	-126 dBm
Rango de T^{o}	-30 / 85^{o}C	-40 / 85^{o}C	-40 / 85^{o}C	-40 / 85^{o}C	-40 / 85^{o}C
Precio	22 €	3,82 €	3,20 €	16,50 €	2,20 €

Figura 4. Ejemplo de dos módulos SigFox. RC1682 (izq.) y eRIC-SIGFOX (dcha.). Fuente: Datasheet RC1682 y eRIC-SIGFOX

Entre los distintos módulos evaluados, se puede distinguir dos tipos: los que son únicamente el integrado del transceptor de radiofrecuencia como el ATA8520E o el AX-SFEU y los que son módulos totalmente integrados (*Plug&Play*) como el eRIC-SIGFOX o el RC1682. El precio de los módulos *Plug&Play* es más elevado alrededor de 15-25€ frente 2-4€ para los transceptores. El aumento del precio es debido a que estos módulos además del transceptor de radiofrecuencia incluyen un microcontrolador o dispositivo programable que efectúa la comunicación con el

transceptor y la interfaz de salida del módulo, facilitando así su uso. También suelen incluir los circuitos necesarios de acondicionamiento de las pistas de radiofrecuencia. Estas características simplifican en gran medida su implementación, pero además de suponer un aumento del coste, suponen un aumento del consumo.

Debido a este aumento del consumo de un 50% (49mA frente a 32.5mA) y del coste se ha optado por las soluciones tipo transceptor concretamente a la opción ATA8520E ya que es la que mejores características presenta en cuanto a consumo y precio.

5.2. Subsistema de adquisición

Para el subsistema de adquisición se requiere del uso de un convertidor analógico digital y el circuito de adaptación de voltaje para entradas de 0-10V o entradas a corriente de 4-20 mA.

Existen distintos tipos de convertidores analógico-digitales. Dependiendo del método de conversión alcanzan unas especificaciones u otras. (5)

- Convertidores por aproximaciones sucesivas (SAR).
- Convertidores basados en Flash.
- Convertidores Pipeline.
- Convertidores Sigma-Delta.

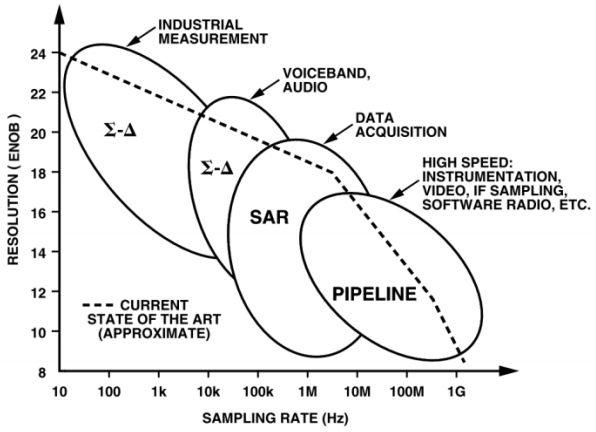


Figura 5. Comparación entre las distintas tecnologías de ADC. Fuente: (6)

Para la aplicación en cuestión, se requiere la lectura de la salida del sensor y del nivel de la batería. Ambas medidas no requieren de una elevada tasa de muestreo, como se ha justificado anteriormente, se requieren únicamente de mediciones puntuales. Por otra parte, si es importante la precisión de la conversión en la lectura de sensor, ya que, para algunos sensores pequeñas variaciones en sus salidas pueden suponer grandes variaciones de presión. Mientras que, el nivel de batería es solamente un indicador el cual acepta pequeños errores en su medición.

Por estos motivos se ha optado por la selección de un convertidor Sigma-Delta para la lectura del sensor, mientras que para la lectura del nivel de batería se ha optado por un convertidor Pipeline o SAR.

Una tarjeta de adquisición en la mayoría de los casos requiere de un cristal externo para proporcionar los ciclos de reloj necesarios y de una referencia de tensión estable para realizar la conversión. Como es el caso de la tarjeta de adquisición AD7734

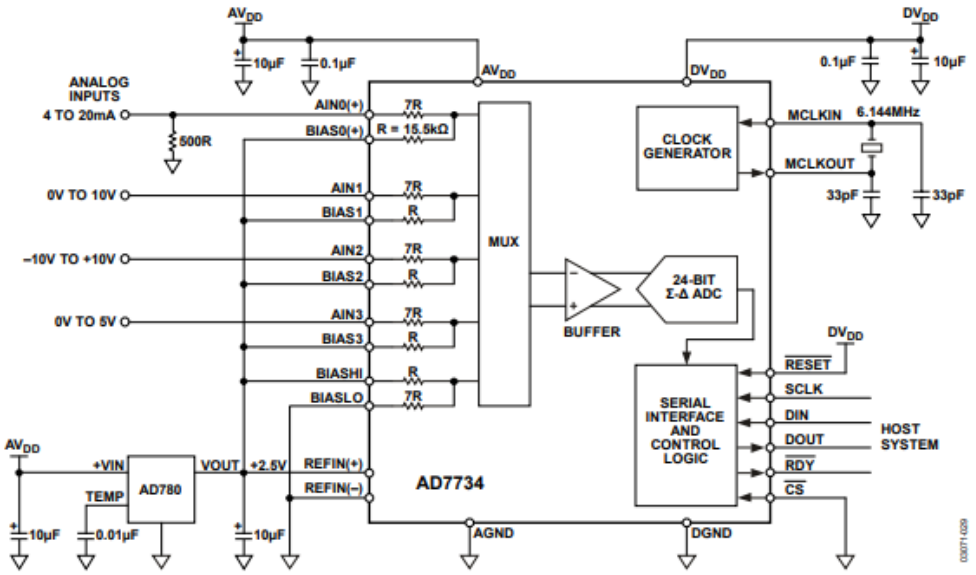


Figura 6. Conexión tarjeta de adquisición AD7734. Fuente: AD7734 Datasheet.

Con el objetivo de reducir el tamaño del sistema a desarrollar, así como de reducir sus costes, se ha planteado el uso de los convertidores analógicos-digitales internos del propio microcontrolador. La mayoría de los microcontroladores poseen periféricos ADC de bajas prestaciones, no obstante, existen dispositivos especializados para el tratamiento de señales analógicas con convertidores Sigma-Delta y referencias de tensiones internas.

Por otra parte, en cuanto al sistema de adaptación de la entrada analógica se han tenido en cuenta, un diseño para reducir la tensión de la salida del sensor a la entrada de la ADC y un diseño para convertir la corriente de salida del sensor (4-20 mA) a la tensión de entrada del ADC.

Para reducir la tensión en caso de tratarse de un sensor de salida en tensión se ha empleado un divisor de tensiones y en caso de tratarse de salida de corriente se convierte a tensión mediante una resistencia.

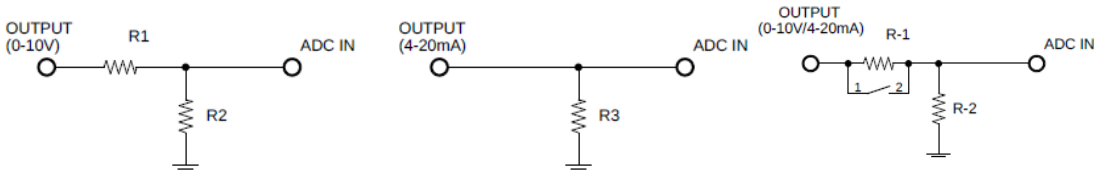


Figura 7. Adaptación de tensiones a la entrada de la ADC. Para salida en tensión, salida en corriente y conjunta de izquierda a derecha. Fuente: elaboración propia.

Ambos sistemas se unen mediante un interruptor. El interruptor se realiza mediante un transistor MOSFET de forma que mediante software se permite el cambio entre entrada en tensión (MOSFET en estado de corte) y la entrada en corriente (MOSFET en conducción).

5.3. Subsistema de interfaz con el operador: display numérico y módulo Bluetooth

El subsistema de interfaz con el operador está formado por dos subsistemas:

- Display numérico: que permita una rápida lectura del sensor.
- Conexión Bluetooth que permita la configuración del dispositivo.

Tabla 3. Comparativa entre distintos Displays. Fuente: elaboración propia.

	TM1637	FDCC1602B	SSH1106
Interfaz	I2C	Paralelo (o I2C)	I2C
Tipo	4 dígitos y 7 segmentos	16 caracteres y 2 líneas	Oled de 1,3" (128x64)
Alimentación	5 V	5V / 3V	5V / 3V
Consumo	0.2-80 mA	4-120 mA	< 27 mA (Sleep <5 µA)
Tamaño	30 x 14 mm	64.5 x 16.4 mm	1,3"
Precio	1,70 €	5,70 €	6,20 €

Figura 8. Displays estudiados TM1637, FDCC1602B y SSH1106 de izquierda a derecha: Fuente: TM1637, FDCC1602B y SSH1106 Datasheets.

En cuanto al display, se ha seleccionado el SSH1106 ya que funciona con el protocolo estándar I2C, permite modos de bajo consumo y al tratarse de una pantalla Oled totalmente configurable permite una mayor flexibilidad en cuanto a mostrar los datos de interés.

Tabla 4. Comparativa módulos Bluetooth. Fuente: elaboración propia.

	RN-42 Microchip	HC-05
Interfaz	UART	UART
Comandos	ASCII	AT
Modo	Transparente SPP o HCI	Transparente SPP o HCI
Consumo (sleep/conectado/enviando)	26 µA / 3mA / 30 mA	x/x/35 mA
Alimentación	3-3.6V	3,3 V
Tasa de transferencia	SSP (240 kps), HCI< 3Mbps	HCI< 2Mbps
Extras	Compatible con Android y iOs	
Precio	8,2€	5,4 €

Figura 9. Módulos Bluetooth, RN-42 (izq.) y HC-05 (dcha.): Fuente: RN-42 y HC-05 Datasheets.

En cuanto al módulo bluetooth se ha considerado el empleo de un módulo Bluetooth 2.1 ya que tiene mayor compatibilidad con dispositivos y no se requiere un ultra bajo consumo ya que no se va a emplear durante su funcionamiento normal. Únicamente se emplea durante la configuración del dispositivo, además los módulos Bluetooth 2.1 son aproximadamente entre un 30-50% más económicos que los Bluetooth 4.0. Entre los dos analizados se ha seleccionado el RN-42 ya que permite modos de bajo consumo y es compatible con dispositivos i-OS.

5.4. Subsistema de alimentación: cargador de batería de Litio y regulador

Se ha decidido añadir un circuito de carga de baterías de Litio para que no sea necesario abrir ningún compartimento para cambiar las baterías. El exceso de humedad de los ambientes en los que el dispositivo va a operar podría dañar los componentes, ya que, al abrirlas la humedad puede quedar atrapada en el interior o se pueden dañar las juntas aislantes permitiendo intrusiones futuras.

También al incluir su batería y su propio cargador le proporciona características de un producto comercial, pues proporciona todo lo necesario para su funcionamiento. Como cargador se ha seleccionado el integrado MCP73833, se trata de un integrado de pequeño tamaño con capacidad de carga lineal tanto para corriente o tensión constante.

Tabla 5. Características del cargador MCP73833. Fuente: elaboración propia.

	MCP73833
Tipo	Cargador lineal baterías
Corriente de carga	Programable hasta 1A
Compatibilidad	Baterías de una celda (ion-Li o Li-po)
Extras	Detección automática de carga completa Protección por temperatura
Tipo carga	Programable a Corriente/Tensión constante
Precio	0,65 €

La tensión de la batería debe ser regulada a una tensión constante para la alimentación de todos los componentes del sistema. Para ello se emplea un regulador cuya entrada se adapte al rango de tensiones de una batería de una celda y su salida esté dentro del rango de la alimentación de cada uno de los componentes.

Tabla 6. Estimación del consumo máximo para cada componente. Fuente: elaboración propia.

Componente	Tensión de alimentación (V)	Consumo máximo (mA)
ATA8520E	1.9-3.6 o 2.4-5.5	32.7
SSH1106	3-5	27
RN-42	3-3.6	50
Microcontrolador (aprox.)	2-3.6	15

Para la tensión de alimentación se ha seleccionado la mínima aceptable por todos los componentes 3.0 V ya que así se puede extraer la máxima energía de la batería y el consumo se ha limitado a 150 mA. Por otra parte, se requiere de la alimentación del transductor de presión. Para definir el rango de tensiones de alimentación se han evaluado distintos sensores comerciales:

Tabla 7. Estudio de las características principales de los transductores de presión. Fuente: elaboración propia.

	M5200 Series	PX3 Series	P51 Series
Fabricante	TE Connectivity	Honeywell	Amphenol
Alimentación	8-30 V	5V/3V (salida radiométrica) o 8-30 V (salida corriente)	5V (salida radiométrica) o 8-30 V (salida tensión/corriente)
Consumo corriente	10 mA	< 3 mA/2.1mA	< 6mA /3mA
Salida	1-5V (salida radiométrica) /0-10V/0.5-4.5V (salida tensión) /4-20mA (salida corriente)	0.5-4.5V/0.33-2.97V (salida radiométrica) o 4-20mA (salida corriente)	0.5-4.5V (salida radiométrica) o 1-5V (salida tensión) 4-20mA (salida corriente)

Figura 10. Ejemplo de transductores, M5200, PX3 y P51 de izquierda a derecha. Fuente: Catálogo: M5200 Series, PX3 Series y P51 Series.

Como se puede ver, existen 3 tipos de sensores:

- **Radiométricos:** su salida es proporcional a la alimentación, se alimentan a 3 o 5 Vdc constantes.
- **Salida Tensión:** admiten un rango de alimentación de 8–30 Vdc y su salida está normalizada a 1-5V o 0-10V.
- **Salida Corriente:** admiten un rango de alimentación 8-30 Vdc y su salida está normalizada a 4-20 mA

Existen multitud de alimentaciones disponibles, como se requiere de un sistema totalmente adaptable a cualquier sensor, todas ellas deben ser evaluadas.

- **Alimentación a 3V (radiométrico):** podría ser alimentado a través de la misma alimentación que la PCB.
- **Alimentación a 5V (radiométrico):** se debería disponer de un regulador de salida 5 V.

- **Alimentación 8-30 V (industrial):** se debería disponer de un regulador de mayor tensión (por ejemplo 12 V) o se podría alimentar directamente de una batería cuyo rango de tensiones se encuentre dentro del rango 8-30 V, como por ejemplo una batería de polímero de Litio de 4S cuya tensión se encuentra alrededor de 14.8 V o una batería de gel cuyo rango está en torno a 12V.

Como se ha deducido anteriormente, se requiere de una tensión de 3V para la alimentación de los componentes de la PCB (microcontrolador, módulo SigFox, Bluetooth etc.) y una tensión de 3V, 5V o 8-30V para la alimentación del sensor.

Para la selección del sistema de alimentación se proponen dos alternativas para su evaluación:

- **Opción 1:** batería única de rango de tensión de 8-30 V, con un regulador de salida 3V para la PCB y otro de 5V para la alimentación del sensor, en caso de que sea necesario.
- **Opción 2:** 2 sistemas de baterías independientes, una para la alimentación de la PCB 3V y otro para el sensor 8-30V o 5V.

Para la selección de una de las alternativas propuestas es necesario en primero lugar, profundizar sobre los distintos tipos de conversores DC/DC y de sus especificaciones. (7) (8)

Regulador lineal: basan su funcionamiento en un elemento activo (transistor bipolar), o uno pasivo (diodo Zener). Su funcionamiento simula una resistencia variable, que ajusta su valor para mantener constante la tensión de salida. Su eficiencia es mayor cuanto menor es la caída de tensión, y solo pueden operar a modo de reductores.

Regulador de conmutación: basan su funcionamiento en la conmutación de uno o más interruptores (transistores) para transformar un nivel de tensión en otro. Son altamente eficientes cuando trabajan a su carga nominal y son capaces de reducir, aumentar e invertir las tensiones.

Tabla 8. Comparativa entre regulador lineal y convertidor conmutado. Fuente: elaboración propia.

	Regulador lineal	Convertidor conmutado
Función	Solo reductor	Aumento, reducción o inversión
Eficiencia	Baja a media. La eficiencia es alta para diferencias de tensión reducidas.	Alta para corrientes próximas a la corriente nominal, y baja para corrientes reducidas
Disipación de calor	Alta, si la diferencia de tensión es alta	Baja a media
Complejidad	Baja	Media a alta
Coste	Bajo	Medio a alto, requiere de mayor número de componentes
Ruido	Baja, bajo nivel de ruido	Media a alta, debido a la conmutación

A priori, parece ser que el convertidor conmutado presenta mejores características, pero para bajos consumos, como cuando el microcontrolador entra en modo Sleep, apagando todos los periféricos, la corriente en reposo (≈5-10 mA) es mucho mayor al consumo (≤ 1µA).

Ya que para alcanzar una vida útil de batería de unos 2 años se requiere reducir la media del consumo por debajo del miliamperio, el empleo de convertidores conmutados queda totalmente descartada.

Al emplear, reguladores lineales, se requiere que la tensión de salida sea lo más próxima a la tensión de entrada, para así mejorar su eficiencia. Por este motivo se ha optado por la **opción 2,** que dispone de dos baterías independientes, una adecuada a la tensión de funcionamiento del sistema 3V y otra para la alimentación del sensor 8-30 V o 5V.

El regulador seleccionado es el NCP551SN30T1G, cuyas características son las siguientes:

Tabla 9. Características del regulador NCP551SN30T1G. Fuente: elaboración propia.

	NCP551SN30T1G
Tipo	Regulador lineal
Corriente de reposo	4.0 µA
Máxima tensión de entrada	12 V
Tensión de salida	3.0 V
Máxima corriente de salida	150 mA
Precisión	< 2%

Por otra parte, el sistema de alimentación incluye la conexión y desconexión de alimentación de ciertos periféricos como es el caso del sensor. Para ello, se emplea el uso de un MOSFET dual con la siguiente configuración.

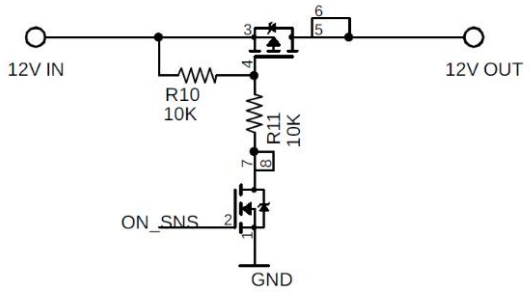


Figura 11. Esquema de conexión y desconexión de la alimentación controlado por software. Fuente: elaboración propia.

5.5. Subsistema controlador: microcontrolador

Para la selección del microcontrolador en primer lugar se han descrito los periféricos mínimos que debe incluir para el control de todos los subsistemas anteriormente descritos:

- ATA8520E:
  1x SPI: (MISO, MOSI, CLK, CS)
  1x Input: PB6
  3x Output: PB4, PC0 y PC1
- ADC
  1x Convertidor Sigma-Delta 16 bits
  1x Convertidor PipeLine o SAR
- RN42
  1x UART: (Tx y Rx)
  1x Input: COMM
- SSH1106
  1x I2C: (SDA, SCL)

	PIC24FJ128GC006	**STM32L412**
CPU	16 bit	32 bit
Velocidad	Hasta 16 MIPS, CPU @ 32 MHz	Hasta 100 DMIPS, CPU @ 80 MHz
Memoria FLASH	128 KB	128 KB
Memoria RAM	8 KB	40 KB
Deep sleep	75nA + 350 nA RTCC	340 nA con RTCC
ADCs	2x 16 bit Sigma-Delta ADC	2x Sigma-Delta ADC
UARTs	4x UART	4x UART
I2s	2x I2C	2x I2C
SPI	2x SPI	2x SPI
USB	Sí	Sí
Precio	3,63 €	4,38 €

La selección entre ambos dispositivos, atendiendo exclusivamente a las especificaciones se vuelve compleja, ya que ambos presentan características muy similares. También, presentan un precio similar. Finalmente se ha optado por el PIC24FJ128GC006, debido a la reputación del fabricante MicroChip.

6. DISEÑO DEL PROTOTIPO INICIAL

Una vez seleccionado los elementos de mayor relevancia que formarán el hardware del sistema propuesto, el siguiente paso es, la realización de la interconexión entre los distintos subsistemas y la realización de la circuitería necesaria para el correcto funcionamiento de cada uno de ellos.

6.1. Subsistema de comunicación: módulo SigFox

El módulo seleccionado es el ATA8520E, en este caso el módulo debe conectarse al microcontrolador PIC24FJ128GC006 mediante una comunicación digital SPI y por otra parte se debe realizar la conexión de radiofrecuencia hasta el terminal para la antena.

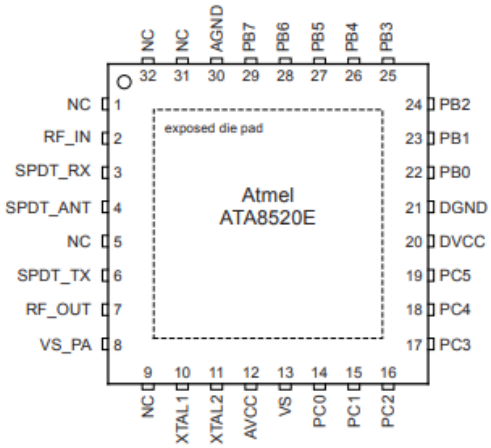


Figura 12. Pinout del integrado ATA8520E. Fuente ATA8520E Datasheet.

En cuanto a la comunicación digital requiere la conexión de los siguientes pines:

- PC0: NRESET (low active): para realizar un reset al dispositivo.
- PC1: NPWRON (low active): para despertar el módulo de su estado de bajo consumo.
- PB1: SCK (SPI): línea de reloj de la comunicación SPI.
- PB2: MOSI (SPI): línea de transmisión de datos desde el maestro.
- PB3: MISO (SPI): línea de recepción de datos del maestro.
- PB4: PWRON: `para despertar el módulo de su estado de bajo consumo.
- PB5: NSS (low active): línea de selección de dispositivo de la comunicación SPI.
- PB6: Event (low active): para comunicación de eventos (Ej: finalización de un envío).

Por otra parte, el circuito de radiofrecuencia es más complejo. El módulo SigFox funciona con una comunicación Half-duplex, esto quiere decir que el emisor de radiofrecuencia, emplea la misma antena que el receptor de radiofrecuencia. Para eso, se requiere un conmutador de antena que conecte el pin emisor con la antena cuando está enviando y cambie al pin receptor cuando está recibiendo.

Para el control del conmutador de antena el integrado proporciona una serie de pines:

- PB0: activo cuando se requiere el empleo del Front-End (envío o recepción).
- PB7: activo cuando el módulo está enviando, y bajo cuando está recibiendo.

Tabla 10. Control del FEM externo desde el propio integrado ATA8520E. Fuente: elaboración propia.

	PB0	PB7
FEM desactivado	0	X
Uplink activo	1	1
Downlink activo	1	0

El integrado posee también un conmutador interno que se puede emplear tanto como FEM interno como para el control de un FEM externo que la lógica no coincida con PB7. Es el caso del SPDT, durante un envío conecta SPDT_ANT con SPDT_TX y durante una recepción conecta SPDT_ANT con SPDT_RX.

No obstante, el fabricante recomienda el uso de un FEM externa principalmente por dos motivos:

- Para conseguir la sensibilidad de -126 dBm que establece SigFox. La sensibilidad del receptor del ATA8520E es de -121dBm. Además, en caso de utilizar un filtro SAW en la línea de recepción, se deben compensar las pérdidas que suponen (3 dBm). Esto se consigue mediante un FEM con un LNA (Low-Noise Amplifier) incluido de 10 dBm
- Se recomienda el empleo de filtros en la línea de recepción y en la de envío.
    - En la de recepción recomienda el empleo de un filtro SAW para bloquear las señales de RF fuera de banda. El rango de frecuencia requerido es de 869.525 MHz ± 96 kHz
    - En el envío se recomienda el empleo de un filtro LC para suprimir las emisiones espurias. Como por ejemplo el segundo y tercer armónico de la frecuencia de 868.13 MHz

Por lo tanto, la conexión global se resume con el siguiente esquema:

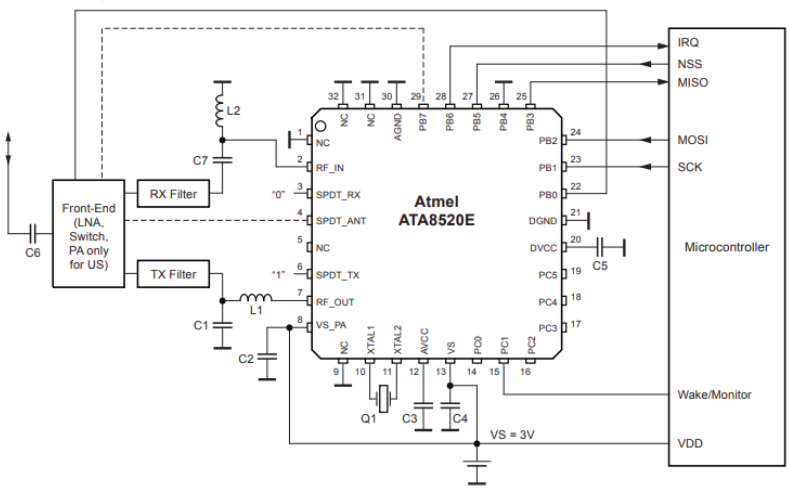


Figura 13. Esquema para módulo ATA8520E modo uplink y downlink. Fuente: ATA8520E Datasheet.

6.1.1. Implementación del filtro TX

Como se ha comentado el filtro LC se utilizada para suprimir los armónicos de orden superior a la frecuencia de trabajo. El filtro LC implementado es un filtro de paso bajo Tschebyscheff con una frecuencia de corte de 1 GHz.

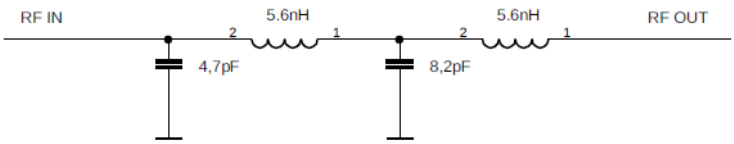


Figura 14. Filtro LC en línea de envío. Fuente: ATA8520E Datasheet.

6.1.2.Selección del filtro SAW

El filtro SAW se emplea para bloquear las señales RF fuera de banda (869.525 MHz ± 96 kHz). Los filtros SAW o de onda acústica superficial se basan en el uso de cristales piezoeléctricos o de cerámica, convierten las señales eléctricas en mecánicas. Los requisitos son:

- Filtro paso banda: 869.525 MHz ± 96 kHz
- Pérdidas por inserción 3 dB aprox.
- Impedancia de carga de 50 Ω

Finalmente, el filtro SWA seleccionado ha sido B39871B3440U410, cuyas características son:

- Centro de frecuencia 869 MHz
- Ancho de banda 2MHz
- Pérdidas por inserción máximas de 3.1 dB (típico 1.2 dB)
- Máximo rizado p-p de 1.2 dB (típico 0.6 dB)
- Impedancia de entrada y salida de 50 Ω

6.1.3.Selección del módulo FEM

El módulo FEM requiere de un LNA con una ganancia de 10 dB para aumentar la sensibilidad de recepción desde los -121 dBm hasta los -126 dBm, teniendo en cuanta unas pérdidas aproximadas de 3dB en el filtro SAW. Además, debe permitir su encendido y apagado controlado por PB0 y la conexión a la salida con la antena controlada por PB7.

El módulo seleccionado es el SKY65378-11

- LNA integrado, con ganancia programable
- Bajo ruido < 2.5dB
- Rápido encendido/apagado < 1 µs
- Alimentación 2.0-4.8V
- Modo sleep < 1 µA

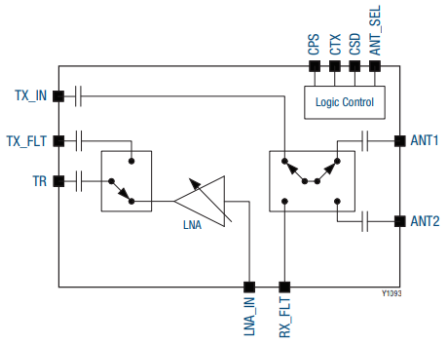


Figura 15. Circuito interno del SKY65378-11. Fuente: SKY65378-11 Datasheet.

El módulo SKY65378-11 tiene las siguientes entradas de control:

- CSD: control de apagado: permite introducir el módulo en modo Sleep:
  - 0 = Modo Sleep.
  - 1 = Activo.
- CPS: Seleccionador de recepción:
  - 0 = Conecta LNA_IN con TX_FLT
  - 1 = Conecta LNA_IN con TR
- CTX: Selecciona la entrada a la salida de antena:
  - 0 = Conecta con RX_FLT (recibir).
  - 1 = Conecta con TX_IN (transmitir).
- ANT_SEL: Selecciona la antena de salida:
  - 0 = Conecta ANT1.
  - 1 = Conecta ANT2.

Para el dispositivo en cuestión:

- Se requiere una antena (ANT1), por lo que ANT_SEL se conecta directamente a GND.
- El encendido y apagado se maneja con el pin PB0 conectado directamente a CSD.
- La recepción se realiza exclusivamente por un pin (TR). No obstante, para entrar en modo Sleep se requiere que CPS=0 por lo que se conecta también a PB0.
- La selección de envío o recepción se realiza con PB7 conectado directamente a CTX.
- ANT2 se conecta a una línea de 50 Ω para evitar daños internos (recomendación del fabricante).

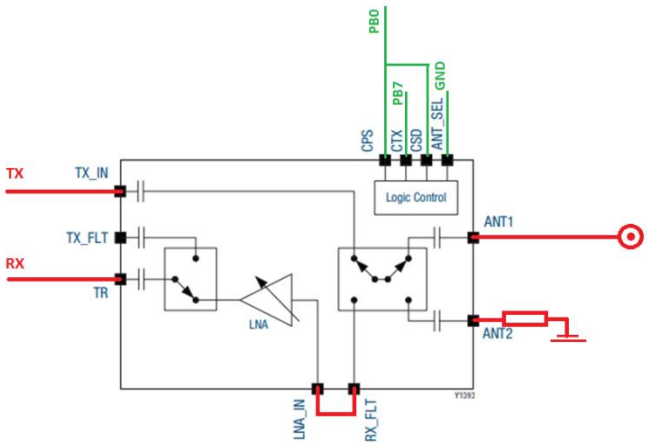


Figura 16. Conexión del integrado SKY65378-11. Fuente: elaboración propia.

6.1.4. Implementación completa ATA8520E

A los esquemas anteriores se les ha añadido los componentes LC recomendados por el fabricante y un MOSFET dual para desconectar la alimentación de la parte de radiofrecuencia, ya que ésta tiene consumos de 15 µA debido a resistencias internas de pull-up.

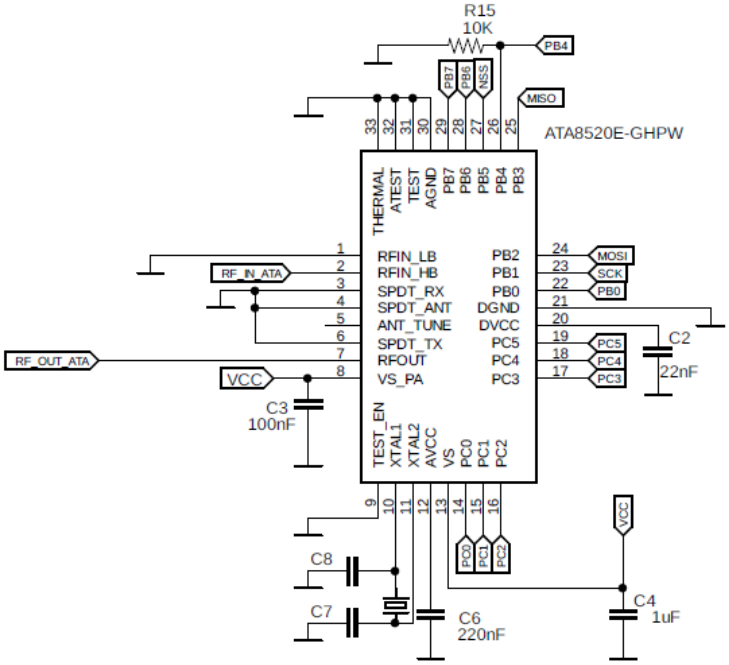


Figura 17. Conexiones módulo SigFox ATA8520E. Fuente: elaboración propia.

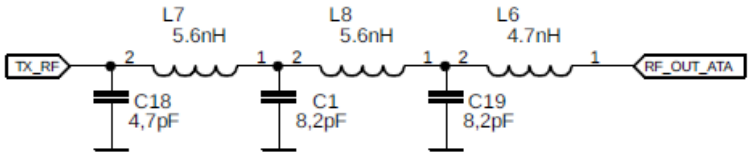


Figura 18. Filtro LC en línea de transmisión. Fuente: elaboración propia.

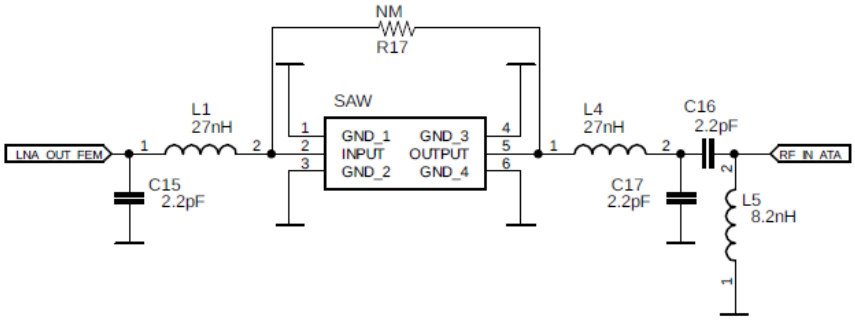


Figura 19. Filtro SAW en la línea de recepción. Fuente: elaboración propia.

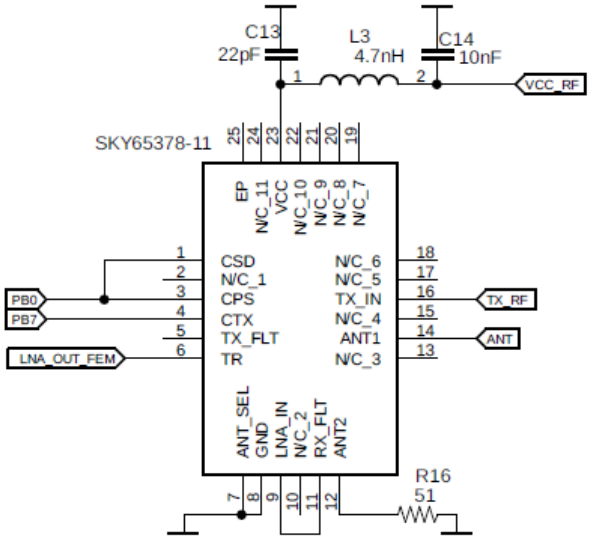


Figura 20. Módulo FEM SKY65378. Fuente: elaboración propia.

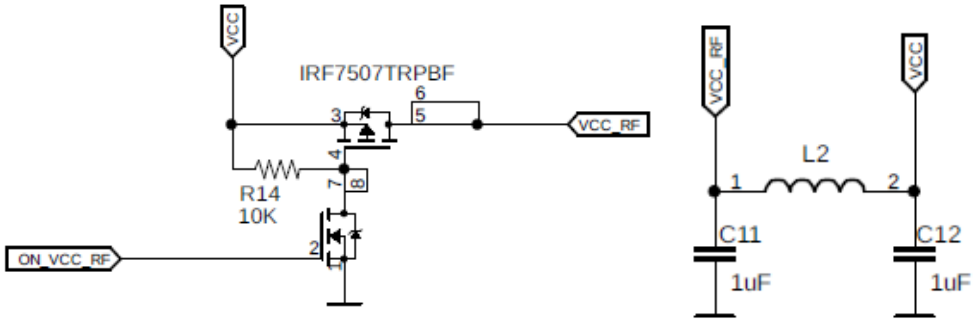


Figura 21. Circuito de alimentación módulo ATA8520E y RF. Fuente: elaboración propia.

6.2. Subsistema de adquisición:

El subsistema de adquisición tiene que adaptar las señales de salida del sensor (0-10V o 4-20mA) a la entrada de la ADC. Se ha implementado el circuito explicado anteriormente seleccionando componentes con una exactitud de 0.1%.

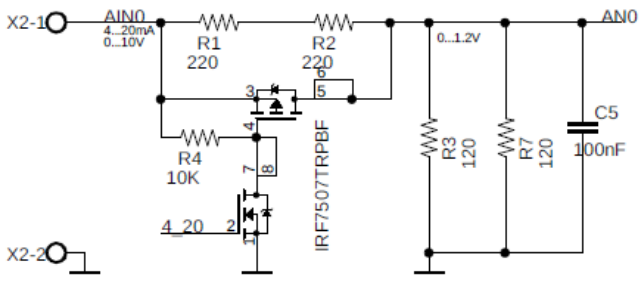


Figura 22. Circuito de adaptación de tensiones para entradas de 0-10 V o 4-20 mA. Fuente: elaboración propia.

Se ha añadido un condensador de 100 nF a la entrada de la ADC para reducir el ruido de altas frecuencias.

Por otra parte, la lectura del nivel de batería se realiza mediante un divisor de tensiones. El extremo en lugar de conectarse directamente a GND se conecta a un GPIO (GN) para poder así evitar el consumo continuo del divisor.

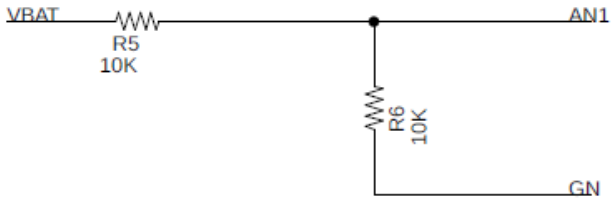


Figura 23. Divisor de tensiones para la lectura del nivel de batería. Fuente: elaboración propia.

6.3. Subsistema de interfaz con el operador: display numérico y módulo Bluetooth

6.3.1. Display numérico: SSH1106

El display seleccionado no requiere ningún circuito extra para su implementación, únicamente es necesario conectar la alimentación (3,3 V y GND) y los 2 pines de la comunicación I2C (SDA y SCL). Para ello se ha puesto en la PCB diseñada un conector de 4 pines

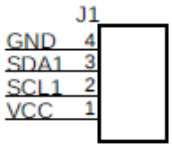


Figura 24. Conector 4 pines para el display SSH1106. Fuente: elaboración propia.

6.3.2. Módulo Bluetooth: RN-42

Para el módulo bluetooth RN-42 se requiere de los siguientes pines:

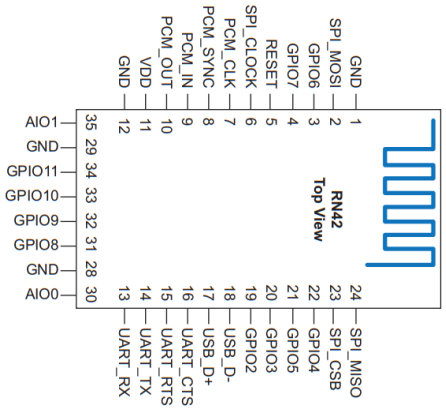


Figura 25. Pinout módulo bluetooth RN-42. Fuente: Datasheet RN-42.

- UART_TX: línea de transmisión UART.
- UART_RX: línea de recepción UART.
- UART_CTS: línea *Clear to Send* para *Flow Control* por hardware.
- UART_RTS: línea *Request to Send* para *Flow Control* por hardware.
- GPIO2: permite detectar si se encuentra vinculado a otro dispositivo.

- GPIO5: salida de estado del Bluetooth.
    - Parpadeo 1Hz: esperando conexión
    - Parpadeo 10 Hz: modo configuración
    - Fijo: dispositivo vinculado

Los pines de Tx y Rx se han conectado a los pines de comunicación del micro Rx y Tx. Los de Flow control (CTS y RTS) se han conectado entre sí por no requerir realizar el control desde el microcontrolador. Esto es debido a la baja tasa de transferencia requerida, únicamente se emplea para la configuración del dispositivo. La salida del GPIO5 se ha conectado a un LED para facilitar la programación del dispositivo y poder observar cuando el bluetooth está encendido, conectado o configurándose. La salida del GPIO2 se ha conectado a una entrada digital del microcontrolador para poder conocer cuándo se establece la conexión. También se ha añadido un MOSFET dual para desconectar/conectar la alimentación del bluetooth con el objetivo de reducir el consumo de la batería.

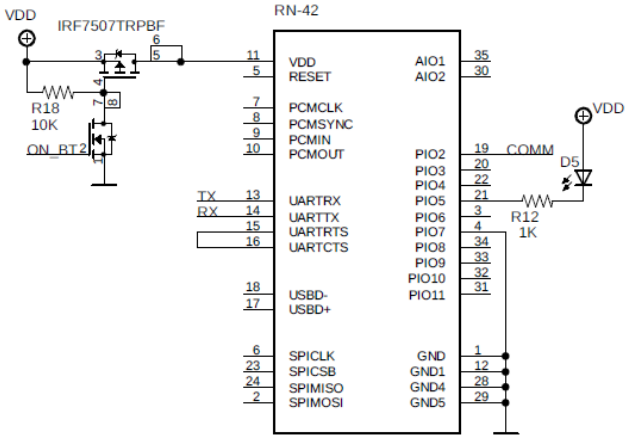


Figura 26. Conexión módulo bluetooth RN-42. Fuente: elaboración propia.

6.4. Subsistema de alimentación: cargador de batería Litio

El cargador de baterías seleccionado MCP73833 requiere la siguiente conexión.

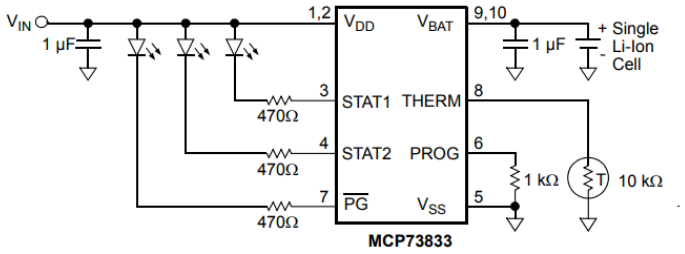


Figura 27. Circuito de requerido para la carga de batería de litio de una celda. Fuente: Datasheet MCP73833.

- V_{DD}: Tensión de carga desde V_{reg} + 0.3 V hasta 6V
- $STAT_1$ y $STAT_2$: indicadores de carga de la batería
- PG: Indica que la tensión de entrada está por encima del umbral máximo de carga
- PROG: Permite ajustar la corriente de carga $I_{reg}(mA) = \frac{1000V}{R_{prog}(k\Omega)}$
- THERM: Permite la suspensión de la carga si se supera la temperatura de +150°C. Se debe conectar a la NTC propia de la batería. En caso de no disponer NTC la batería se debe añadir una resistencia de 10 kΩ

Para el prototipo se ha implementado para una carga de 750 mA

$$R_{prog}(k\Omega) = \frac{1000\ V}{I_{reg}(mA)} = \frac{1000\ V}{750\ mA} = 1.33\ k\Omega \qquad \text{Ec.2}$$

Se ha añadido un conector USB, a través el cual se realiza la carga de la batería, la conexión USB y la programación del microcontrolador.

La conexión USB requiere de los siguientes pines:

- VCC: 5V
- D-: Data –
- D+: Data +
- GND

El microcontrolador, se programa a través del puerto ICSP (In-Circuit Serial Programming™) el cual requiere los siguientes pines:

- PGC (Program Clock): señal de reloj provista al microcontrolador para su sincronía
- PGD (Program Data): Línea serial de datos para escritura/lectura/verificación de la memoria de programa del microcontrolador.
- RESET: reinicio del microcontrolador
- GND

A la entrada de la batería se ha incorporado el regulador NCP551SN30 que transforma la tensión de la batería a 3 V para la alimentación completa de la PCB.

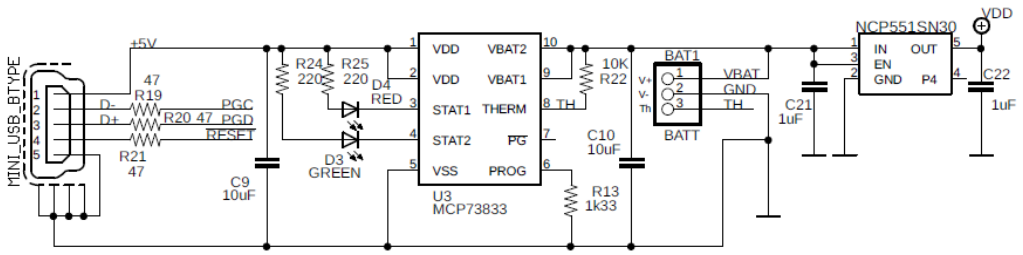


Figura 28. Circuito de alimentación y carga del sistema. Fuente: elaboración propia.

6.5. Circuitos auxiliares

Se han añadido dos LEDs para facilitar la programación/depuración y tres interruptores REED para el control del usuario:

- REED 1: reinicia el dispositivo y entra en modo configuración, activando el Bluetooth.
- REED 2: comprueba la medición del sensor, enciende el display y el sensor, y muestra las lecturas.
- REED 3: realiza una configuración automática, una vez en modo configuración mediante este interruptor se puede salir del modo e iniciar una recepción automática de la configuración a través de un downlink de SigFox.

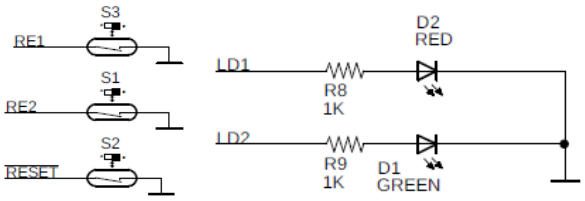


Figura 29. Interruptores REED izquierda, LEDs auxiliares derecha. Fuente: elaboración propia.

6.6. Subsistema controlador: microcontrolador

En primer lugar, al microcontrolador seleccionado PIC24FJ128GC006 se conectan los pines de alimentación y masa con sus correspondientes condensadores de desacoplo.

6.6.1. Osciladores principal y secundario

El siguiente paso es la conexión de los osciladores. El microcontrolador seleccionado requiere de dos osciladores:

- Oscilador principal (OSCI y OSCO): es el encargado de suministrar el reloj interno a la CPU. Se ha seleccionado ABM8G-11.0592-4Y-T3 de 11,059 MHz
- Oscilador secundario (SOSCO y SOSCI): es el encargado de suministrar el reloj al RTCC (Real time clock and calendar). Se ha seleccionado el SG-3030CM de 32,768 KHz.

En el modo de bajo consumo Deep Sleep, el reloj interno de la CPU es apagado, mientras que el reloj del RTCC y el Watchdog se mantienen encendidos. Esto permite tener la CPU paralizada, reduciendo así el consumo, manteniendo los módulos RTCC y Watchdog encendidos. De este modo, se puede despertar el micro de forma periódica mediante una interrupción del RTCC o del Watchdog. El oscilador del RTCC es mucho más lento que el reloj de la CPU, lo que supone una drástica reducción del consumo.

6.6.2. Convertidor Analógico-Digital Sigma-Delta

Como se ha comentado anteriormente, se emplea el convertidor Analógico-Digital Sigma Delta interno del microcontrolador.

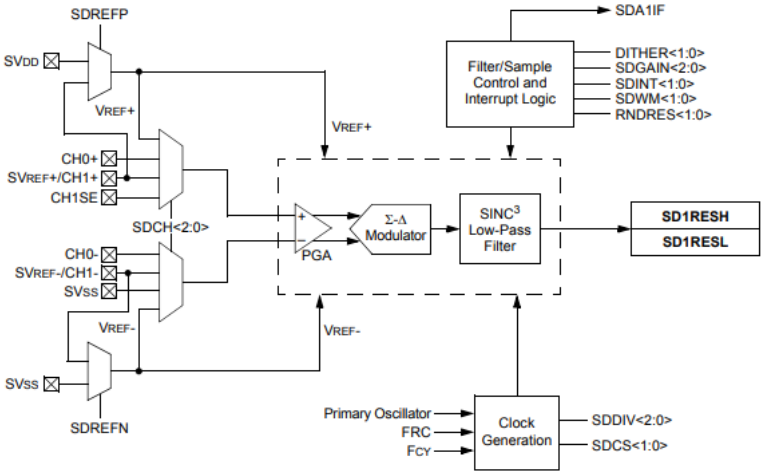


Figura 30. Esquema interno Convertidor Analógico-Digital Sigma-Delta. Fuente: Datasheet PIC24FJ128GC006.

Se trata de un convertidor diferencial (pudiéndose convertir en unipolar seleccionando SVss en el canal negativo). Como referencias de tensión se puede emplear o bien la interna (la tensión de alimentación) o una referencia externa (introducida por el canal 1). El empleo de la alimentación como referencia no es recomendable, ya que presenta una gran cantidad de ruido. El PIC24FJ128GC006 dispone de referencias de tensión internas de alta precisión (BGBUF0, BGBUF1 y BGBUF2) y dos de ellas pueden ser externalizadas (BGBUF1 y BGBUF2) por lo que se ha optado por utilizar una de ellas.

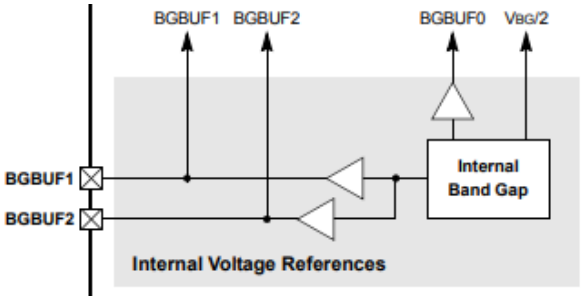


Figura 31. Esquema de referencias internas de tensión del PIC24. Fuente: Datasheet PIC24FJ128GC006.

Por lo tanto, es necesario extraer la referencia interna a un pin del microcontrolador (BGBUF1 o BGBUF2) e introducirla al pin de referencia del convertidor analógico-digital para utilizarla. Finalmente se ha optado por emplear el canal 0, como la entrada analógica del sensor y el canal 1 como la entrada de la referencia, externalizada por el pin BGBUF2.

- PIN nº4: Salida de tensión de referencia BGBUF2
- PIN nº18: Entrada CH0+: conectada a la salida del circuito adaptador de tensión del sensor
- PIN nº23: Entrada CH0-: conectada a GNS
- PIN nº24: Entrada CH1+/SVref+: conectada a la salida de la tensión de referencia BGBUF2
- PIN nº25: Entrada CH1-/SVref-: conectada a GND

6.6.3. Comunicación con el resto de los periféricos

Se ha requerido la conexión de los pines de comunicación, entradas y salidas digitales descritas anteriormente para el control de cada uno de los módulos. Para la selección se ha tenido en cuenta que las interfaces I2C y USB no son configurables, es decir tienen unos pines fijos. Mientras, que las interfaces UART y SPI si lo son.

- PIN nº1: GPIO RE5 (RE1 entrada del interruptor REED 1)
- PIN nº7: MCLR (Pin de reset, conectado al REED 0 y al conector de programación)
- PIN nº13: GPIO RB3 (RE2 entrada del interruptor REED 2)
- PIN nº14: GPIO RB2 (ON_SNS: salida de activación de la alimentación del sensor)
- PIN nº15: PGEC1 (Reloj de programación ICSP)
- PIN nº16: PGED1 (Línea de datos programación ICSP)
- PIN nº33: GPIO RF3 (ON_BT: salida de activación del módulo Bluetooth)
- PIN nº34: V_{BUS} Tensión requerida de +5V para detección de USB
- PIN nº36: D- comunicación USB
- PIN nº37: D+ comunicación USB
- PIN nº42: GPIO RD2 (COMM: entrada de detección de dispositivo Bluetooth vinculado)
- PIN nº43: SDA1 Comunicación I2C, puerto nº1
- PIN nº44: SCL1 Comunicación I2C, puerto nº1
- PIN nº45: RP12 Pin configurable a Rx1 Comunicación UART, puerto nº1

- PIN nº46: RP11 Pin configurable a Tx1 Comunicación UART, puerto nº1
- PIN nº47: GPIO RC13 (ON_VCC_RF: salida de componentes de radiofrecuencia)
- PIN nº49: NSS: RP24 Pin configurable a NSS1 Comunicación SPI, puerto nº1
- PIN nº50: GPIO RD2 (PB4: salida PWRON para despertar el módulo ATA8520E)
- PIN nº51: RP22 Pin configurable a MISO1 Comunicación SPI, puerto nº1
- PIN nº52: RP25 Pin configurable a MOSI1 Comunicación SPI, puerto nº1
- PIN nº53: RP20 Pin configurable a SCK1 Comunicación SPI, puerto nº1
- PIN nº54: GPIO RD6 (PB6: entrada detección de eventos del ATA8520E)
- PIN nº61: GPIO RE1 (PC1: salida NPWRON para despertar el módulo ATA8520E)
- PIN nº62: GPIO RE2 (PC0: salida para reset del módulo ATA8520E)
- PIN nº63: GPIO RE3 (LED1: salida encendido LED 1)
- PIN nº64: GPIO RE4 (LED2: salida encendido LED 2)

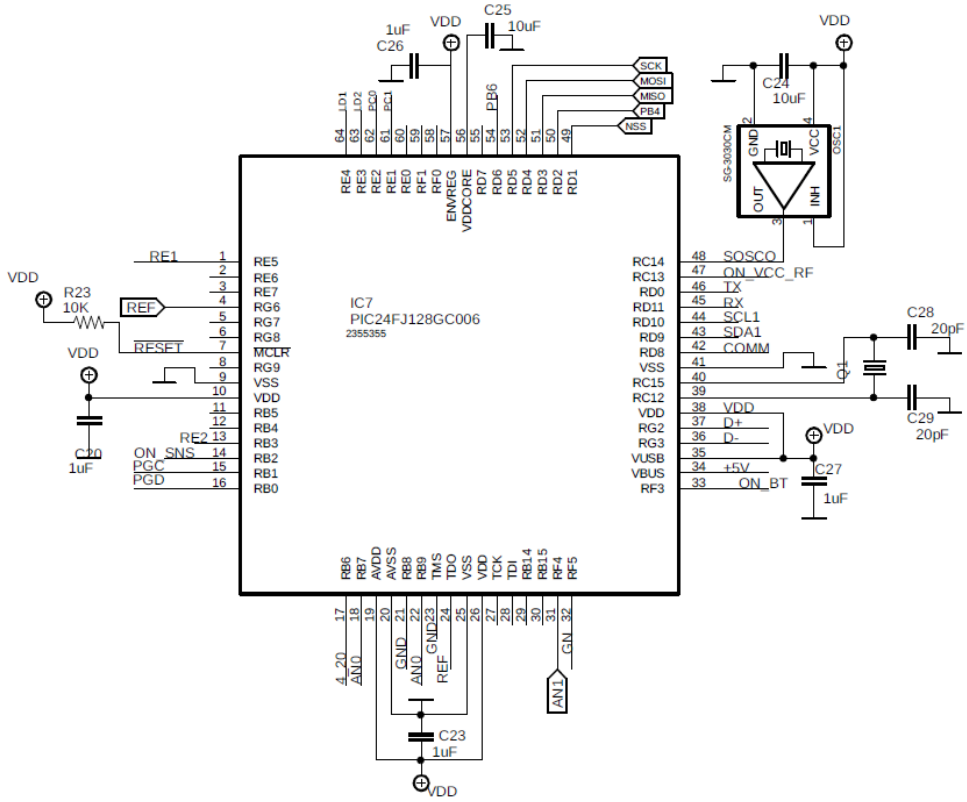


Figura 32. Esquema de conexión del microcontrolador PIC24FJ128GC006. Fuente: elaboración propia.

7. DISEÑO DEL CIRCUITO IMPRESO

Una vez realizado el esquemático global del sistema, el siguiente paso ha sido su diseño en una placa impresa. Para la realización del diseño se ha empleado el programa Eagle del fabricante Autodesk.

7.1. Nomenclatura y definiciones

- **Esquemático:** diagrama o esquema que representa las interconexiones entre los distintos elementos.
- **Boardfile:** archivo que recoge el diseño físico de la placa impresa, el diseño se conoce como Layout.
- **Pad:** zona de contacto entre un terminal de un componente y el pin de la PCB
    - **Pad Through-hole:** orificios circulares, que constan de una zona de soldadora y atraviesan la placa.
    - **Pad SMD:** áreas rectangulares de cobre sobre una de las capas exteriores (TOP o BOTTOM).
- **Huella – Footprint:** se trata del diseño de los pads para cada componente. Existen huellas estandarizadas (DIP, SOIC, TSSOP, con el número de pines, DIP16, SOIC8…) y huellas específicas para componentes personalizados. Cada fabricante pone a la disposición la huella del componente en su hoja de datos, es habitual que existan diferentes integrados y por lo tanto distintas huellas para un mismo componente.
- **Soldermask:** máscara de soldadura, se emplea para automatizar el montaje de la PCB, resaltando los pads sobre los que se debe extender la pasta de soldadura.
- **Silkscreen:** máscara de serigrafía, se emplea para realizar anotaciones y/o identificación de componentes y terminales.
- **Net:** segmento de línea que une dos pads de uno o dos componentes.
    - **Net en esquemático:** se trata de una unión realizada en el esquemático.
    - **Rat:** net en el Boardfile que aún no ha sido trazada en cobre.
    - **Traza:** una net en el Boardfile que ya se ha trazado en cobre.
- **Vía:** se emplea para pasar una traza o net entre distintas capas de la PCB. Requiere la definición de su diámetro de pad y de perforación.
- **Layer o capa:** corresponden a procesos de fabricación:
    - **Capas de cobre**
    - **Capas de serigrafía**
    - **Capas de taladros**
    - **Capas de información**
    - **Capas de tamaño de PCB**
- **Gerber:** se trata de un formato de fichero estándar en la industria de fabricación de PCB. La información va repartida en distintos ficheros (fichero de cobre en la cara TOP, en la BOTTOM, soldermask de la cara TOP, de la BOTTOM…).

7.2. Normas y recomendaciones en el Layout

Normas generales

- Es preferible definir el tamaño de la PCB antes de empezar a colocar los componentes, y así adaptar su colocación al espacio disponible.
- Colocar en primer lugar los componentes de entrada y salida. Estos deben ser colocados atendiendo a su conexión en exterior a la placa.
- Es preferible agrupar los componentes según bloques lógicos. De esta forma se minimizan las trazas de conexión entre los distintos bloques.

Normas sobre el trazado (9)

- Siempre se deben realizar las pistas lo más cortas posibles para así minimizar los efectos parásitos resistivos y capacitivos.
- Dimensionar el ancho de pista para permitir el paso de la corriente máxima:
  - 4 mm : 10 A
  - 2 mm : 5 A
  - 1.5 mm : 4 A
  - 1 mm : 3 A
  - 0.5 mm : 2 A
  - 0.2 mm : 0.5 A
- Las señales digitales suelen ser inferiores a 500 mA
- La separación mínima se dimensiona según la tensión que soporten.
  - V < 10 V : 0.3 mm
  - 10 V < V < 50 V : 0.5 mm
  - 50 V < V < 100 V : 0.7 mm
  - 100 V < V < 170 V : 1.0 mm
  - 170 V < V < 250 V : 1.2 mm
  - 250 V < V < 500 V : 3.0 mm
- Se debe evitar el trazado de ángulos de 90 o menores

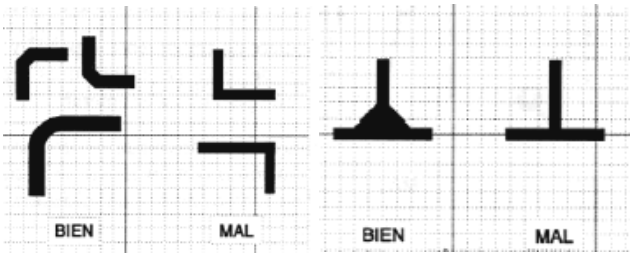


Figura 33. Ejemplo de trazados correctos. Fuente: (9).

- Mantener una separación entre las pistas y los bordes de al menos 5 mm
- Evitar el paso de pistas entre terminales de componentes activos (transistores…)
- Se debe realizar perforaciones sobre la placa para su posterior colocación
- Pistas paralelas deben mantener la misma distancia de separación en toda su trayectoria
- Es recomendable el empleo de planos de tierra para mejorar el desempeño de la misma y reducir los bucles de corrientes, así como las emisiones electromagnéticas

7.3. Teoría de circuitos impresos para radiofrecuencia

Impedancia de entrada (10)

La impedancia de entrada de una red eléctrica es la medida de la oposición a la corriente (impedancia), tanto estática (resistencia) como dinámica (reactancia), en la red de carga que es externa a la fuente eléctrica. La impedancia de entrada se obtiene a partir del circuito equivalente de Thévenin de una red eléctrica.

Impedancia de entrada en sistemas de alta frecuencia (RF)

Cuando la impedancia característica de una línea de transmisión, $Z_{Línea}$ no coincide con la impedancia de la red de carga, $Z_{Entrada}$, la red de carga reflejará parte de la señal fuente. Esto puede crear ondas estacionarias en la línea de transmisión. Para minimizar los reflejos, la impedancia característica de la línea de transmisión y la impedancia del circuito de carga deben ser iguales o "coincidentes" para reducir reflexiones de señal.

Las reflexiones pueden generar distorsiones, atenuaciones y daños en la circuitería. Cuando las impedancias son coincidentes se denomina conexión emparejada y el proceso de emparejamiento como adaptación de impedancias.

Los valores más habituales de impedancias son 50Ω y de 75Ω.

$$Z_{carga} = Z_{línea} = Z_{fuente} \qquad \text{Ec.3}$$

Por lo tanto, en una línea de transmisión de radiofrecuencia se deben seleccionar los componentes con las impedancias de entrada y salidas coincidentes, así como diseñar la pista para su impedancia controlada.

Cálculo de impedancia de línea

Es la relación que existe entre la diferencia de potencial que se aplica y la corriente absorbida en una línea de transmisión. La impedancia de una línea depende de sus parámetros primarios: (resistencia, inductancia y conductancia)

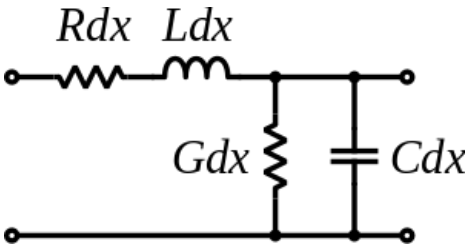


Figura 34. Esquema circuito equivalente a la impedancia de línea. Fuente: elaboración propia.

Según el modelo propuesto se puede obtener la impedancia de línea mediante la siguiente expresión:

$$Z_0 = \sqrt{\frac{R + j \cdot L\omega}{G + j \cdot C\omega}} \qquad \text{Ec.4}$$

Cálculo de impedancia de línea en PCB

Cable coaxial

La impedancia característica del cable Z_0 está determinado por la constante dieléctrica del aislante interno y los radios de los conductores interno y externo. En los sistemas de radiofrecuencia, donde la longitud del cable es comparable a la longitud de onda de las señales transmitidas, una impedancia característica del cable uniforme es importante para minimizar las pérdidas. Las impedancias de fuente y carga se eligen para que coincidan con la impedancia del cable para garantizar una transferencia de potencia máxima y una relación de onda estacionaria mínima.

$$Z_0 = 138 \cdot \log\left(\frac{D}{d}\right)\left(\frac{1}{\sqrt{k}}\right) \qquad \text{Ec.5}$$

Stripline

Stripline utiliza una banda plana de metal que se intercala entre dos planos de tierra paralelos. El material aislante del sustrato forma un dieléctrico. El ancho de la tira, el grosor del sustrato y la permitividad relativa del sustrato determinan la impedancia característica de la línea.

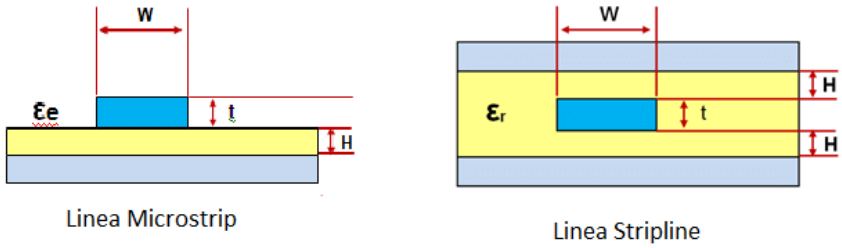


Figura 35. Esquema de construcción de una línea Microstrip y una Sripline. Fuente: (11).

Para evitar la propagación de modos no deseados, los dos planos de tierra deben estar en corto entre sí. Esto se logra comúnmente mediante una fila de vías paralelas a la tira en cada lado. Al igual que el cable coaxial, la línea de banda no es dispersiva y no tiene frecuencia de corte.

$$\beta = \omega\sqrt{\mu_0 \cdot \varepsilon_0 \cdot \varepsilon} \qquad \text{Ec.6}$$

$$Z_0 = \frac{30\pi}{\sqrt{\varepsilon_r}} \cdot \frac{b}{W_e + 0.441b} \qquad \text{Ec.7}$$

$$W_e = ancho\ efectivo = \begin{cases} \frac{W}{b} & \frac{W}{b} > 0.35 \\ \frac{W}{b}\left(0.35 - \frac{W}{b}\right)^2 & \frac{W}{b} < 0.35 \end{cases} \qquad \text{Ec.8}$$

Microstrip

La línea Microstrip se ha convertido en la metodología más conocida y más empleada en circuitos de RF y microondas. Emplea un plano de tierra inferior y un conductor de ancho "*W*" en la capa superior. Ambas capas están separadas una distancia "*d*" de material dieléctrico cuya constante es "ε_r" y una permitividad magnética "μ".

En contraste con la línea Stripline, la naturaleza de dos medios (discontinuidad de sustrato: aire + material dieléctrico) hace que su modo dominante sea híbrido (Cuasi-TEM) y no TEM, con el resultado de que la velocidad de fase, la impedancia característica y la variación del campo se vuelvan ligeramente dependiente de la frecuencia.

La línea Microstrip es dispersiva. Para su cálculo se emplea un nuevo concepto, la constante dieléctrica efectiva ε_{eff} que tiene en cuenta que la mayoría de los campos eléctricos están restringidos dentro del sustrato, pero existe una fracción de la energía total dentro del aire sobre la linea.

$$\varepsilon_e = \frac{\varepsilon_r + 1}{2} + \frac{\varepsilon_r - 1}{2} \cdot \frac{1}{\sqrt{1 + \frac{12d}{W}}} \qquad \text{Ec.9}$$

Para el cálculo de la impedancia de línea se emplea la siguiente expresión:

$$Z_0 = \begin{cases} \frac{60}{\sqrt{\varepsilon_r}} \ln\left(\frac{8d}{W} + \frac{W}{4d}\right) & \frac{W}{d} < 1 \\ \frac{120\pi}{\sqrt{\varepsilon_e} \cdot \left[\frac{W}{d} + 1.393 + 0.667 \cdot \ln\left(\frac{W}{d} + 1.444\right)\right]} & \frac{W}{d} > 1 \end{cases} \qquad \text{Ec.10}$$

Ground Coplanar Waveguide

Las GCPW se utilizan como alternativa a la línea Microstrip. La separación entre la tira y el suelo suele ser mayor que el grosor h del sustrato, por lo que el campo GCPW se concentra entre la tira y el plano de tierra del sustrato, comportándose, así como una línea Microstrip.

Al igual que con las líneas Microstrip es necesario conectar ambos planos de masa mediante vías paralelas a la línea de transmisión para reducir las pérdidas por irradiación.

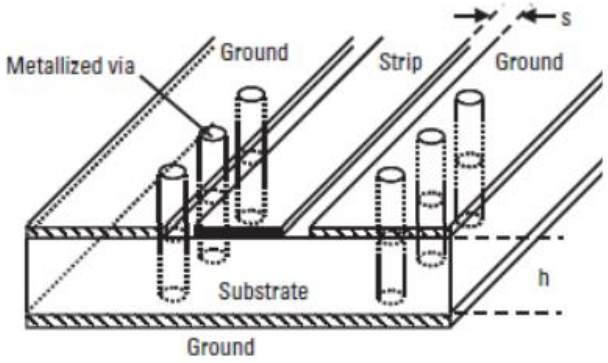


Figura 36. Esquema de una línea GCPW. Fuente: (12).

Para una impedancia y un grosor de sustrato determinados, el ancho de la banda requerido (W) es significativamente menor que el necesario para una línea Microstrip.

$$Z_0 = \frac{60\pi}{\sqrt{\varepsilon_{eff}}} \cdot \frac{1}{\frac{\mathrm{K(k)}}{\mathrm{K(k')}} + \frac{K(k_1)}{K(k_1')}} \qquad \text{Ec.11}$$

$$\mathrm{k} = \frac{\mathrm{a}}{\mathrm{b}} \qquad \text{Ec.12}$$

$$\mathrm{k}' = \sqrt{1 - k^2} \qquad \text{Ec.13}$$

$$\mathrm{k}_1' = \sqrt{1 - k_1^2} \qquad \text{Ec.14}$$

$$\mathrm{k}_1 = \frac{\tanh\left(\frac{a\pi}{4h}\right)}{\tanh\left(\frac{b\pi}{4h}\right)} \qquad \text{Ec.15}$$

$$\varepsilon_{eff} = \frac{1 + \varepsilon_r \cdot \frac{\mathrm{K}(\mathrm{k}')}{\mathrm{K}(\mathrm{k})} \cdot \frac{\mathrm{K}(\mathrm{k}_1)}{\mathrm{K}(\mathrm{k}_1')}}{1 + \frac{\mathrm{K}(\mathrm{k}')}{\mathrm{K}(\mathrm{k})} \cdot \frac{\mathrm{K}(\mathrm{k}_1)}{\mathrm{K}(\mathrm{k}_1')}} \qquad \text{Ec.16}$$

Cálculo de las características de pistas

La tecnología seleccionada ha sido la línea GCPW ya que presenta características similares a la Stripline (no depende de la frecuencia de trasmisión) pero requiere únicamente de 2 planos de cobre. Debido a la complejidad de las ecuaciones mencionadas anteriormente, existen numerosos programas para efectuar el cálculo. Para poder realizarlo, primero es necesario conocer las características de la PCB que se va a fabricar.

Con el objetivo de abaratar costes, se han seleccionado las especificaciones por defecto de la mayoría de los fabricantes. Una PCB fuera de estás especificaciones (por ejemplo, con una capa de dieléctrico de menor grosor) puede incrementar su coste en 10 veces.

Con estos datos, se han realizado pruebas para ajustar el ancho de la pista (W) y la separación con el plano de masa superior (S). Teniendo en cuenta como limitante que el ancho de pista no puede ser mayor al PAD de los componentes de la línea de radiofrecuencia. (Para unos componentes RLC de tamaño 0603 el ancho del PAD corresponde con 0.8 mm).

Para los cálculos se ha empleado la calculadora Sourceforge (13), se trata de una calculadora de pistas de radiofrecuencia online y gratuita. En primer lugar, se han definido las características y especificaciones de la PCB.

- Espesor de dieléctrico: 1.5 mm
- Constante dieléctrica relativa: 4.8 mm
- Espesor de la capa de cobre: 0.035 mm

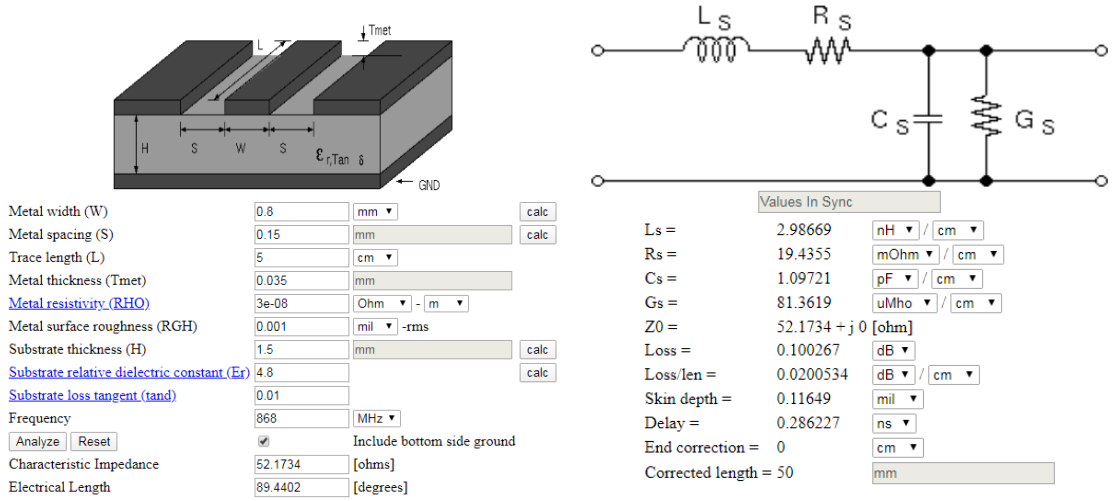


Figura 37. Aplicación online para el cálculo de impedancia de pista. Fuente: Calculadora SourceForge.

Tras una serie de iteraciones se ha optado por un ancho de pista de 0.8 mm y una separación entre pistas de 0.15 mm, consiguiéndose así una impedancia de 52.17 Ω

7.4. Diseño de la tarjeta de circuito impreso PCB

Para el diseño de la PCB se ha empleado el software Eagle de Autodesk. Eagle por defecto, incluye numerosos elementos en su biblioteca, no obstante, ha sido necesario la inserción o creación de nuevos componentes como por ejemplo el integrado ATA8520E.

En primer lugar, se ha elaborado el esquemático global del sistema a partir de los esquemas enunciados anteriormente. Posteriormente se ha procedido al ensamblado en la placa. Para ello, en primer lugar, se ha definido el tamaño requerido de 5 x 5 cm y la configuración de 2 capas. Estas características has sido seleccionadas para abaratar el proceso de producción.

Una vez definido el circuito completo de la PCB diseñada, el siguiente paso es implementar el diseño en la placa impresa. Para ello, es recomendable en primer lugar realizar un posicionamiento preliminar de los distintos integrados de la placa.En primer lugar, los de mayor tamaño (microcontrolador, módulo Bluetooth, módulo SigFox, conectores entradas y salidas…) atendiendo a sus conexiones relativas entre ellos. El siguiente paso es la colocación de los componentes de menor tamaño (condensadores, resistencias…) realizando los ajustes necesarios sobre los dispositivos colocados inicialmente.

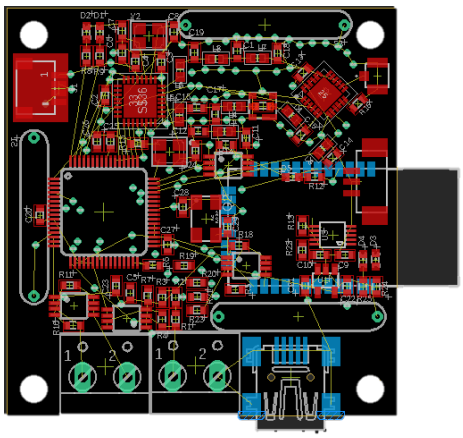


Figura 38. PCB con la dimensión definida y con los elementos colocados. Fuente: elaboración propia.

A continuación, el siguiente paso es el trazado de las distintas pistas, atendiendo a las recomendaciones enunciadas anteriormente.

- Para las pistas de señal se ha empleado un ancho de 0.1524/0.254 mm
- Para las pistas de alimentación se ha empleado un ancho de 0.3048 mm
- Para las pistas de radiofrecuencia se ha empleado un ancho de 0.8 mm

Finalmente, se han extendido los planos de masas superior e inferior de la parte de radiofrecuencia al conjunto de la placa, para así reducir los bucles de retorno de corriente, garantizándose la mayor continuidad posible. A excepción de los osciladores, los cuales es preferible que no dispongan de planos de masa en sus proximidades para un mejor funcionamiento.

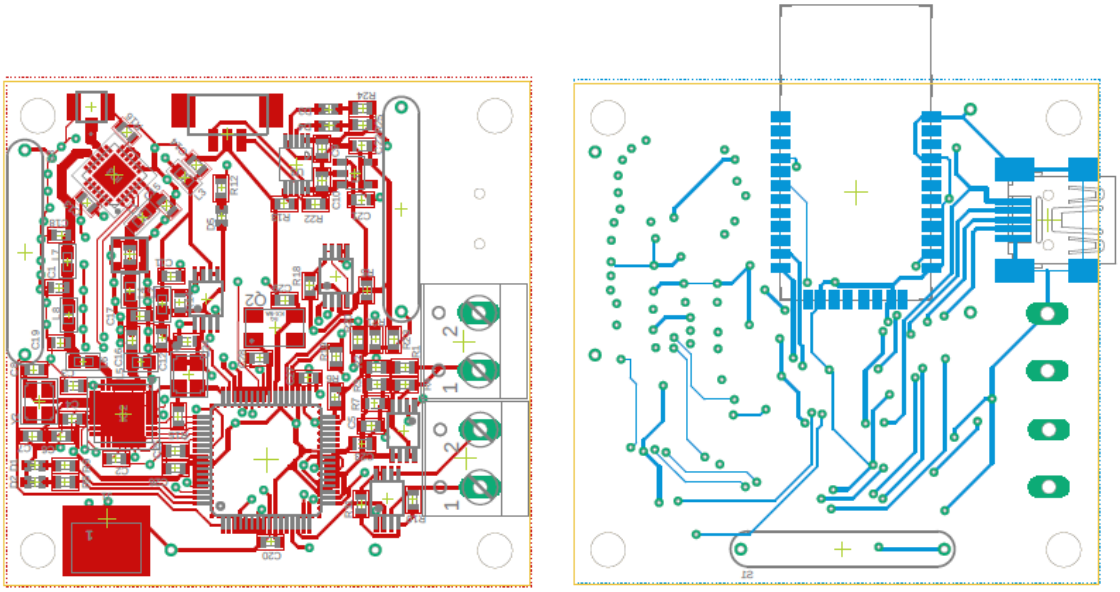


Figura 39. Esquema del trazado de las pistas. Top (izqd.) y Bottom (dcha.). Fuente: elaboración propia.

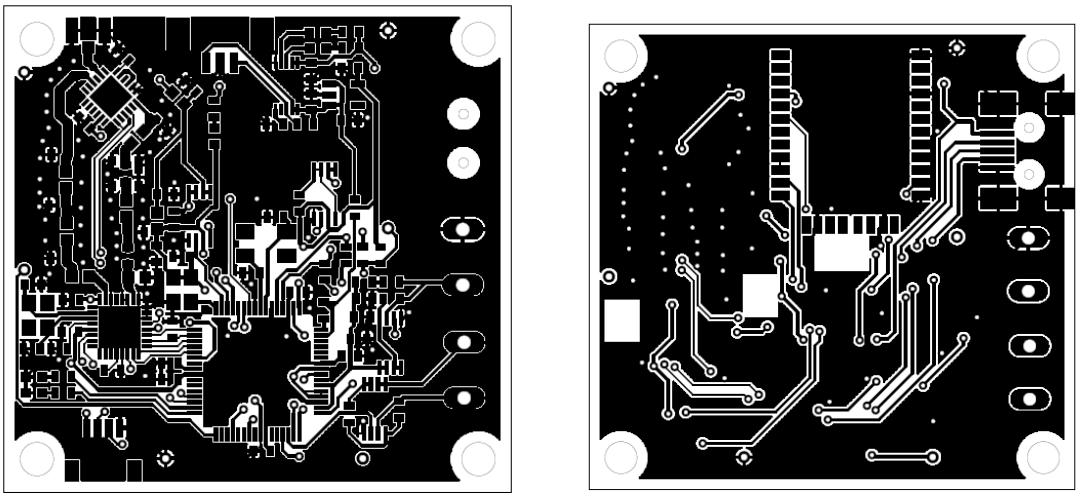


Figura 40. Plano de cobre. Top (izqd.) y Bottom (dcha.). Fuente: elaboración propia.

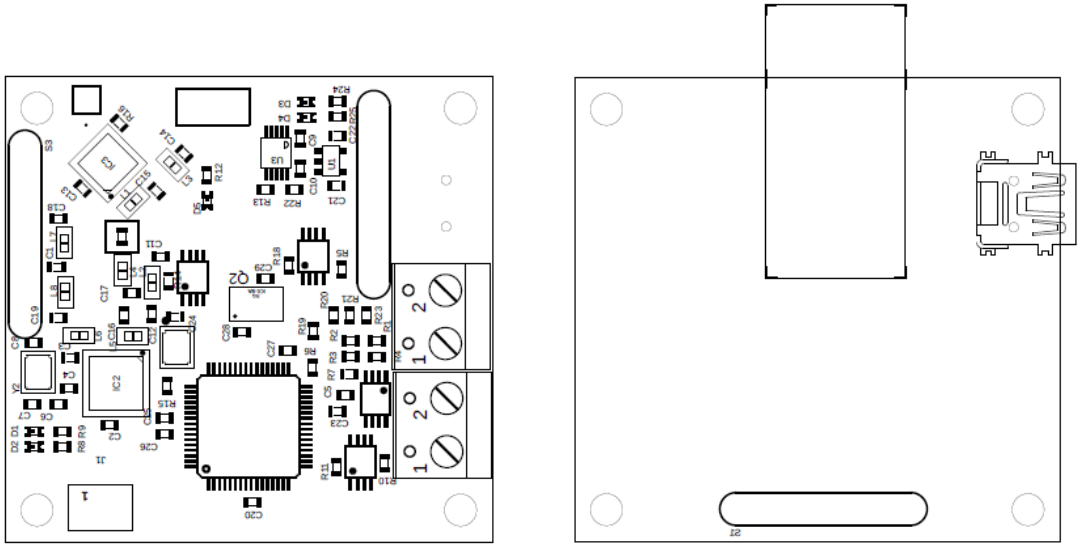


Figura 41. Colocación de los distintos componentes. Top (izqd.) y Bottom (dcha.). Fuente: elaboración propia.

Tras su diseño, se he procedido a su impresión y a su ensamblaje, para ello se ha servido del taller del Departamento de Ingeniería Electrónica.

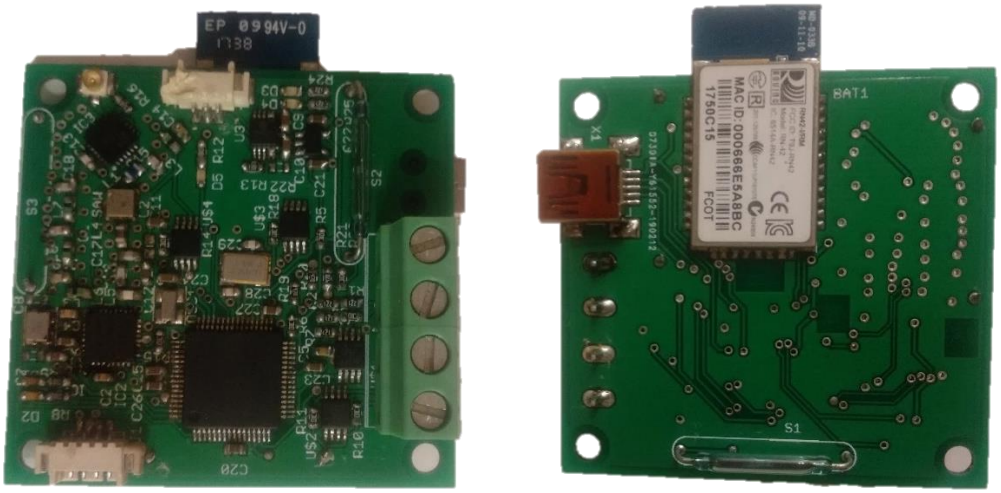


Figura 42. PCB ensamblada y soldada. fuente: elaboración propia.

8. PROGRAMACIÓN DEL MICROCONTROLADOR

8.1. Entorno de programación

Para la programación del microcontrolador PIC24FJ128GC006, se requiere de:

- **IDE (Integrated Development Environment) de Programación:** se trata de un entorno de programación, el cual está formado por un editor de texto, un compilador, un depurador, todo ello empaquetado en una única aplicación.
- **Compilador:** Habitualmente, los compiladores son externos al propia IDE de programación. Se encargan de traducir el código creado por el usuario (habitualmente en lenguaje C) al código ensamblador y a su traducción en binario para la escritura de la memoria de programa del microcontrolador.
- **Programador:** se trata de un dispositivo electrónico que se emplea para la escritura del programa. Realiza la interfaz entre el ordenador (USB o Ethernet) y el microcontrolador (JTAG, ICSP, SWD, EDBG…)
- **Depurador:** habitualmente incluido en el propio programador, permite la verificación del código mediante la ejecución del programa en modo depuración.

8.1.1. IDE de programación

El IDE empleado es el MPLAB IDE v8.92. Se trata de la última versión de la tecnología heredada de MPLAB IDE, creada por Microchip Technology en Microsoft Visual C ++. MPLAB admite la gestión de proyectos, edición, depuración y programación de microcontroladores PIC y DSPIC de 8, 16 y 32 bits.

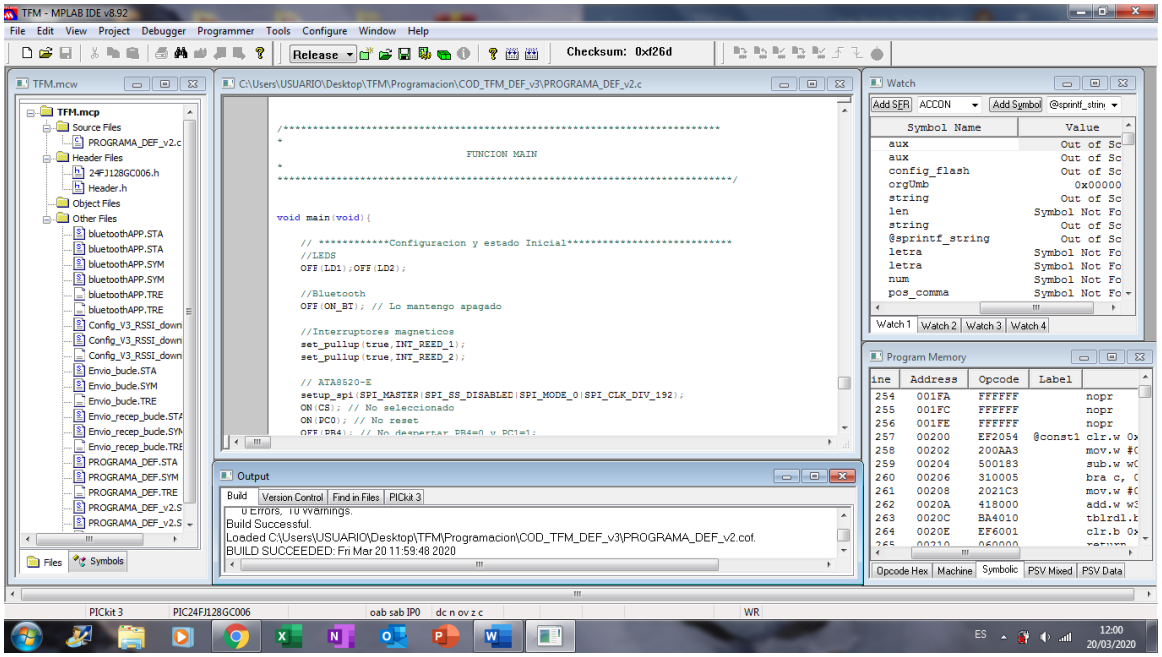


Figura 43. Entorno de programación MPLAB IDE v8.92. Fuente: elaboración propia.

8.1.2.Compilador

El compilador empleado es CCS, CCS fue el primer compilador en código C para microcontroladores del fabricante Microchip. Actualmente continúa ofreciendo soluciones de software a los desarrolladores de aplicaciones integradas que utilizan dispositivos PIC® MCU y PIC24 / dsPIC® DSC.

Los compiladores CCS son fáciles de usar y rápidos de aprender. Incluye un texto detallado explicando el lenguaje C, las funciones desarrolladas en la librería y cómo se pueden aplicar a los microcontroladores PIC.

Características clave del compilador:

- Permite migrar fácilmente entre todos los dispositivos Microchip PIC MCU.
- Minimiza el tiempo de desarrollo con: controladores periféricos y construcciones C estándar.
- Secuencias de entrada / salida de estilo C ++ con formato de datos completo a cualquier dispositivo o para cadenas.
- Permite el uso de librerías CCS y código abierto.
- Funciones convenientes como #bit y #byte permiten colocar variables C en direcciones absolutas.
- El tipo integral de un bit (Short Int) permite al compilador generar código orientado a un Bit muy eficiente.
- Permite definir, configurar y administrar las interrupciones fácilmente.

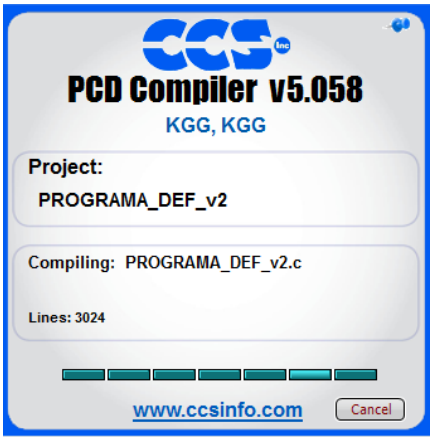


Figura 44. Compilación mediante CCS sobre el IDE MPLAB v8.92. Fuente: elaboración propia.

8.1.3.Programador/Depurador

El programador empleado se trata del PICkit 3. Es un depurador en circuito simple y de bajo coste controlado por un PC que ejecuta el software MPLAB en una plataforma Windows.

El programador/depurador In-Circuit PICkit 3 se basa en la programación en serie en circuito ICSP y Enhanced ICSP (2 hilos).

Las características de PICkit 3 incluyen:

- Soporte USB de alta velocidad con controladores estándar de Windows
- Ejecución en tiempo real
- Procesadores funcionando a velocidades máximas
- Control de sobretensión/cortocircuito incorporado
- Bajo voltaje a 5V (rango de 1.8-5V)
- LED de diagnóstico (encendido, activo, estado)
- Programación de lectura/escritura y memoria de datos del microcontrolador.
- Borrado de todos los tipos de memoria (EEPROM, ID, configuración y programa) con verificación
- Congelación periférica en el punto de ruptura.

Figura 45. Programador/Depurador PICkit 3. Fuente: Microchip.

8.2. Funcionamiento general

El funcionamiento del sistema se ha definido de acuerdo con el siguiente flujograma:

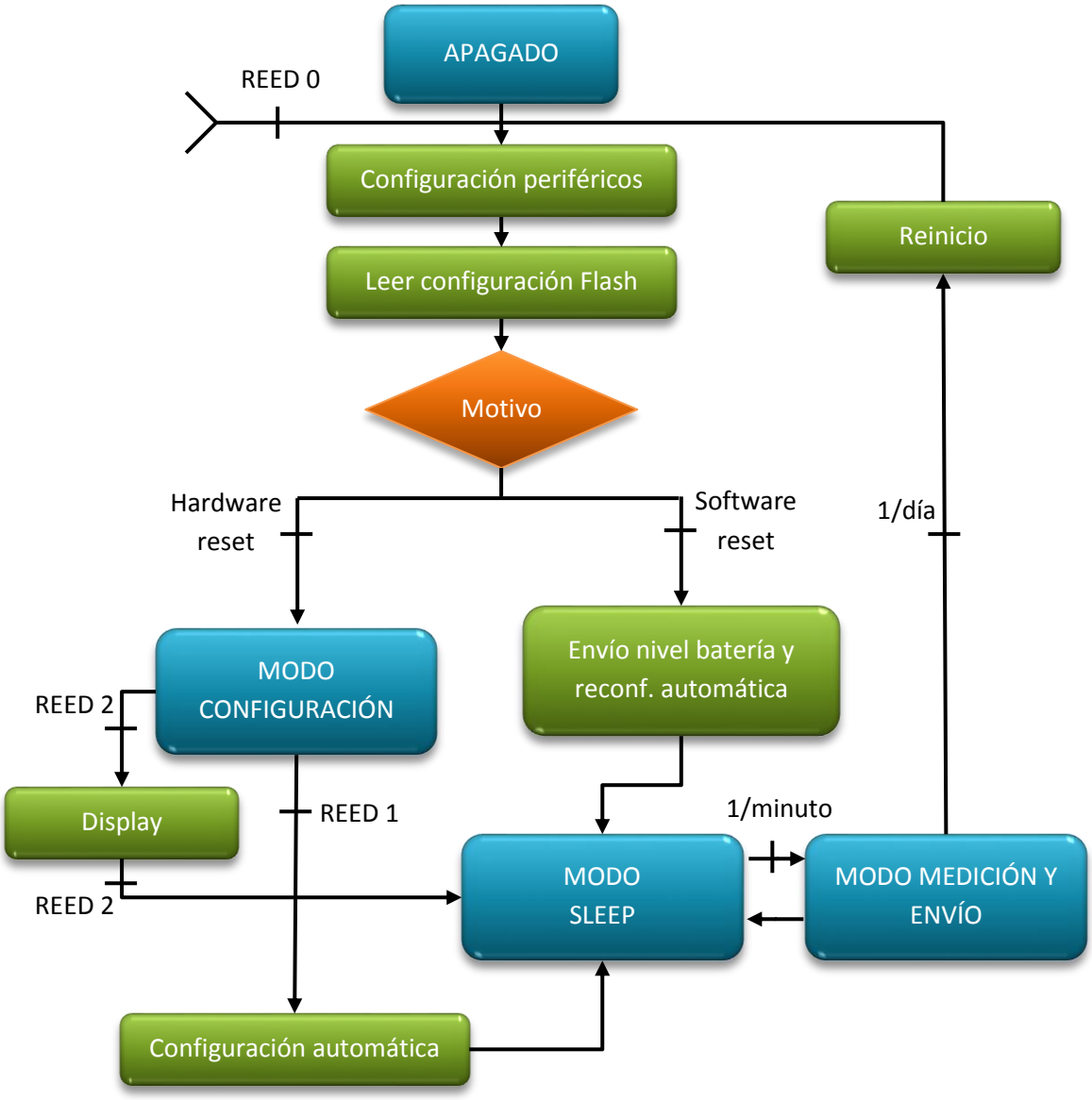


Figura 46. Flujograma del funcionamiento general. Fuente: elaboración propia.

8.2.1.Definición de estados

- **MODO CONFIGURACIÓN:** se emplea para configurar los parámetros de adquisición (frecuencia de muestreo, umbrales de alarma, etc.), calibrar el sensor, comprobar la cobertura, etc. a través de una aplicación móvil conectando con el dispositivo por medio del Bluetooth.
- **PROCESO CONFIGURACIÓN AUTOMÁCTICA:** se emplea para que la configuración sea recibida de forma automática vía un mensaje SigFox.
- **MODO MEDICIÓN Y ENVÍO PERIÓDICO:** se trata de una interrupción en la que se toma medidas del sensor cada minuto, para comprobar los umbrales de alarmas, y se realiza el almacenamiento de las medidas en el paquete de envío y su posterior envío en el momento programado.
- **MODO DISPLAY MEDICIONES:** se trata de un estado en el que se realizan mediciones consecutivas del sensor y se muestran a través del display. Se emplea para comprobar que las mediciones son correctas, sin necesidad de realizar la conexión a través de la aplicación móvil de una forma rápida y sencilla.
- **MODO SLEEP:** se trata de un modo de bajo consumo, donde los periféricos se mantienen bloqueados/apagados.
- **PROCESO REINICIO POR SOFTWARE:** en este modo tras el reinicio diario se realiza una medición del nivel de batería y se envía mediante SigFox con petición de respuesta, la respuesta puede ser configurada para cambiar la configuración actual del sistema.

8.2.2.Proceso

El microcontrolador al encenderse (pasar de estado OFF a ON) en primer lugar configura los periféricos, ajusta los valores de configuración a los guardados anteriormente o a los valores por defecto si no se ha realizado ninguna escritura todavía y entra en MODO CONFIGURACIÓN. Del MODO DE CONFIGURACIÓN se puede salir por cuatro motivos.

- Se active uno de los dos interruptores magnéticos:
  - El interruptor 1: finaliza el modo configuración y activa la configuración automática.
  - El interruptor 2: finaliza el modo configuración y enciende el display donde muestra las lecturas del sensor para su comprobación.
- El usuario envíe el comando de finalización del modo configuración.
- Finalice el timeout (30 segundos) sin que se realice ninguna conexión Bluetooth o se pierda la conexión actual.

En caso de activarse el interruptor magnético 1, el sistema envía un mensaje vía SigFox con petición de respuesta, al recibirse la respuesta, se realiza la configuración y el sistema pasa al MODO SLEEP. En caso de activarse el interruptor magnético 2, el sistema muestra las mediciones a través del display hasta que vuelve a activarse el interruptor, pasando al MODO SLEEP. En caso de que se reciba el comando de finalización de configuración o finalice el timeout, el dispositivo pasa directamente al MODO SLEEP

Del MODO SLEEP, solo se puede salir por dos motivos:

- Salte una interrupción de la RTC y active el MODO MEDICIÓN Y ENVÍO PERIÓDICO, cuando finaliza la interrupción el sistema vuelve al MODO SLEEP

- Se active el interruptor magnético 0, y realice un reinicio (por hardware) al sistema volviéndose así al estado inicial (al igual que un encendido OFF-ON)

En caso de la interrupción de la RTC, el sistema pasa al MODO MEDICIÓN Y ENVÍO PERIÓDICO, donde se realiza la medición del sensor y el envío de mensajes de acuerdo con la configuración establecida. También desde la interrupción se realiza el reinicio (por software) diario programado. Cuando se realiza un reinicio por software, el sistema realiza los ajustes iniciales, pero no entra en MODO CONFIGURACIÓN, en su lugar pasa al PROCESO REINICIO POR SOFTWARE.

En el MODO REINICIO POR SOFTWARE, el sistema envía el nivel de batería vía SigFox y recibe una respuesta la cual puede cambiar la configuración actual del sistema. Al finalizar el sistema vuelve al MODO SLEEP.

En el caso de que se active el interruptor magnético 0, la respuesta del sistema es la misma que durante un encendido OFF-ON.

8.3. Modo configuración

El proceso seguido durante el **MODO CONFIGURACIÓN** se puede resumir de acuerdo con el siguiente flujograma:

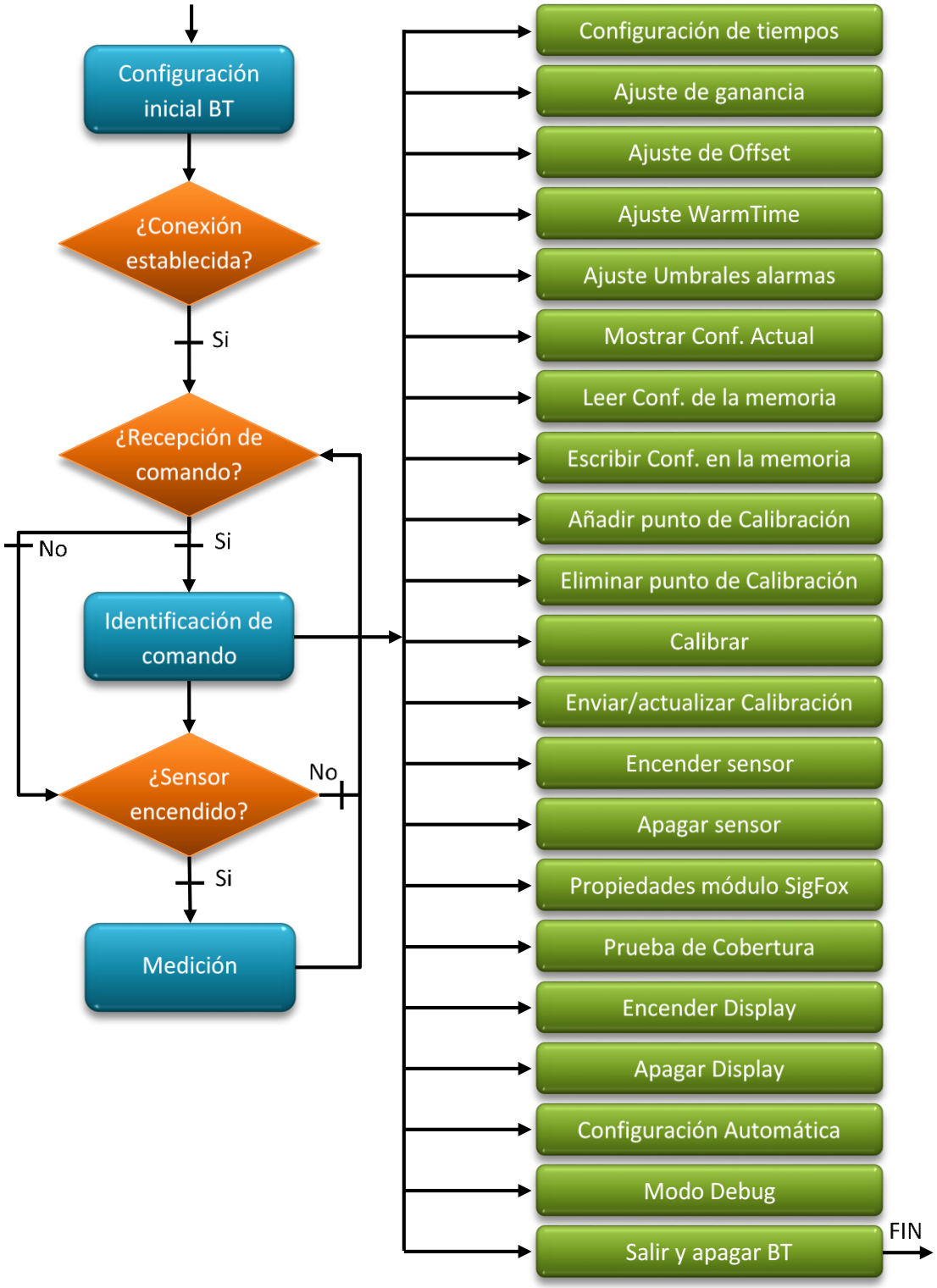


Figura 47. Flujograma del modo configuración 1 de 2. Fuente: elaboración propia.

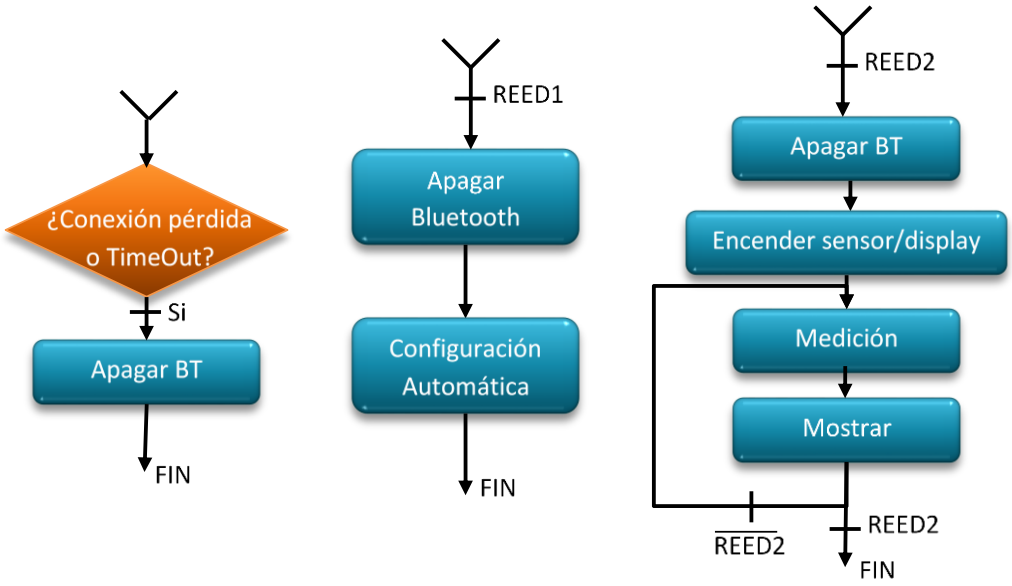


Figura 48. Flujograma del modo configuración 2 de 2. Fuente: elaboración propia.

8.3.1. Definición de estados

Configuración inicial del bluetooth: durante esta etapa se configura las propiedades del módulo bluetooth (modo de funcionamiento, seguridad…)

Esperar conexión: el sistema queda a la espera durante 30 segundos a que se realice la conexión con un dispositivo bluetooth o se active alguno de los interruptores magnéticos.

Recepción de comandos: se realiza una comprobación periódica al buffer del uart para comprobar si se ha recibido algún comando

Identificación de comando: si se recibe algún comando se comprueba de cual se trata y se ejecuta el código correspondiente a ese proceso

Configuración tiempos y tipo de sensor: se recibe por medio de la conexión bluetooth los siguientes parámetros: frecuencia de muestreo, n.º de medidas por paquete de envío, tiempo para realizar el reinicio y el tipo de salida del sensor (modo corriente o tensión).

Ajuste ganancia: se recibe un nuevo valor de ganancia (float).

Ajuste offset: se recibe un nuevo valor de offset (float).

Ajuste WarmTime: se recibe un nuevo valor de tiempo de encendido del sensor (8 bits, 1=10 ms).

Ajuste umbrales alarmas: se recibe el umbral del valor mínimo y del máximo, se recibe en decimal (float) y el microcontrolador calcula su equivalencia a su lectura en la ADC (en bits) teniendo en cuenta el tipo de salida del sensor (corriente o tensión), la ganancia y el offset ajustados. Se almacena únicamente los 8 bits más significativos del valor umbral.

Mostrar configuración actual: se envían todos los parámetros de configuración anteriormente mencionados para ser impresos por la pantalla de la aplicación.

Leer configuración: el microcontrolador lee la última configuración escrita en la memoria de programa y actualiza la actual.

Escribir configuración: el microcontrolador escribe en la memoria de programa la configuración actual. (este proceso también se ejecuta siempre que se cambia alguno de los parámetros de configuración).

Encender sensor: se ajusta la entrada de la ADC al tipo de salida del sensor (corriente o tensión), se activa el divisor de tensiones de la batería y se activa la alimentación del sensor.

Apagar sensor: se corta la alimentación al sensor y el divisor de tensiones de la batería.

Encender display: se ajusta la entrada de la ADC al tipo de salida del sensor (corriente o tensión) y se enciende el display.

Apagar display: se apaga el display

Añadir punto de calibración: se añade un nuevo punto de calibración, se toma una nueva lectura de la adc y se recibe el valor correspondiente mediante el bluetooth.

Eliminar punto de calibración: se elimina el último punto añadido a la lista de calibración.

Calibrar: se realiza una aproximación mediante mínimos cuadrados a los puntos almacenados para la calibración, obteniéndose así un nuevo valor de ganancia y de offset.

Enviar calibración: se envía el valor de los nuevos valores de calibración (ganancia y offset) vía mensaje SigFox para su almacenamiento en la nube.

Propiedades módulo SigFox: el microcontrolador envía por bluetooth las propiedades del módulo SigFox (ID, PAC y versión). Estos datos son necesarios para dar de alta un nuevo producto en la plataforma SigFox.

Prueba de cobertura: el microcontrolador envía un mensaje de prueba vía SigFox para comprobar si hay cobertura en el punto de instalación.

Configuración automática: el microcontrolador envía un mensaje con petición de respuesta para recibir una nueva configuración, una vez recibida se actualiza y la muestra.

Modo Debug: activa y desactiva el modo depuración. En modo depuración se emplean los leds para diferenciar el paso entre los distintos estados, cuando envía, cuando sale del modo Sleep, etc.

Salir: el microcontrolador apaga la alimentación del módulo bluetooth y sale del modo configuración para entrar en modo Sleep.

Realizar mediciones: se muestra una medición de presión y el nivel de batería si se ha activado las mediciones por medio de la aplicación.

Mostrar presión por display: se muestra el valor de presión por el display.

8.3.2. Proceso

Al entrar en el modo configuración, se activa la alimentación del Bluetooth. Espera durante 500 ms y realiza su configuración inicial (el nombre del dispositivo, método de autentificación…). Finalizada la configuración, espera a que se realice una conexión durante 30 segundos. Una vez realizada la conexión, en este punto el sistema realiza comprobaciones periódicas para ver si se ha recibido un nuevo comando. En el caso de que este se reciba, se ejecuta el código correspondiente a ese comando. Paralelamente a las comprobaciones

periódicas, se realizan las mediciones del sensor y del nivel de batería en caso de que éstas estén activadas. Cuando finaliza el proceso de un comando, se vuelve a comprobar si se ha recibido un nuevo comando. En caso de recibir el comando **Configuración tiempos y tipo de sensor**, el dispositivo recibe a continuación el siguiente mensaje de 8 bytes:

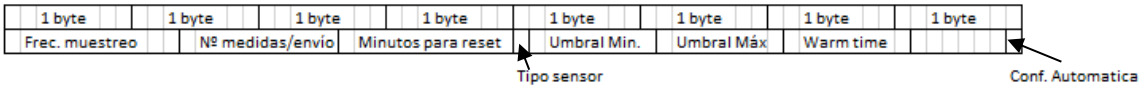


Figura 49. Formato de recepción de configuración. Fuente: elaboración propia

El formato de recepción se trata de 8 bytes, para que éste pueda ser recibido también mediante un mensaje de respuesta de la red SigFox (limitado a 8 bytes). El microcontrolador se encarga de descifrar del mensaje los distintos, parámetros de configuración, actualizar sus parámetros y escribirlos en la memoria permanente.

En caso de recibir el comando **Ajuste de ganancia u offset**, a continuación, se recibe el valor de ganancia u offset según corresponda en formato float de 32 bits. Para el envío ha sido previamente fragmentado en 4 bytes. Una vez recibido se almacena en la memoria permanente de nuevo fragmentado en 4 bytes.

En caso de recibir el comando **Ajuste de umbrales de alarma**, a continuación, se recibe el valor de los umbrales máximo y mínimo en formato decimal (float) de 32 bits. Para su almacenamiento, el microcontrolador debe traducir el valor en formato decimal correspondiente a la magnitud física (por ejemplo: en bares) a su equivalente a la lectura de la ADC (en bits). Para la conversión se emplea los valores de ganancia, offset y tipo de sensor seleccionado previamente definidos.

$$Z_{out\ sensor}(V\ o\ mA) = \frac{Valor - Offset}{G} \qquad \text{Ec.17}$$

$$Z_{ADC}(V) = \begin{cases} Z_{out\ sensor}\ (mA) \cdot \dfrac{R_1}{1000\ mA/A} & Si\ modo\ corriente \\ Z_{out\ sensor}\ (V) \cdot \dfrac{R_1}{R_1 + R_2} & Si\ modo\ tensión \end{cases} \qquad \text{Ec.18}$$

$$Z_{ADC}(bits) = Z_{ADC}(V) \cdot \frac{2^{16}}{Rango(V)} \qquad \text{Ec.19}$$

Finalmente, los valores de alarma máxima y mínima son actualizados y almacenados en la memoria permanente. En caso de recibir el comando **Mostrar configuración actual:** el microcontrolador envía los siguientes mensajes mediante el bluetooth:

Configuración Actual:
Frecuencia X minutos
Tamaño paquete: X medidas/envío
Modo corriente seleccionado ***o*** *Modo tensión seleccionado*
Warm time: X ms
Ganancia: X
Offset: X
Min Alarmas: X
Max Alarmas: X

Donde X, corresponde al valor de ese parámetro en la configuración actual.

En caso de recibir el comando **Leer configuración o Guardar configuración**, el microcontrolador procede a leer la configuración y actualizarla o guardar la actual en la memoria. Para ello se emplea el mismo formato de datos que se ha visto en el comando **Configuración tiempos y tipo de sensor.**

En caso de recibir el comando **Encender sensor,** el microcontrolador ajusta la entrada de la ADC al tipo de sensor seleccionado, activa la alimentación del sensor, activa el divisor de tensiones del nivel de batería y activa en las **Mediciones periódicas** el envío de datos (lectura del sensor y nivel de batería) por medio de mensaje bluetooth. Los mensajes de medición van precedidos con el identificador (*) para su correcta manipulación.

En caso de recibir el comando **Apagar sensor,** se corta la alimentación del sensor, se abre el divisor de tensiones de la batería y se desactiva las mediciones periódicas.

En caso de recibir el comando **Añadir punto de calibración**, a continuación, se recibe el valor de la presión en formato decimal (float de 32 bits) en magnitud física, se toma un nuevo valor de la ADC (V) y se almacenan ambos como un nuevo punto de calibración. El valor de tensión tomado a la entrada de la ADC debe ser convertido al nivel de tensión o corriente a la salida del sensor, para ello se emplea el tipo de sensor seleccionado y la función de transferencia asociada a ese tipo.

$$Z_{out\ sensor} = \begin{cases} Si\ modo\ corriente \rightarrow & Z_{out\ sensor}\ (mA) = \dfrac{1000\ mA/A}{R_1} \cdot Z_{ADC}(V) \\ Si\ modo\ tensión \rightarrow & Z_{out\ sensor}(V) = \dfrac{R_1 + R_2}{R_1} \cdot Z_{ADC}(V) \end{cases} \quad \text{Ec.20}$$

En caso de recibir el comando **Eliminar punto de calibración**¸ se elimina el último punto añadido para la calibración.

En caso de recibir el comando **Listar puntos de calibración**, el microcontrolador envía por un mensaje bluetooth el listado de los distintos puntos almacenadas para la calibración.

[3.58 , 1.2]
[5.22 , 3.8]
[8.32 , 5.95]

En caso de recibir el comando **Calibrar,** el microcontrolador procede a realizar un ajuste de la recta de calibración por los puntos almacenadas. El ajuste se realiza por mínimos cuadrados atendiendo a las siguientes expresiones:

$$y = m \cdot x + n \quad \text{Ec.21}$$

$$m = \frac{N \cdot S_{xy} - S_x \cdot S_y}{N \cdot S_{xx} - S_x \cdot S_x} \quad \text{Ec.22}$$

$$n = \frac{S_{xx} \cdot S_y - S_x \cdot S_{xy}}{N \cdot S_{xx} - S_x \cdot S_x} \quad \text{Ec.23}$$

$$S_x = \sum_{i=1}^{N} x_i \qquad S_y = \sum_{i=1}^{N} y_i \qquad S_{xx} = \sum_{i=1}^{N} x_i^2 \qquad S_{xy} \sum_{i=1}^{N} x_i \cdot y_i \qquad \text{Ec.24}$$

Una vez finalizada la calibración, el microcontrolador envía a través de un mensaje Bluetooth su resultado:

Calibrado: m= 2.5689 n= 1.0543

En caso de que la calibración sea satisfactoria, se debe enviar a continuación el comando de **Enviar calibración**, al recibir este comando el dispositivo actualiza la calibración actual a la última obtenido por el proceso de calibración y envía un mensaje a través de la red SigFox con los valores de ganancia y offset obtenidos. (La secuencia para realizar un envío a través de la red SigFox se detallará más adelante en el apartado correspondiente).

En caso de recibir el comando **Ver propiedades SigFox,** el microcontrolador, despierta el módulo SigFox y extrae sus propiedades (versión, número PAC e ID) y las envía por un mensaje Bluetooth

SigFox Version: 2.3
ID: 18D41
PAC: 6ABC1589

En caso de recibir el comando **Prueba de cobertura**, el microcontrolador envía un mensaje de prueba a través de la red SigFox. Este procedimiento debe ser realizado una vez colocada la antena en su posición definitiva, para así comprobar que se obtiene una buena señal de cobertura.

En caso de recibir el comando **Encender Display,** el microcontrolador, activa la alimentación del display, la del sensor y activa en las **Mediciones periódicas** mostrar la lectura del sensor por medio del display.

El proceso sigue, hasta que se recibe el comando **Apagar Display,** donde se desactivan la alimentación del display y la del sensor (en caso de que no esté activa las mediciones periódicas por bluetooth).

En caso de recibir el comando **Configuración automática,** el microcontrolador envía un mensaje a través de la red SigFox con petición de respuesta. El mensaje de respuesta tiene el mismo formato en que se almacena la configuración en la memoria permanente del microcontrolador.

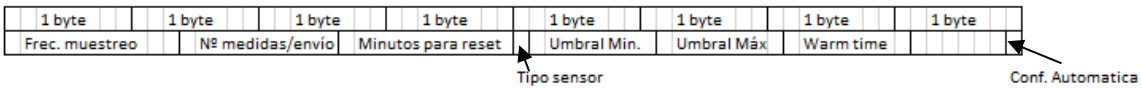


Figura 50. Formato de recepción de configuración. Fuente: elaboración propia.

Una vez se recibe el mensaje, para actualizar la configuración se realiza el mismo procedimiento que cuando se lee la configuración desde la memoria permanente.

En caso de recibir el comando **Modo Debug**, en caso de que el modo depuración esté desactivado se activa y viceversa. En el modo Debug, se emplean los dos leds auxiliares para identificar el paso de un estado a otro, el tiempo de envío y recepción de un mensaje SigFox entre otros casos. También se muestra mensajes con más información durante la conexión bluetooth. A modo de seguridad, ya que el consumo de los leds reduciría drásticamente la autonomía de la

batería, el modo Debug no se almacena en la memoria permanente. De esta forma en caso de que quedase activo, se desactivaría tras el reinicio diario.

En caso de recibir el comando **Salir** o perderse la conexión Bluetooth, el dispositivo saldría del bucle de configuración y apagaría el módulo Bluetooth finalizando así el **MODO CONFIGURACIÓN.** Antes de salir, se desactiva los periféricos que pudiesen haber sido activados (sensor, display, divisor de tensión de batería…)

Si se activan uno de los dos interruptores magnéticos, ya sea durante una conexión bluetooth o durante su espera. El microcontrolador saltaría del modo configuración:

- Y en caso de tratarse del interruptor nº 1: entraría en el **MODO CONFIGURACIÓN AUTOMÁTICA**. Este modo se utiliza para una nueva instalación, en la que el instalador desconoce los parámetros de configuración y se realiza a distancia.
- Y en caso de tratarse del interruptor nº 2: se activaría el display, el sensor y se mostrarían lecturas periódicas del sensor (teniendo en cuenta la configuración almacenada) hasta que este volviese a ser activado (**MODO DISPLAY MEDICIONES)**. Este modo no está destinado al momento de instalación sino a una comprobación rápida de que el sensor realiza mediciones correctas, sin necesidad de realizar la conexión Bluetooth.

  Al finalizar uno de los dos procedimientos, saldría del **MODO CONFIGURACIÓN.**

8.4. Modo medición y envío periódico

El proceso seguido durante el **MODO MEDICIÓN Y ENVÍO,** se puede resumir de acuerdo con el siguiente flujograma:

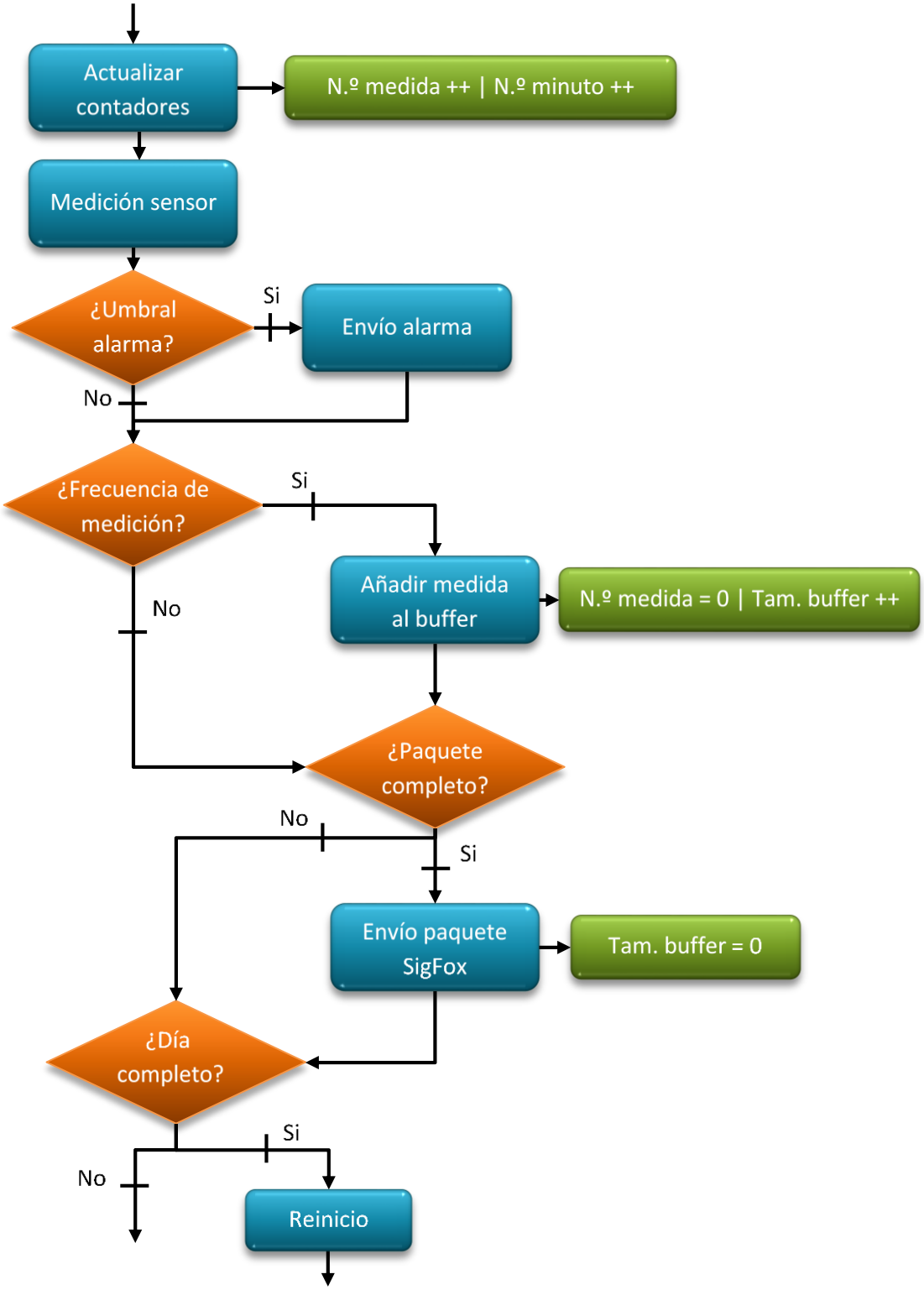


Figura 51.Flujograma modo y envío periódico. Fuente: elaboración propia.

8.4.1. Definición de estados

Actualizar contadores: se actualizan los contadores del N.º de medida realizada y el minuto de ejecución. Ambos contadores se emplean para detectar el momento en que se debe tomar una muestra y el momento en el que se debe enviar un paquete de muestras a través de la red SigFox.

Medición del sensor: se toma una muestra de la lectura del sensor.

Comprobación de alarmas: se comprueba si el valor obtenido de la lectura es inferior o superior a los umbrales de alarmas definidos.

Envío de alarma: se envía el valor de la lectura obtenida precedido con un identificador, para detectar el tipo de alarma.

Comprobar frecuencia de muestreo: se realiza la comprobación si el N.º de medida realizada es igual a la frecuencia de muestreo.

Añadir medida al buffer: se almacena el valor de la lectura y se actualiza el contador del tamaño del buffer y se reinicia el contador del N.º de medida.

Comprobación del tamaño del buffer: se comprueba si el tamaño del buffer es igual al tamaño definido para el paquete de envío.

Envío paquete SigFox: se realiza el envío del buffer almacenado, en caso de que el buffer sea mayor al tamaño máximo de mensaje, este se envía fragmentado en mensajes consecutivos.

Comprobación del minuto de ejecución (día completo): se comprueba si el minuto de ejecución es igual al tiempo para realizar el reinicio diario.

Proceso reinicio por software: tras el reinicio, el dispositivo envía el nivel de batería mediante un mensaje con petición de respuesta, para así poder actualizar la configuración actual.

8.4.2. Proceso

Al **MODO MEDICIÓN Y ENVÍO**, se entra de forma periódica cada minuto desde el **MODO SLEEP.** En primer lugar, se actualizan los contadores del N.º de medida y del N.º de minuto de ejecución. Posteriormente, se toma una medida del sensor, una vez finalizada la medición se comprueban los umbrales de alarmas.

Para la comprobación se tiene en cuenta que, por motivos del límite del tamaño del mensaje de configuración, únicamente se almacenan los 8 bits más significativos del umbral. Por lo tanto, se debe reducir la lectura de la ADC de 16 bits a 8 bits para su comprobación.

En caso de que los umbrales se superen, se envía un mensaje de alarma a través de la red SigFox. El mensaje posee la siguiente estructura:

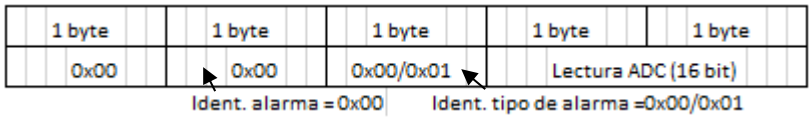


Figura 52. Formato mensaje de alarma. Fuente: elaboración propia.

Todos los mensajes de alarma tienen 5 bytes, los dos primeros corresponde siempre al 0x00, el tercero se trata de un identificador del tipo de alarma, 0x00 si se ha superado el umbral

mínimo y 0x01 si se ha superado el umbral máximo. Los dos últimos bytes corresponden a la lectura de la ADC.

Posteriormente a la comprobación de alarmas, se comprueba si el N.º de medidas (minutos) corresponde al valor del tiempo de muestreo (en minutos). En caso de que coincida, el valor se añade al buffer de medidas y se reinicia el contador del N.º de medidas y se actualiza el contador del tamaño del buffer.

A continuación, se comprueba si el tamaño del buffer corresponde al tamaño del paquete definido, en caso afirmativo, se envía el buffer a través de los mensajes SigFox necesarios. El tamaño máximo de un mensaje SigFox son 12 bytes, que corresponden a 6 medidas (2 bytes = 16 bits).

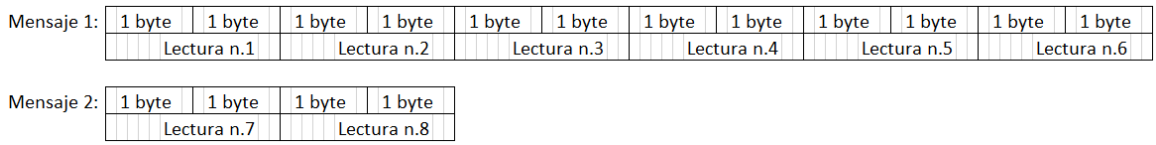


Figura 53. Ejemplo de envío de paquete de 8 lecturas dividido en 2 mensajes. Fuente: elaboración propia.

Una vez finalizado el envío del paquete de medidas o en el caso de que algunas de las comprobaciones anteriores (la comprobación del Nº. de medidas y la del tamaño del buffer) sea negativa, el sistema comprueba si el minuto de ejecución coincide con el minuto para el reinicio, en caso afirmativa procede al proceso de reinicio por software y en caso negativo finaliza el **MODO MEDICION Y ENVÍO**.

En el proceso de reinicio por software, en primer lugar, se realiza un reinicio del dispositivo. Tras el reinicio, el dispositivo toma una muestra del nivel de batería. Una vez realizada la muestra, el dato se envía a través de un mensaje SigFox con petición de respuesta. La respuesta sigue el formato descrito anteriormente:

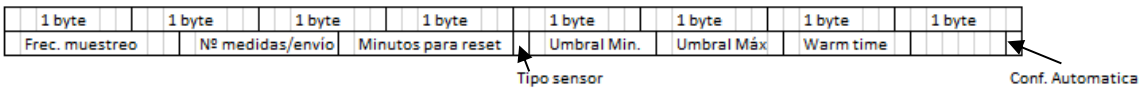


Figura 54. Formato de recepción de configuración. Fuente: elaboración propia.

El proceso es similar al de configuración automática, a diferencia que, en este caso, antes de actualizar los valores de configuración se comprueba el último bit del mensaje. En caso de que este sea igual a 1 se realiza la actualización y en caso contrario, no se tiene en cuenta el mensaje recibido. De esta forma, se puede activar/desactivar la reconfiguración automática a distancia.

8.5. Ajuste de periféricos

En el siguiente apartado, se describen las distintas configuraciones necesarias para el correcto funcionamiento de cada uno de los periféricos.

8.5.1. Comunicación SPI: módulo SigFox (ATA8520E)

Interfaz ATA8520E

El módulo ATA8520E posee una interfaz SPI, admite una frecuencia de hasta 125 kHz, en modo 0 con los siguientes requisitos temporales:

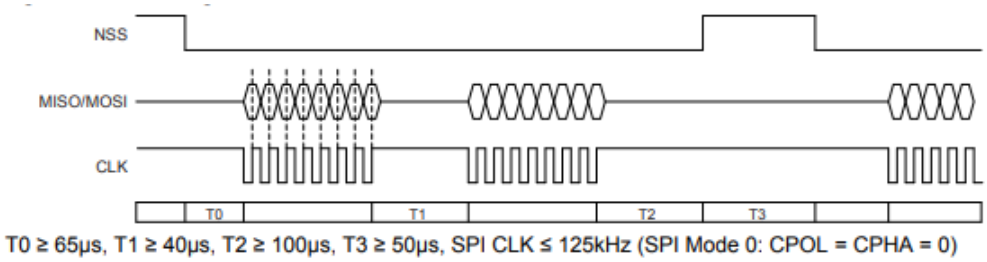


Figura 55. Especificaciones interfaz SPI módulo ATA8520E. Fuente: Datasheet ATA8520E.

En primer lugar, se ha de inicializar la interfaz SPI del microcontrolador a los parámetros descritos anteriormente, para que ambos dispositivos puedan comunicarse. La interfaz SPI se configura a través del siguiente registro:

REGISTER 16-2: SPIxCON1: SPIx CONTROL REGISTER 1

U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	DISSCK(1)	DISSDO(2)	MODE16	SMP	CKE(3)
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSEN(4)	CKP	MSTEN	SPRE2	SPRE1	SPRE0	PPRE1	PPRE0
bit 7							bit 0

Figura 56. Registro de configuración de la interfaz SPI, módulo 1. Fuente: Datasheet PIC24FJ128GC06.

- DISSCK: desactiva el pin SCKx (modo master)
  - 1 = desactivado.
  - 0 = activado.
- DISSDO: desactiva el pin SDOx
  - 1 = desactivado
  - 0 = activado
- MODE16: tamaño del paquete de mensaje
  - 1 = 16 bits
  - 0 = 8 bits
- SMP: momento de toma de dato
  - 1 = al final de la transmisón del dato
  - 0 = en medio de la tranmisión del dato
- CKE: flanco de reloj activo
  - 1 = transicciones de flanco activo a no activo
  - 0 = transicciones de flanco no activo a activo
- SSEN: activación del pin SS
  - 1 = desactivo
  - 0 = activo
- CKP: polaridad del reloj CLK
  - 1 = activo a nivel bajo
  - 0 = activo a nivel alto
- MSTEN: selección del modo del SPI
  - 1 = Master mode
  - 0 = Slave mode
- SPRE: divisor de reloj secundario:
  - 111 = 1:1
  - 110 = 2:1
  - • • •
  - 000 = 8:1
- PPRE: divisor de reloj primario:
  - 11 = 1:1
  - 10 = 4:1
  - 01 = 16:1
  - 00 = 64:1

La configuración empleada para el registro se resume en la siguiente tabla:

Tabla 11. Parámetros del registro de configuración del SPI.

Parámetro	Valor asignado	Equivalente librería CCS
DISSCK	0	Default
DISSDO	0	Default
MODE16	0	SPI_MODE_8B (0x0000)
SMP	0	SPI_SAMPLE_AT_MIDDLE (0x0000)
CKE	1	SPI_XMIT_L_to_H (0x0100)
SSEN	1	SPI_SS_DISABLED (0x0080)
CKP	0	SPI_SCK_IDLE_LOW(0x0000)
MSTEN	1	SPI_MASTER (0x0020)
SPRE **PPRE**	101 00	SPI_CLK_DIV_192 (0x0014)

Además de la interfaz SPI, el módulo ATA8520E posee unos pines de entradas y salidas para la interacción con un controlador externo.

- **PC0 (INPUT: NRESET):** realiza un reinicio al integrado, cuando se mantiene a nivel bajo de tensión
- **PC1 (INPUT: NPWRON1):** permite despertar al ATA8520E de su estado de reposo, mediante un cambio a estado bajo de tensión
- **PB4 (INPUT: PWRON):** mismo significado que NPWRON, pero con estado de tensión alto.
- **PB6 (OUTPUT: EVENT):** permite la detección de eventos cuando pasa a nivel bajo.

Para la configuración y el uso del dispositivo se emplea una serie de comandos. A continuación, se resume los comandos empleados.

Comandos ATA8520E

En primer lugar, es necesario configurar el módulo ya que por defecto no viene configurado para su utilización. La configuración se mantiene almacenada en la memoria EEPROM del dispositivo ATA8520E, por lo que solo es necesario configurarlo una vez antes de usarlo.

Configuración del sistema (0x11): Permite la definición de los puertos de salida/entrada, de determinados parámetros de la red SigFox (Número de mensajes por envío, modo envío/recepción, la región de funcionamiento) y el nivel de tensión de alimentación. Tras la configuración se realiza un evento.

Master		Store Sys Conf (0x11)	EDDRC	EPORTC	repeat	SysConf
ATA8520E		Dummy	Dummy	Dummy	Dummy	Dummy

Figura 57. Comando de configuración del sistema. Fuente: ATA8520E Datasheet.

- **EDDRC:** define el tipo de GPIO de los pines del puerto C (0: input, 1: output)
- **EPORTC:** define el valor de salida de los pines del puerto C en caso de que estén configurados como salida o activa la resistencia de pull-up cuando están definidos como entradas.
- **Repeat:** Define el número de mensajes que son enviados en cada transmisión:

  - 0x00: envía 1 mensaje
  - 0x01: envía 2 mensajes
  - 0x02: envía 3 mensajes (valor por defecto, definido por el protocolo SigFox)
- **SysConf:**
  - **Bit 7-4:** sin función, valor 0xb0011
  - **Bit 3:** selecciona la tensión de alimentación (0: 5V, 1: 3V)
  - **Bit 2:** selecciona el modo TX/RX, (0: envío/recepción activos, 1: solo envío activo)
  - **Bit 1:** selecciona la región de operación (0: US, 1: EU)
  - **Bit 0:** sin función, valor 0xb1

Para el dispositivo se ha empleado la siguiente configuración:

- **EDDRC:** 0x00: Todos los pines definidos como entrada
- **EPORTC**: 0x00: Todos los pines con el pull-up desactivado
- **Repeat:** 0x02: Envió de mensaje con 3 repeticiones
- **SysConf: 0x3B (0b00111011)**
  - **Bit 7-4:** 0xb0011
  - **Bit 3: 0xb1** tensión de alimentación de 3V
  - **Bit 2: 0xb0** modo TX/RX de envío/recepción
  - **Bit 1: 0xb1** Región de EU (europa)
  - **Bit 0:0xb1** 0xb1

Configuración de frecuencias (0x1A): Este comando se emplea para configurar las frecuencias en las que el módulo trabaja para el envío y la recepción de mensajes. El valor de frecuencia se define en un entero de 32 bits sin signo en hertzios. Tras la configuración se realiza un evento.

Master		Store Frequen-cies (0x1A)	freqTX [7..0]	freqTX [15..8]	freqTX [23..16]	freqTX [31..24]	freqRX [7..0]	freqRX [15..8]	freqRX [23..16]	freqRX [31..24]
ATA8520E		Dummy	Dummy	Dummy	Dummy	Dummy	Dummy	Dummy	Dummy	Dummy

Figura 58. Comando de configuración de frecuencias. Fuente: ATA8520E Datasheet.

Para la región de Europa las frecuencias son:

- **TX:** .868.130.000 Hz = 0x33BE9CD0
- **RX:** 869.525.000 Hz = 0x33D3E608

Hay que tener en cuenta que los bytes se almacenan en orden inverso, por lo que el mensaje de configuración realizado es:

Comando	**Frecuencia TX**				**Frecuencia RX**			
0x1A	0xD0	0x9C	0xBE	0x33	0x08	0xE6	0xD3	0x33

Ajuste de valor RSSI: permite corregir la ganancia o pérdida derivada del circuito de antena (incluida la propia antena). El valor RSSI se define en entero de 8 bits con signo.

Master		Adjust RSSI(0x24)	Value
ATA8520E		Dummy	Dummy

Figura 59. Comando de ajuste del valor RSSI. Fuente: ATA8520E Datasheet.

Teniendo en cuanta la siguiente expresión. Se ha ajustado a un valor de 5 dB.

$$RSSI(db) = 0{,}5 \cdot RSSI(bits)$$

Comando	Valor RSSI
0x24	0x0A

Leer códigos de estado: permite leer el estado interno del dispositivo y limpiar el evento generado (PB6) tras una de las siguientes operaciones:

- Tras un encendido o un reinicio.
- Cuando finaliza una operación de envío/recepción.
- Cuando realiza una lectura de temperatura y tensión de alimentación.
- Cuando finaliza una escritura en la memoria EEPROM.
- Cuando finaliza el modo test.

Master		Get Status (0x0A)	Dummy	Dummy	Dummy	Dummy	Dummy
ATA8520E		Dummy	Dummy	SSM status	Atmel status	SIGFOX status	SIGFOX status2

- **SSM status:** empleado internamente
- **Atmel status:**
  - **Bit 6:** sistema listo para trabajar.
  - **Bit 5:** mensaje enviado.
  - **Bit 4-1:** código de error.
    - **0xb0000:** sin error.
    - **0xb0001:** comando erróneo.
    - **0xb0010:** error genérico.
    - **0xb0011:** error en la frecuencia.
    - **0xb0100:** error de uso.
    - **0xb0101/0110:** error de apertura y cierre.
    - **0xb0111:** envío erróneo.
  - **Bit 0:** PA on/off indicador.
- **SigFox status:**
  - **0x00:** sin error.
  - **0x30:** longitud de mensaje mayor a 12 bytes.
  - **0x3E:** timeout superado en la recepción del mensaje.
  - **0x4E:** timeout superado en la recepción de 1 bit.
- **SigFox status2:**
  - **0x00:** sin error.
  - **0x10:** error de inicialización.
  - **0x18:** error durante el envío.

- **0x40:** error en la frecuencia RF.
- **0x68:** erro durante la espera para la recepción.

Leer la versión Atmel: se utiliza para comprobar la versión firmware instalada en el dispositivo.

Master		Atmel Version (0x06)	Dummy	Dummy	Dummy
ATA8520E		Dummy	Dummy	MajorVers	MinorVers

Figura 60. Comando de leer versión atmel. Fuente: ATA8520E Datasheet.

Leer el número ID: se emplea para leer el número ID (4 bytes) necesario para su registro.

Master		Get ID (0x12)	Dummy	Dummy	.....	Dummy
ATA8520E		Dummy	Dummy	UID[3]		UID[0]

Figura 61. Comando de leer el número ID. Fuente: ATA8520E Datasheet.

Leer el número PAC: se emplea para leer el número PAC (16 bytes) necesario para su registro.

Master		Get PAC (0x0F)	Dummy	Dummy	.....	Dummy
ATA8520E		Dummy	Dummy	PAC ID[0]		PAC ID[15]

Figura 62. Comando de leer el número PAC. Fuente: ATA8520E Datasheet.

Activar el modo OFF: como se ha comentado anteriormente el módulo se despierta del modo OFF mediante la activación de alguno de sus pines (NPWRON o PWRON). En cambio, para entrar en el modo OFF es necesario realizarlo mediante el siguiente comando.

Master		OFF Mode (0x05)
ATA8520E		Dummy

Figura 63. Comando de activar el modo Sleep. Fuente: ATA8520E Datasheet.

Escribir el buffer de envío: se emplea para almacenar el mensaje que se quiere enviar. El mensaje tiene un tamaño máximo de 12 bytes.

Master		Write TX Buffer (0x07)	RF TX Num bytes	RF TX Bytes 0	.....	RF TX Num bytes-1
ATA8520E		Dummy	Dummy	Dummy		Dummy

Figura 64. Comando de escritura del buffer de envío. Fuente: ATA8520E Datasheet.

Enviar mensaje: se emplea para iniciar el proceso de envío de un mensaje previamente cargado en la memoria. Esta operación tarda aproximadamente 7 segundos (con 3 repeticiones activas) y al finalizar se genera un evento en PB6.

Master		Send Frame (0x0D)
ATA8520E		Dummy

Figura 65. Comando de inicialización del envío. Fuente: ATA8520E Datasheet.

Envío/recepción de mensaje: se emplea para iniciar el envío igual que el comando anterior, pero en este caso con petición de respuesta. La operación finaliza aproximadamente a los 50 segundos, generando un nuevo evento en PB6.

Master		Send/Receive Frame (0x0E)
ATA8520E		Dummy

Figura 66. Comando de inicialización del envío con petición de respuesta. Fuente: ATA8520E Datasheet.

Leer el buffer de recepción: se emplea para leer el mensaje recibido.

Master		Read RX Buffer (0x10)	Dummy	Dummy	.....	Dummy
ATA8520E		Dummy	Dummy	RX Byte 0		RX Byte 7

Figura 67. Comando de lectura del buffer de recepción. Fuente: ATA8520E Datasheet.

8.5.2.Comunicación UART: módulo bluetooth (rn42)

Interfaz RN-42

El módulo RN-42 posee una interfaz doble, UART y USB, la comunicación se ha realiza a través de la interfaz UART ya que es más fácilmente controlable desde un microcontrolador.

A su vez, el módulo permite dos modos de funcionamiento:

- **Serial Port Profile (SPP):** exclusivo de la interfaz UART, simula una interfaz serie, es decir permite una comunicación transparente como si dos líneas UART estuvieran unidas.
- **Host Controller interface (HCI):** apto para UART y USB, el módulo bluetooth no emplea la pila Bluetooth, sino que se emplea a modo de radio, realizando funcionalidades MAC a bajo nivel

Por su sencillez y los bajos requerimientos se ha decidido el empleo del modo SPP a través de UART. Por defecto, la interfaz UART viene configurada a 115200 bps, 8 bits, no paridad y 1 bit de stop. La interfaz UART del microcontrolador se inicializa a través del siguiente registro:

REGISTER 18-1: UxMODE: UARTx MODE REGISTER

R/W-0	U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
UARTEN(1)	—	USIDL	IREN(2)	RTSMD	—	UEN1	UEN0
bit 15							bit 8

R/W-0, HC	R/W-0	R/W-0, HC	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	RXINV	BRGH	PDSEL1	PDSEL0	STSEL
bit 7							bit 0

Figura 68. Registro de configuración de la interfaz UART. Fuente: Datasheet PIC24FJ128GC06.

- UARTEN: activación del puerto UART
  - 1 = activado
  - 0 = desactivado
- USIDL: si se paraliza en modo Idle
  - 1 = si se paraliza
  - 0 = no se paraliza
- IREN: IrDA enconder/decoder
  - 1 = activado
  - 0 = desactivado
- RTSMD: selección del modo RTS
  - 1= Simplex mode
  - 0= Flow control
- UEN: activación de pines
  - 11 = TX, RX y BCLK son activos
- ABAUD: detección automática de la velocidad (baudrate)
  - 1 = activo
  - 0 = desactivo
- RXINV: Inversión de polaridad en la recepción
  - 1 = RX desactivo es 0
  - 0 = RX desactivo es 1
- BRGH: Activación de alta velocidad
  - 1 = activo
  - 0 = desactivo
- PDSEL: Paridad y tamaño de mensaje
  - 11 = 9 bits y no paridad
  - 10 = 8 bits y paridad impar
  - 01 = 8 bits y paridad par

- 10 = TX, RX, CTS y RTS activos
- 01 = TX, RX y RTS activos
- 00 = TX y RX activos

- WAKE: despertar mediante interrupción
  - 1 = activo
  - 0 = desactivo
- LPBACK: activa el Loopback
  - 1 = activo
  - 0 = desactivo
  - 00 = 8 bits y no paridad
- STSEL: bit de stop
  - 1 = 2 bits de stop
  - 0 = 1 bit de stop

Además de este registro de configuración, posee otro registro para seleccionar la velocidad de transmisión (UXBRG)

$$Baud\ rate = \begin{cases} \dfrac{F_{cyy}}{16 \cdot (UxBRG + 1)} & Si\ BRGH = 0 \\ \dfrac{F_{cyy}}{4 \cdot (UxBRG + 1)} & Si\ BRGH = 1 \end{cases} \qquad \text{Ec.25}$$

La configuración empleada para el registro se resume en la siguiente tabla:

Tabla 12. Parámetros del registro de configuración del UART. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
UARTEN	1	UART activo
USIDL	0	No stop en Idle
IREN	0	Desactivado
RTSMD	0	Flow control
UEN	0	TX y RX activos
WAKE	0	Desactivado
LPBACK	0	Desactivado
ABAUD	0	Desactivado
RXINV	0	Desactivado
BRGH	0	Velocidad standard
PDSEL	00	8 bits y no paridad
STSEL	0	1 bit de stop

$$UxBRG = 0x02 \rightarrow Baud\ rate = \begin{cases} \dfrac{11.059.200/2}{16 \cdot (2 + 1)} = 115.200\ bps \end{cases} \qquad \text{Ec.26}$$

Finalizada la configuración del periférico UART1 del microcontrolador, el siguiente paso es configurar el módulo Bluetooth. Para ello el módulo tiene dos modos de funcionamiento:

- Modo comando: admite comandos de configuración.
- Modo transparente: envío de datos a través de la conexión bluetooth.

Para entrar en modo comando, es necesario enviar la cadena '$$$', mientras que para volver al modo transparente se puede realizar de varias formas:

- Si no se recibe ningún comando en 60 segundos.
- Si se recibe la cadena '---<cr>'.
- Si se reinicia el módulo.

Los comandos empleados para la configuración del módulo han sido los siguientes:

- SF,1: realiza un reset a valores de fábrica.
- SM,0: modo esclavo.
- SA,0: modo abierto, emplea Bluetooth versión 2.0 sin encriptación.
- SN,MiniFOX: cambia el nombre del dispositivo a MiniFOX.
- R,1: realiza un reinicio al módulo.

Una vez realizada la configuración, el módulo se puede emplear como si se tratara de un puerto serial entre el microcontrolador y el dispositivo conectado. Además de las líneas TX y RX, posee un pin que permite detectar cuando hay una vinculación activa.

8.5.3.Comunicación I2C: display (SSH1106)

El display seleccionado posee una interfaz I2C, para configurar la interfaz I2C en el microcontrolador se requiere el ajuste del siguiente registro:

REGISTER 17-1: I2CxCON: I2Cx CONTROL REGISTER

R/W-0	U-0	R/W-0	R/W-1, HC	R/W-0	R/W-0	R/W-0	R/W-0
I2CEN	—	I2CSIDL	SCLREL	IPMIEN	A10M	DISSLW	SMEN
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC	R/W-0, HC
GCEN	STREN	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

Figura 69. Registro de configuración de la interfaz I2C. Fuente: Datasheet PIC24FJ128GC06.

- I2CEN: activación del puerto I2C
  - 1 = activado
  - 0 = desactivado
- I2CSIDL: si se paraliza en modo Idle
  - 1 = si se paraliza
  - 0 = no se paraliza
- IPMIEN: activación IPMI (Intelligent Platform Management Interface)
  - 1 = activo
  - 0 = desactivo
- A10M: selección de la longitud de dirección de esclavo
  - 1 = 10 bits
  - 0 = 7 bits
- DISSLW: activación del Slew Rate Control
  - 1 = desactivado
  - 0 = activado
- SCLREL, GCEN, STREN: solo modo Slave I2C
- SMEN: activación de las umbrales en las entradas/salidas SMBus
  - 1 = activado
  - 0 = desactivado
- ACKDT: inversión del acknowledge
  - 1 = invertido NACK
  - 0 = normal ACK
- ACKEN: inicia el acknowledge
  - 1 = activado
  - 0 = desactivado
- RCEN: habilitación de recepción
  - 1 = activado
  - 0 = desactivado
- PEN: inicia de condición de parada
  - 1 = activado
  - 0 = desactivado
- RSEN: habilitación de inicio repetido
  - 1 = activado
  - 0 = desactivado
- SEN: habilitación de inicio
  - 1 = activado
  - 0 = desactivado

Tabla 13. Parámetros del registro de configuración del I2C. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
I2CEN	1	I2C activado
I2CSIDL	1	Stop en IDLE
IPMIEN	0	IPMI desactivado
A10M	0	Dirección de esclavo de 7 bits
DISSLW	0	Slew rate activado
SMEM	0	Desactivado SMBus
ACKDT	0	Acknowledge normal

La velocidad de comunicación se ajusta en el registro I2CXBRG de acuerdo a la siguiente expresión:

$$FSCL = \frac{F_{CY}}{I2CxBRG + 1 + \frac{F_{CY}}{10.000.000}} \qquad \text{Ec.27}$$

Por lo tanto, para definir una velocidad de 400 kHz se ajusta el registro al siguiente valor:

$$I2CxBRG = \frac{11.059.200/2}{400.000} - \frac{\frac{11.059.200}{2}}{10.000.000} - 1 = 12.27 \approx 12 = 0x0C \qquad \text{Ec.28}$$

Finalizada la configuración del periférico I2C1 del microcontrolador, el siguiente paso es configurar el display. Su configuración se realiza mediante comandos, para la identificación de los comandos se debe seguir el siguiente formato:

Start bit	Address	0x00	Command	Stop bit

Figura 70. Esquema de codificación de comandos para el Display. Fuente: elaboración propia.

Para la inicialización del display se deben realizar los siguientes comandos:

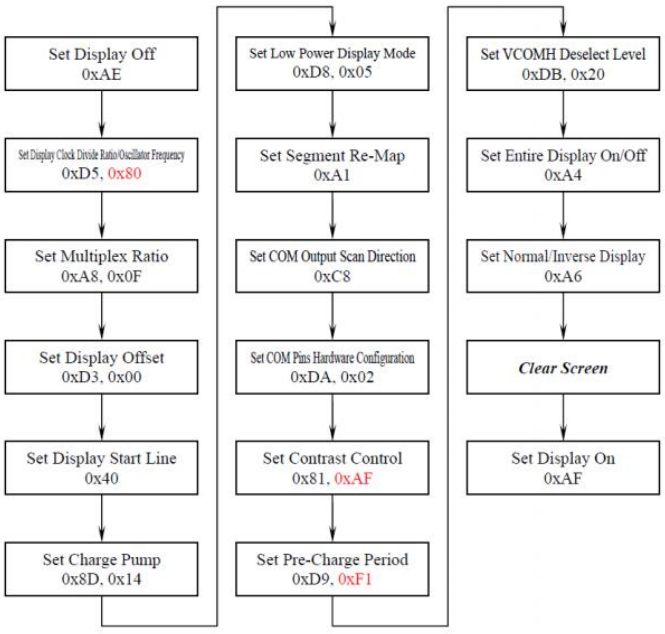


Figura 71. Secuencia de comandos de inicialización del display. Fuente: (14)

Una vez inicializada, la forma de trabajar con el display será la escritura completa de todos los píxeles de la pantalla. Para ello en primer lugar se le debe indicar que se quiere escribir el valor de todos los píxeles, esto se realiza indicando todas las columnas y páginas comprendidas en el display (la pantalla es dividida en 128 columnas y 8 páginas). Una vez configurado, el display actualizará los valores de sus píxeles cuando se reciban datos, para la recepción de datos se requiere el siguiente formato:

Start bit	Address	0x40	Data	Stop bit

Figura 72.Esquema de codificación de envío de datos. Fuente: elaboración propia.

Para la escritura, de las distintas letras y números se posee una librería en la que incluye todos los signos alfanuméricos con un tamaño de 5 x 8 bits. Se dispone a su vez de funciones para escalar los signos y colocarlos en una posición determinada sobre la matriz que representa la pantalla.

Por ejemplo, el nueve se representa mediante: 0x26, 0x49, 0x49, 0x49, 0x3E

26	49	49	49	3E
0	1	1	1	0
1	0	0	0	1
1	0	0	0	1
0	1	1	1	1
0	0	0	0	1
1	0	0	0	1
0	1	1	1	1
0	0	0	0	0

Figura 73. Ejemplo del número 9 representado en la librería. Fuente: elaboración propia

8.5.4. Interrupción RTC

La interrupción RTC se emplea para despertar el microcontrolador de forma periódica (1 minuto) para realizar la medición de presión y el envío cuando le corresponde. Para ello en primer lugar, se debe configurar el módulo de tiempo real (RTC) y posteriormente la alarma. Por defecto, permanece seleccionado el External Secondary Oscillator (SOSC), por lo que únicamente se debe configurar del bit de activación del módulo RTC (RTCEN).

En cuanto a la alarma es necesario realizar la siguiente configuración:

REGISTER 23-3: ALCFGRPT: ALARM CONFIGURATION REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ALRMEN	CHIME	AMASK3	AMASK2	AMASK1	AMASK0	ALRMPTR1	ALRMPTR0
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ARPT7	ARPT6	ARPT5	ARPT4	ARPT3	ARPT2	ARPT1	ARPT0
bit 7							bit 0

Figura 74. Registro de configuración de la alarma RTC. Fuente: Datasheet PIC24FJ128GC06.

- ALRMEN: activación de la alarma RTC
  - 1 = activada
  - 0 = desactivada
- CHIME: repetición
  - 1 = repetición activa
  - 0 = repetición desactivada
- AMASK: máscara de configuración
  - 0000 = cada medio segundo
  - 0001 = cada segundo
  - 0010 = cada 10 segundos
  - 0011 = cada minuto
  - 0100 = cada 10 minutos
- ALRMPTR: secuencia de comprobación
  - 11: comprueba año
  - 10: comprueba mes y día
  - 01: comprueba día de la semana y hora
  - 00: comprueba minuto y segundo
- ARPT: número de repeticiones (si CHIME=0)
  - 1111111 =255 repeticiones
  - • • •
  - 0000000= 0 repeticiones

- 0101 = cada 1 hora
- 0110 = cada día
- 0111 = cada semana
- 1000 = cada mes
- 1001 = cada año

Tabla 14. Parámetros del registro de la alarma RTC. Fuente: elaboración propia.

Parámetro	Valor asignado	Equivalente librería CCS
ALRMEN	1	RTC_ALARM_ENABLE
CHIME	1	RTC_CHIME_ENABLE
AMASK	0011	RTC_ALARM_MINUTE
ALRMPTR	00	default
ARPT	00000001	1

Además de los registros anteriores, posee una serie de registros para configurar el tiempo actual y el tiempo de alarma. Para su uso se ha configurado el mismo tiempo de alarma que de tiempo actual, añadiendo 5 minutos.

8.5.5.ADC Sigma-Delta

El periférico Sigma-delta ADC se emplea para la lectura del sensor, para su empleo ha sido necesario realizar la siguiente configuración:

REGISTER 27-1: SD1CON1: S/D CONTROL REGISTER 1

R/W-0	U-0	R/W-0	R/W-0	r-0	R/W-0	R/W-0	R/W-0
SDON	—	SDSIDL	SDRST	—	SDGAIN2	SDGAIN1	SDGAIN0
bit 15							bit 8

R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
DITHER1	DITHER0	—	VOSCAL	—	SDREFN	SDREFP	PWRLVL
bit 7							bit 0

Figura 75. Registro de configuración 1 de 3 del convertidor Sigma-Delta. Fuente: Datasheet PIC24FJ128GC06.

- SDON: activación del módulo S/D
  - 1 = activado
  - 0 = desactivado
- SDSIDL: S/D estado en modo Idle
  - 1 = paralizado
  - 0 = continuado
- SDRST: reinicio
  - 1 = reinicia el módulo
  - 0 = modo normal
- SDGAIN: ganancia
  - 101 = 32
  - 100 = 16
  - 011 = 8
  - 010 = 4
  - 001 = 2
  - 000 = 1
- DITHER: Modo Dither
  - 11 = Dither elevado
  - 10 = Dither medio
  - 01 = Dither Bajo
  - 00 = No Dither
- VOSCAL: empleo para medir el offset interno
  - 1 = configurado para medir el error offset
  - 0 = modo normal
- SDREFN: referencia negativa
  - 1 = externa ($SVref_{pin}$)
  - 0 = interna (SV_{SS})
- SDREFP: referencia positiva
  - 1 = externa ($SVref_{pin}$)
  - 0 = interna (SV_{DD})
- PWRLVL: amplificador del ancho de banda
  - 1 = doble de ancho de banda
  - 0 = ancho de banda normal

Tabla 15. Parámetros del registro 1 de 3 del convertidor Sigma-Delta. Fuente: elaboración propia.

Parámetro	Valor asignado	Equivalente librería CCS
SDON	1	SDADC_ENABLED
SDSIDL	0	Default
SDRST	0	Default
SDGAIN	000	SDADC_GAIN_1
DITHER	11	SDADC_DITHER_HIGH
VOSCAL	0	Default
SDREFN	0	SDADC_SVREF_SVSS
SDREFP	1	
PWRLVL	0	SDADC_BW_NORMAL

REGISTER 27-2: SD1CON2: S/D CONTROL REGISTER 2

R/W-0	R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0
CHOP1	CHOP0	SDINT1	SDINT0	—	—	SDWM1	SDWM0
bit 15							bit 8

U-0	U-0	U-0	R/W-0	R/W-0	U-0	U-0	HS/C-0
—	—	—	RNDRES1	RNDRES0	—	—	SDRDY
bit 7							bit 0

Figura 76. Registro de configuración 2 de 3 del convertidor Sigma-Delta. Fuente: Datasheet PIC24FJ128GC06.

- CHOP: activación CHOP
  - 11 = activo
  - 00 = desactivo
- SDINT: generación de interrupción
  - 11 = cada ciclo de reloj
  - 10 = cada 5 ciclos de reloj
  - 01 = cuando el resultado aumenta
  - 00 = cuando el resultado disminuye
- SDWM: actualización de resultado (SD1RESH/SD1RESL)
  - 10 = no actualizados
  - 01 = cada interrupción
  - 00 = cada interrupción si SDRDY = 0
- RNDRES: redondeo
  - 11 = redondeo a 8 bits
  - 10 = redondeo a 16 bits
  - 01 = redondeo a 24 bits
  - 00 = sin redondeo
- SDRDY: bit de estado del filtro
  - 1 = terminado
  - 0 = sin terminar

Tabla 16. Parámetros del registro 2 de 3 del convertidor Sigma-Delta. Fuente: elaboración propia.

Parámetro	Valor asignado	Equivalente librería CCS
CHOP	11	SDADC_CHOPPING_ENABLED
SDINT	11	SDADC_INT_EVERT_SAMPLE
SDWM	01	SDADC_RES_UPDATE_EVERY_INT
RNDRES	10	SDADC_ROUND_16_BITS
SDRDY	0	Default

REGISTER 27-3: SD1CON3: S/D CONTROL REGISTER 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SDDIV2(1)	SDDIV1(1)	SDDIV0(1)	SDOSR2	SDOSR1	SDOSR0	SDCS1	SDCS0
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	SDCH2	SDCH1	SDCH0
bit 7							bit 0

Figura 77. Registro de configuración 3 de 3 del convertidor Sigma-Delta. Fuente: Datasheet PIC24FJ128GC06.

- SDDIV: divisor del reloj
  - 110 = 64
  - 101 = 32
  - 100 = 16
  - 011 = 8
  - 010 = 4
  - 001 = 2
  - 000 = 1
- SDOSR: ratio de sobremuestreo
  - 110 = 16
  - 101 = 32
  - 100 = 64
  - 011 = 128
  - 010 = 256
  - 001 = 512
  - 000 = 1024
- SDCS: selección de la fuente del reloj
  - 10 = Reloj primario
  - 01 = FRC (8 MHz)
  - 00 = Reloj del sistema
- SDCH : selección de canales
  - 011 = V_{REF}
  - 010 = CH1SE/SVSS
  - 001 = CH1+/CH1-
  - 000 = CH0+/CH0-

Tabla 17. Parámetros del registro 3 de 3 del convertidor Sigma-Delta. Fuente: elaboración propia.

Parámetro	Valor asignado	Equivalente librería CCS
DDIV	001	SDADC_CLOCK_DIV_2
SDOSR	000	SDADC_OSR_1024
SDCS	01	SDADC_CLK_FRC
SDCH	000	Default

Una vez configurado el convertidor Sigma-Delta, es necesario también configurar la referencia de voltaje externa seleccionada. Para el convertidor Sigma-delta se ha seleccionada la referencia BGBUF2

REGISTER 25-2: BUFCONx: BAND GAP BUFFERS 1 AND 2 CONTROL REGISTERS

R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	U-0	U-0
BUFEN	—	BUFSIDL	BUFSLP	—	—	—	—
bit 15							bit 8

R/W-0	R/W-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0
BUFOE	BUFSTBY	—	—	—	—	BUFREF1(1)	BUFREF0(1)
bit 7							bit 0

Figura 78. Registro de configuración de referencias internas. Fuente: Datasheet PIC24FJ128GC06.

- BUFEN: activación:
  - 1 = activado
  - 0= desactivado
- BUFSIDL: estado en modo Idle
  - 1 = desactivado
  - 0 = activado
- BUFSLP: estado en modo Sleep
  - 1 = desactivado
  - 0 = activado
- BUFOE: activación pin de salida
  - 1 = activado
  - 0 = desactivado
- BUFSTBY: estado en modo Standby
  - 1 = modo bajo consumo
  - 0 = modo normal
- BUFREF: nivel de tensión
  - 11 = 3.072 V
  - 10 = 2.560 V
  - 01 = 2.048 V
  - 00 = 1.2 V

Tabla 18. Parámetros del registro de referencias interna BGBUF2. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
BUFEN	1	Activo
BUFSIDL	1	Desactivado en Idle
BUFSLP	1	Desactivado en Sleep
BUFOE	1	Pin de salida activo
BUFSTBY	1	Bajo consumo en Standby
BUFREF	11	Referencia de 3.072 V

Además, de la configuración del convertidor y de la referencia de tensión, para unas lecturas precisas del sensor, es necesario su calibración. Para ello, se realiza una medición del offset, midiendo la referencia negativa (VOSCAL=1)

$$Offset\ (bits) = LecturaADC(VOSCAL = 1) \qquad \text{Ec.29}$$

Luego se reajusta el rango de 16 bits la referencia positiva (SDCH=011).

$$ValorMAX(bits) = LecturaADC(SDCH = 011) - Offset \qquad \text{Ec.30}$$

Por lo tanto, las lecturas se corrigen de acuerdo con la siguiente expresión:

$$Lectura.corregida(16\ bits) = (LecturaADC - Offset) \cdot \frac{0x7FFF}{ValorMAX} \qquad \text{Ec.31}$$

$$Lectura.corregida(V) = (LecturaADC - Offset) \cdot \frac{3.072\ V}{ValorMAX} \qquad \text{Ec.32}$$

8.5.6.ADC pipeline

Por otra parte, el convertidor pipeline que se emplea para la lectura del nivel de batería requiere de la siguiente configuración.

REGISTER 26-1: ADCON1: A/D CONTROL REGISTER 1

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADON	—	ADSIDL	ADSLP	FORM3	FORM2	FORM1	FORM0
bit 15							bit 8

R/W-0	R/W-0	U-0	U-0	U-0	U-0	U-0	R/W-0
PUMPEN	ADCAL(2)	—	—	—	—	—	PWRLVL
bit 7							bit 0

Figura 79. Registro de configuración 1 de 2 del convertidor pipeline. Fuente: Datasheet PIC24FJ128GC06.

- ADON: activación
  - 1 = activado
  - 0 = desactivado
- ADSIDL: estado en modo Idle
  - 1 = desactivado
  - 0 = activado
- ADSLP: estado en modo Sleep
  - 1 = activo
  - 0 = desactivo
- FORM: formato de datos de salida
  - 0111 = fracción con signo (sddd dddd dddd 0000)
  - 0110 = fracción (dddd dddd dddd 0000)
  - 0101 = entero con signo (ssss sddd dddd dddd)
- PUMPEN: reducción de impedancia (carga del interruptor)
  - 1 = activo
  - 0 = desactivo
- ADCAL: calibración interna
  - 1 = inicia calibración
  - 0 = modo normal
- PWRLVL: power mode
  - 1 = Full-Power (1-10 MHz)
  - 0= Low-Power (1-2.5 MHz)

- 0100 = entero (0000 dddd dddd dddd)
- 0011 = fracción con signo (sddd dddd dddd 0000)
- 0010 = fracción (dddd dddd dddd 0000)
- 0001 = entero con signo (ssss sddd dddd dddd)
- 0000 = entero (0000 dddd dddd dddd)

Tabla 19. Parámetros del registro 1 de 2 del convertidor pipeline. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
ADON	1	Activación
ADSIDL	0	Continua en Idle
ADSLP	0	Paraliza en Sleep
FORM	0000	Entero (0000 dddd dddd dddd)
PUMPEN	0	Carga del interruptor desactivada
ADCAL	0	Calibración desactivada
PWRLVL	0	Bajo consumo

REGISTER 26-2: ADCON2: A/D CONTROL REGISTER 2

R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0	r-1	r-1
PVCFG1	PVCFG0	—	NVCFG0	—	BUFORG	—	—
bit 15							bit 8

r-0	r-0	U-0	U-0	U-0	U-0	R/W-0	r-0
—	—	—	—	—	—	RFPUMP	—
bit 7							bit 0

Figura 80. Registro de configuración 2 de 2 del convertidor pipeline. Fuente: Datasheet PIC24FJ128GC06.

- PVCFG: voltaje de referencia positiva
  - 10 = ref. interna BGBUF1
  - 01 = referencia externa V_{REF+}
  - 00 = referencia AV_{DD}
- NVCFG0: voltaje de referencia negativa
  - 1 = referencia externa V_{REF-}
  - 0= referencia AV_{SS}
- BUFORG: orden del buffer de resultados generados
  - 1 = indexados por el número de canal
  - 0 = indexados mediante un buffer FIFO
- RFPUMP
  - 1 = optimización para referencias bajas (< 0.65 ·AV_{DD})
  - 0= modo normal

Tabla 20. Parámetros del registro 2 de 2 del convertidor pipeline. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
PVCFG	10	Referencia interna BGBUF1
NVDFG0	0	Referencia AV_{DD}
BUFORG	0	Orden de resultados FIFO
RFPUMP	0	Modo normal

Una vez configurado el convertidor PipeLine, al igual que con el Sigma-Delta es necesario configurar la referencia de voltaje externa seleccionada. Para el convertidor Pipeline se ha seleccionada la referencia BGBUF1, el registro de configuración sigue el mismo formato que el de la referencia BGBUF2 y se ha configurado de la siguiente forma:

Tabla 21. Parámetros del registro de referencias interna BGBUF1. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
BUFEN	1	Activo
BUFSIDL	1	Desactivado en Idle
BUFSLP	1	Desactivado en Sleep
BUFOE	0	Pin de salida desactivado
BUFSTBY	1	Bajo consumo en Standby
BUFREF	11	Referencia de 3.072 V

8.5.7.Modo Deep Sleep

El modo Deep Sleep se emplea para reducir el consumo cuando el microcontrolador no debe realizar ninguna tarea, únicamente esperar a la siguiente interrupción programada. Durante el Deep Sleep:

- El reloj del sistema es apagado
- El consumo de corriente se reduce al mínimo
- Los entradas y salidas son congeladas
- Únicamente quedan activos los periféricos RTCC y WDT si son previamente activados

REGISTER 10-1: DSCON: DEEP SLEEP CONTROL REGISTER(1)

R/W-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
DSEN	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	r-0	R/W-0	R/C-0, HS
—	—	—	—	—	—	DSBOR(2)	RELEASE
bit 7							bit 0

Figura 81. Registro de configuración 1 de 2 del modo Deep Sleep. Fuente: Datasheet PIC24FJ128GC06.

- DSEN: activación
  - 1 = activado
  - 0 = desactivado
- DSBOR: Evento BOR
  - 1 = DSBOR estaba activo y se generó un evento BOR durante el Deep Sleep
  - 0 = no se generó evento
- RELEASE: estado de los pines I/O
  - 1 = mantiene sus estados anteriores
  - 0 = libera los pines de E/S

Tabla 22. Parámetros del registro de configuración del modo Deep Sleep. Fuente: elaboración propia.

Parámetro	Valor asignado	Significado
DSEN	1	Activo
RELEASE	1	Mantiene los pines E/S

REGISTER 10-2: DSWAKE: DEEP SLEEP WAKE-UP SOURCE REGISTER(1)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0, HS
—	—	—	—	—	—	—	DSINT0
bit 15							bit 8

R/W-0, HS	U-0	U-0	R/W-0, HS	R/W-0, HS	R/W-0, HS	U-0	U-0
DSFLT	—	—	DSWDT	DSRTCC	DSMCLR	—	—
bit 7							bit 0

Figura 82. Registro de configuración 2 de 2 del modo Deep Sleep. Fuente: Datasheet PIC24FJ128GC06.

- DSINT0: evento generado por interrupción en INT0
- DSFLT: evento generado por fallo de configuración del Deep Sleep
- DSWDT: evento generado por el WachDog Timer
- DSRTCC: evento generado por el RTCC
- DSMCLR: evento generado por el pin MCLR

8.5.8. Motivo del reinicio

El microcontrolador debe saber cuál ha sido el motivo de su reinicio, ya que dependiendo de cual haya sido se ejecutará unas funciones u otras. El modo de identificar el motivo es identificándolo en el registro RCON (Reset Control Register)

REGISTER 7-1: RCON: RESET CONTROL REGISTER

R/W-0	R/W-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0
TRAPR(1)	IOPUWR(1)	—	RETEN(2)	—	DPSLP(1)	CM(1)	PMSLP(3)
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1
EXTR(1)	SWR(1)	SWDTEN(4)	WDTO(1)	SLEEP(1)	IDLE(1)	BOR(1)	POR(1)
bit 7							bit 0

Figura 83. Registro de identificación del reinicio del dispositivo. Fuente: Datasheet PIC24FJ128GC06.

- TRAPR: reinicio generado por un conflicto TRAP
- IOPUWR: reinicio generado por fallo en el código
- RETEN: activación del modo de retención a 1.2 V
- DPSLP: el dispositivo se encontraba en Deep Sleep
- CM: reinicio generado por un error en asignación de registro
- PMSLP: alimentación de la memoria de programa durante el modo Sleep
- EXTR: reinicio provocado por pin MCLR
- SWR: reinicio provocado por instrucción del software
- SWDTEN: activación del WDT
- WDTO: reinicio provocado por WDT
- SLEEP: el dispositivo se encontraba en modo Sleep
- IDLE: el dispositivo se encontraba en modo IDLE
- BOR: reinicio provocado por evento Brown-out, caída de tensión.
- POR: reinicio provocado por un encendido

Para el código es necesario diferenciar el reinicio por la instrucción de software (SWR), el de encendido (POR) y el externo (EXTR)

8.5.1. Escribir/leer en memoria de programa

Para que los valores de configuración queden escritos de forma permanente en el dispositivo, estos se almacenan en la memoria de programa. La memoria de programa está organizada en filas de 64 instrucciones (192 bytes). El modo de acceder a la misma es mediante bloques de 8 filas (512 instrucciones). Cada instrucción corresponden con 24 bits, no obstante, esta se almacena en 4 bytes (múltiplo de 16 bits) por lo que el cuarto byte debe ser siempre 0x00. Por motivos de seguridad, antes de escribir cualquier información en estos registros es necesario inicializarlos a 0xFFFFFF, cualquier bit que no sea previamente puesto a 1 permanecerá como 0 independientemente del valor de escritura. Para inicializarlos a 0xFFFFFF es necesario realizar un borrado de la memoria.

Ejemplo de código:

erase_program_memory(orgUmb);

write_program_memory(orgUmb, config_flash, 24);

En primer lugar, se reinicia el bloque formado por la dirección orgUmb, luego se escriben 24 bytes en el bloque orgUmb desde la posición inicial con los valores del array config_flash, teniendo en cuenta que cada cuatro valores es 0x00.

9. PROGRAMACIÓN APP EN ANDROID

Como se ha descrito anteriormente, la configuración y puesta en marcha se realiza mediante una conexión Bluetooth a través de un teléfono móvil. Para ello, se ha implementado una aplicación para Android con el programa Android Studio.

9.1. Android Studio

Android Studio es el entorno de desarrollo (IDE) oficial para la creación de aplicaciones para dispositivos Android. Su programación está basada en el software IntelliJ IDEA de JetBrains, se trata de un programa gratuito disponible para plataformas Windows, Linux y MacOS. (15)

El desarrollo de una aplicación parte de la creación de un nuevo proyecto, cada proyecto contiene uno o más módulos con archivos de código fuente y archivos de recursos. Disponer de varios módulos permite crear varias versiones de la misma aplicación (para dispositivo móvil, para TV…)

Cada módulo en Android Studio está compuesto por los siguientes archivos:

- **Manifests:** se trata de un descriptor de la aplicación. Se define su nombre, el icono de las actividades, los servicios, así como los permisos que requiere la aplicación.
- **Java:** código fuente en Java, incluye tanto el código de la actividad inicial como el de las actividades secundarias.
- **Res:** contiene los recursos que emplea la aplicación, estos pueden clasificarse en distintos tipos:
    - **Drawable:** contiene las imágenes incrustadas en la aplicación
    - **Mipmap:** contiene el icono de la aplicación
    - **Layout:** contiene ficheros XML con vistas de la aplicación, representan las distintas pantallas de las que dispondrá la interfaz de la aplicación.
    - **Menu:** contiene ficheros XML con los menús incluidos en cada aplicación.
    - **Values:** contiene ficheros XML que describen las características de los elementos incluidos en el Layout.
    - **Otros ficheros:** puede requerir ficheros adicionales en formato XML o en otros formatos.
- **Gradle Scripts:** ficheros para la construcción del módulo, permiten su compilación y construcción de la aplicación.

9.2. Programación de la aplicación

En primer lugar, se han definido las distintas funcionalidades de la aplicación, así como la interacción con el usuario. La aplicación está compuesta por dos actividades, en la primera de ellas se realiza la conexión bluetooth y la segunda permite la interacción con el dispositivo:

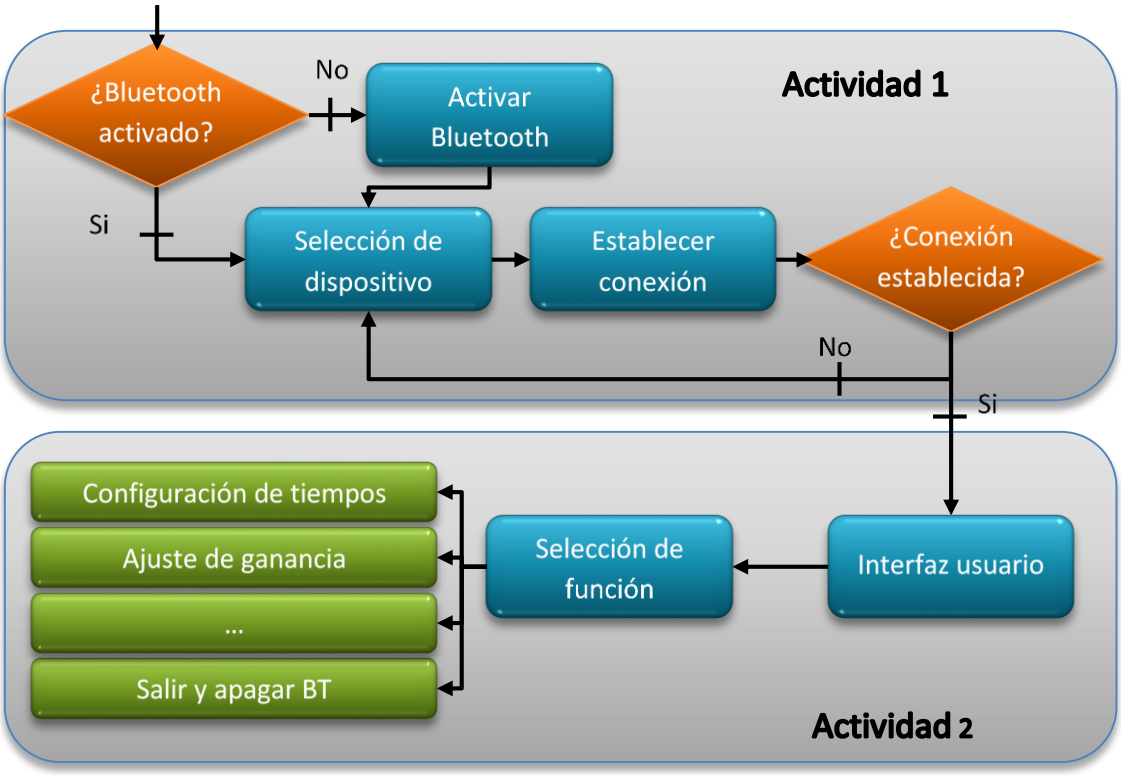


Figura 84. Flujograma del funcionamiento de la aplicación. Fuente: elaboración propia.

Una vez establecida la conexión con el dispositivo mediante una interfaz gráfica se podrán seleccionar las distintas funciones descritas en el apartado 8.3 MODO CONFIGURACIÓN.

9.2.1. Desarrollo de la interfaz gráfica

La aplicación dispone de dos pantallas, una correspondiente a cada una de las actividades. En la primera actividad cuya función es establecer la conexión bluetooth, aparece un listado de los dispositivos disponibles. Para seleccionar el dispositivo al que se quiere conectar, es necesario pulsar sobre la misma línea.

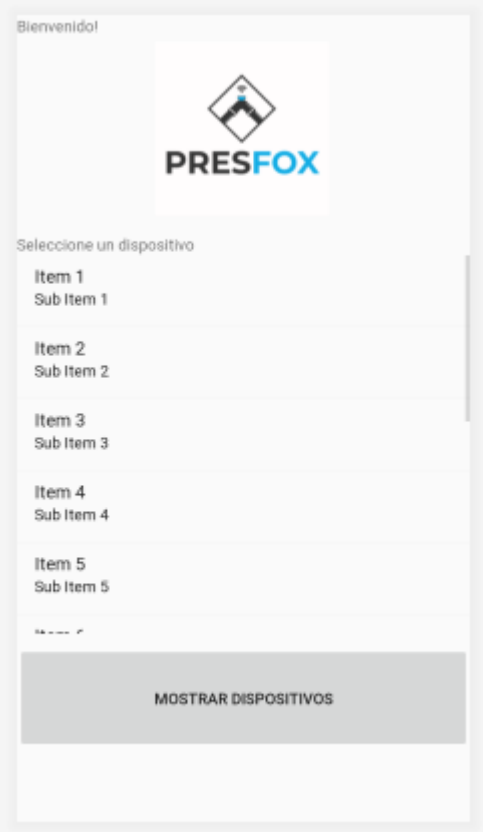


Figura 85. Pantalla actividad inicial: selección de dispositivo. Fuente: elaboración propia.

Al seleccionarse un dispositivo, la dirección MAC es transferida a la siguiente actividad, donde se realiza la conexión con el dispositivo. En esta actividad se muestra una serie de funciones. Al conectarse la aplicación carga en sus opciones la configuración guardada del sistema.

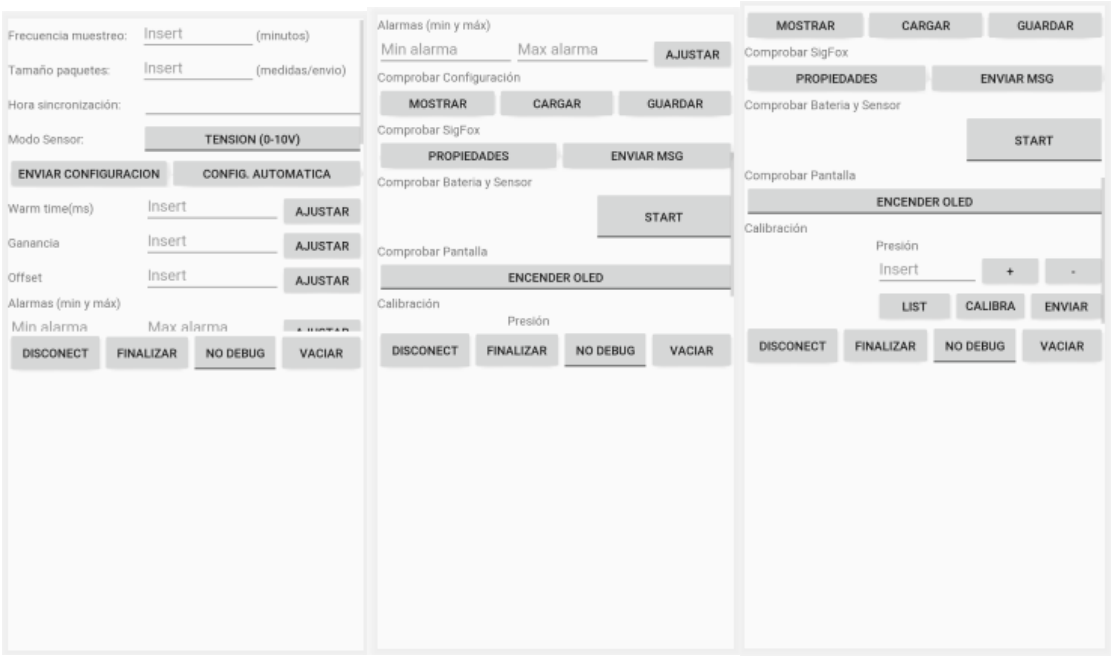


Figura 86. Pantalla actividad principal: pantalla de configuración. Fuente: elaboración propia.

Las funciones han sido agrupadas por categorías, se muestran las funciones de configuración de tiempos, las configuraciones del sensor, las comprobaciones del módulo SigFox, las comprobaciones del sensor y del display y finalmente el apartado de calibración. Cada función está implementada por un pulsador y la entrada de valores numéricos que requiere. En la parte inferior de la pantalla se encuentra un display que muestra la interacción con el dispositivo para su depuración.

Las pantallas han sido definidas mediante ficheros XML. Las pantallas se crean mediante la inserción de distintos elementos (botones, entradas de texto, organizaciones…), cada elemento posee unas características definidas, así como un identificador y una posición jerárquica en la pantalla. Para la creación de las pantallas anteriores se han empleado los siguientes elementos:

- **RelativeLayout:** se trata de una agrupación de vistas, que permite la posición de cada uno de los elementos de acuerdo a la posición relativa a otro elemento (a la izquierda, a la derecha, arriba o abajo)
- **LinearLayout (vertical/horizontal):** se trata de una agrupación de vistas, similar al RelativeLayout con la diferencia de que permite una única dirección, vertical u horizontal.

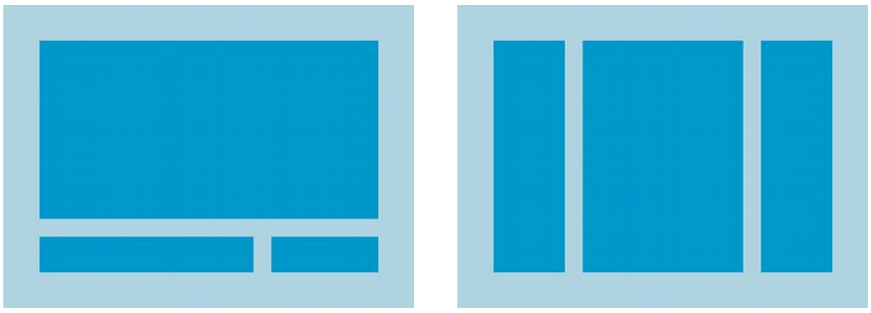


Figura 87. Esquema RelativeLayout (izq.) y LineaLayout (drcha.). Fuente: (15)

- **ScrollView:** se trata de una agrupación de vistas, que permite su deslizamiento, para insertar diversas vistas. Dentro de un Scrollview es necesario que estas sean agrupadas previamente en un LinearLayot (vertical).
- **TextView:** permite mostrar texto al usuario, este puede ser estático o puede ser modificado por el código de la aplicación.
- **EditText:** permite la introducción de texto por parte del usuario, este puede ser limitado a determinados formatos, por ejemplo, a un número decimal (numberDecimal)
- **Button:** se trata de un elemento que puede ser pulsado para conllevar una acción.
- **Tooglebutton:** se trata de un elemento con dos estados, cuyo paso de un estado a otro se realiza mediante una pulsación.
- **ListView:** se trata de un elemento que permite mostrar una lista de opciones y la selección de una de ellas pulsado sobre la opción elegida.

Para cada uno de los elementos se deben definir sus características (color, tamaño, texto…). Además, debe ser insertado de manera jerárquica en la pantalla deben ser identificados mediante un identificador único que permite su utilización el código de la aplicación.

A continuación, se puede ver, un ejemplo de inserción jerárquica donde se han insertado dos *Buttons* al mismo nivel dentro de un *Horizontal LinearLayout*.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="horizontal">

    <Button
        android:id="@+id/config_menu"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CONFIGURACION" />

    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="CALIBRACION" />

</LinearLayout>
```

Figura 88. Resultado del código mostrado anteriormente. Fuente: elaboración propia.

9.2.2. Desarrollo del código fuente

Cada una de las pantallas va asociada a una actividad, cada actividad posee un código fuente (fichero .java) que permite la ejecución de las funciones de la aplicación y controla la interacción con el usuario.

En cada código fuente se pueden diferenciar distintas secciones:

- **Importación de librerías:** para la utilización de distintas funciones, elementos o periféricos del dispositivo móvil es necesario importar las librerías asociadas para su compilación. Por ejemplo, para utilizar la conexión bluetooth del dispositivo es necesario importar las siguientes librerías:

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
```

- **Definición de las variables:** para el empleo de los distintos elementos insertados en la interfaz gráfica, es necesario definir una variable para cada uno de ellos, también se deben definir las variables auxiliares globales requeridas. Por ejemplo:

```
ToggleButton ModeSensor,btOLED,btmeasure;
EditText chooseTime;
```

- **Asignación de variables:** cada una de las variables definidas debe ser asignada a un elemento de la pantalla. Por ejemplo:

```
btSend= (Button)findViewById(R.id.btSend) ;
Freq= (EditText) findViewById(R.id.id_Freq);
```

- **Definición de Listeners:** el lenguaje Java está basado en la programación de objetos basados en eventos. Los eventos son las acciones que el usuario puede realizar con los distintos objetos (Por ejemplo: pulsar un botón). Los *Listeners* se encargan de controlar los eventos, éstos ejecutan distintas funciones cuando se produce el evento que están esperando. Por ejemplo, el siguiente *Listener* ejecuta la función Diconnect() cuando se pulsa el botón btnDis.

```
btnDis.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        Disconnect(); //close connection
    }
});
```

- **Definición de las funciones:** incluye el código referente a cada función llamada. Puede ejecutar distintas funciones. Por ejemplo, la siguiente función cierra el Socket abierto para la conexión bluetooth:

```
private void Disconnect(){
    if (btSocket!=null) { //If the btSocket is busy
        try{
            btSocket.close(); //close connection
        }catch (IOException e){
            msg("Error");}
    }
    finish(); //return to the first layout
}
```

Además de las funciones anteriores, la conexión Bluetooth requiere de un tratamiento especial.

- **myBluetooth=BluetoothAdapter.**`getDefaultAdapter()`: Obtiene un identificador del adaptador Bluetooth local predeterminado.
- **myBluetooth.**`isEnabled()`: permite consultar si el adaptador Bluetooth está activado.
- **myBluetooth.**`getRemoteDevice(address)`: permite conectar el adaptador Bluetooth al dispositivo cuya dirección MAC es address.
- **btSocket=** `dispositivo.createInsecureRfcommSocketToServiceRecord`**(myUUID):** crea una conexión socket al puerto serial virtual (SPP Bluetooth).
- **btSocket.**`connect()` : activa la conexión.
- **socket.**`getInputStream()`: permite la recepción de datos desde la conexión socket.
- **socket.**`getOutputStream.write()`: permite el envío de datos a través de la conexión socket.

Para el envío de los datos, como pueden ser la configuración de tiempos, la ganancia, etc. Se debe cambiar al formato previamente definido en el código del microcontrolador. Por ejemplo, la ganancia es enviada mediante un *float* de 32 bits, dividido en 4 palabras de 8 bits.

```
private void SendGain(){
    if (btSocket!=null){
        try{
            btSocket.getOutputStream().write("G".getBytes());
            try {
                Thread.sleep(100);
            }catch(InterruptedException e){
                msg("Error");
            }
            float f=Float.parseFloat(gain.getText().toString());
            int bits=Float.floatToIntBits(f);
            byte[] bytes =new byte[4];
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);
        }
        catch (IOException e)
        {
```

```
                msg("Error");
            }
        }
    }
```

Por otra parte, los mensajes recibidos desde el microcontrolador han sido formateados de acuerdo con los siguientes puntos:

- Cada mensaje se envía con el carácter retorno de carro (CR, "\r") como finalizador de mensaje.
- Si el mensaje empieza por *: se trata de una lectura de presión y nivel de baterías:

```
*35.325,3.82
```

- Si el mensaje empieza por +: se trata de la lectura de la ADC para un nuevo punto de calibración:

```
+2.853
```

- Si el mensaje empieza por -: se requiere eliminar el último punto de calibración
- Si el mensaje empieza por "$": el mensaje recibido se trata de la configuración actual. Se actualizan los campos de configuración de la aplicación de acuerdo al mensaje recibido

```
$2,6,0,1.00,0.00,0.6,3,4.40,22,250
```

- Si el mensaje no empieza por los carácteres anteriores, el mensaje se muestra en el display de la aplicación

```
Nodo SigFox conectado
```

10. PRUEBAS DE VALIDACIÓN

Para las distintas pruebas se ha desarrollado una placa simulando el comportamiento del sensor de presión. Para poder apreciar la evolución diaria de la medida y facilitar su comprobación se ha implementado un sensor de temperatura. Además, el sensor de temperatura puede ser sustituido por un potenciómetro para variar la tensión en sus comprobaciones.

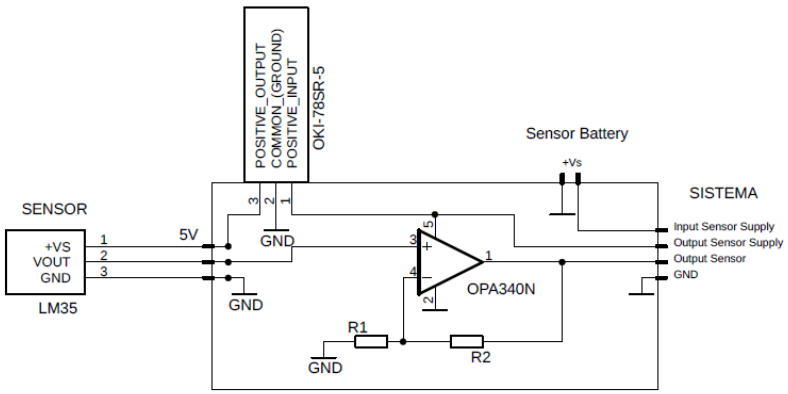


Figura 89. Esquema placa amplificadora sensor de temperatura. Fuente: elaboración propia.

10.1. Comprobación lectura ADC

Para la validación de la lectura de la ADC se ha realizado el siguiente montaje, empleando el circuito anterior con el potenciómetro, donde se han comparado las lecturas del dispositivo con las lecturas del osciloscopio y del multímetro colocado como amperímetro.

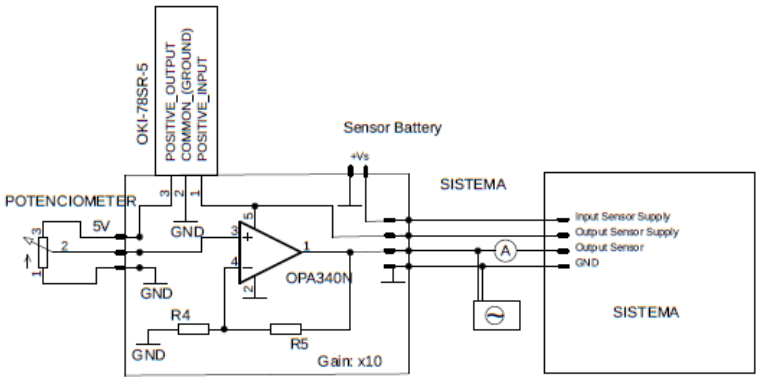


Figura 90. Esquema montaje para la comprobación de lecturas de la ADC. Fuente: elaboración propia.

La tarea ha consistido en posicionar el potenciómetro en su posición más baja, e ir subiendo hasta que se completase el rango de medida (de 0 a 10 V para el modo tensión y de 4-20 mA para el modo corriente).

Se han tomado un total de 5 medidas para cada uno de los modos, la medida de tensión del osciloscopio ha sido medida por su valor medio, la medida de corriente es la proporcionada por el multímetro configurado para un rango de 20mA y las mediciones del dispositivo han sido tomadas como la media entre 10 medidas consecutivas.

Tabla 23. Resumen de medidas para la comprobación de la lectura de la ADC. Fuente: elaboración propia.

Lectura en modo tensión		**Lectura en modo corriente**	
Dispositivo (V)	Osciloscopio (V)	Dispositivo (mA)	Amperímetro
1.254	1.253	5.345	5.35
2.674	2.675	7.325	7.32
5.643	5.645	9.256	9.26
7.854	7.858	13.568	13.57
8.052	8.058	18.268	18.28

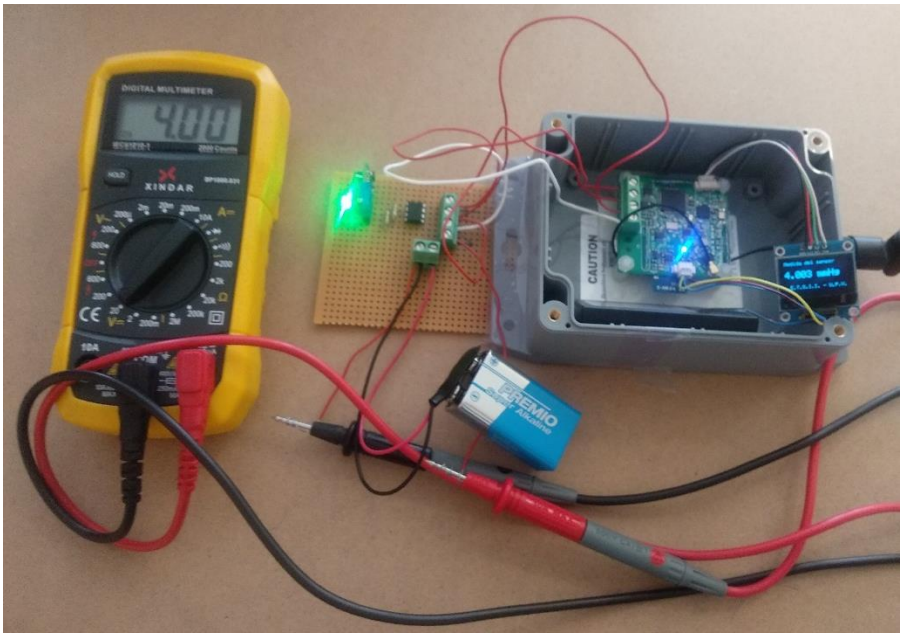


Figura 91. Ejemplo de medición realizada para la validación de las lecturas. Fuente: elaboración propia.

Por otra parte, también se ha realizado un montaje sobre un sensor de presión industrial TN 43006710, cuyas especificaciones son:

- Rango: 0-10 bar
- Salida: 4-20 mA (2-wire)
- Alimentación: 10-30 V_{DC}

La salida medida en este caso ha sido 0.628 mA que corresponde aproximadamente con 1 bar, la presión atmosférica. Tras la ejecución de las pruebas se ha podido comprobar la correcta medida del dispositivo, tanto para sensores con salida en tensión (0-10 V) como sensores con salida en corriente (4-20 mA).

10.2. Comprobación de calibración

Para la comprobación de la calibración del sensor se ha empleado el montaje anterior con el potenciómetro simulando el siguiente sensor, ganancia 1bar/V y un offset de 1 bar. El procedimiento ha sido similar al caso anterior, salvo que en este caso las medidas se han tomado para las tensiones que corresponderían a las presiones 1, 2 y 3 bares.

Figura 92. Comprobación de la correcta calibración del sensor.

10.3. Comprobación cobertura SigFox

Para la validación de la cobertura, en primer lugar, se realizaron envíos y recepciones individuales, comprobándose así la recepción y el envío de los mensajes. Para ello se configuró un Callback de respuesta con el mensaje con el mensaje: 0x00a0206419640000 (8 bytes)

Figura 93. Comprobación envío y recepción desde la aplicación móvil. Fuente: elaboración propia.

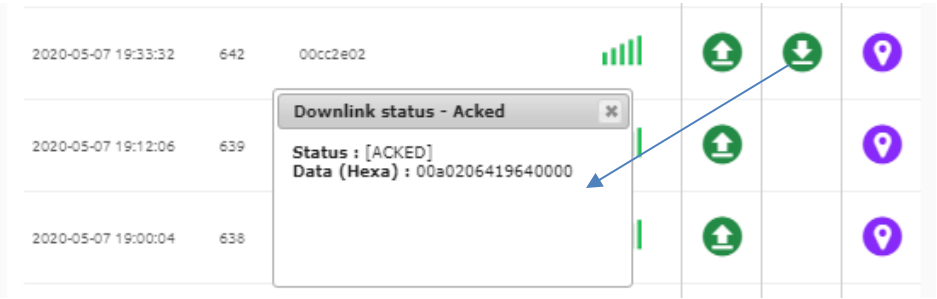


Figura 94. Comprobación envío y recepción en el servidor (back-end) de SigFox. Fuente: elaboración propia.

Posteriormente, para comprobar la fiabilidad del sistema, se ha dejado instalado el sistema con la placa del sensor de temperatura durante 2 meses. Para ello se ha programado para tomar mediciones cada 2 minutos, y realizar envíos cada 6 mediciones (12 minutos, 120 mensajes al día)

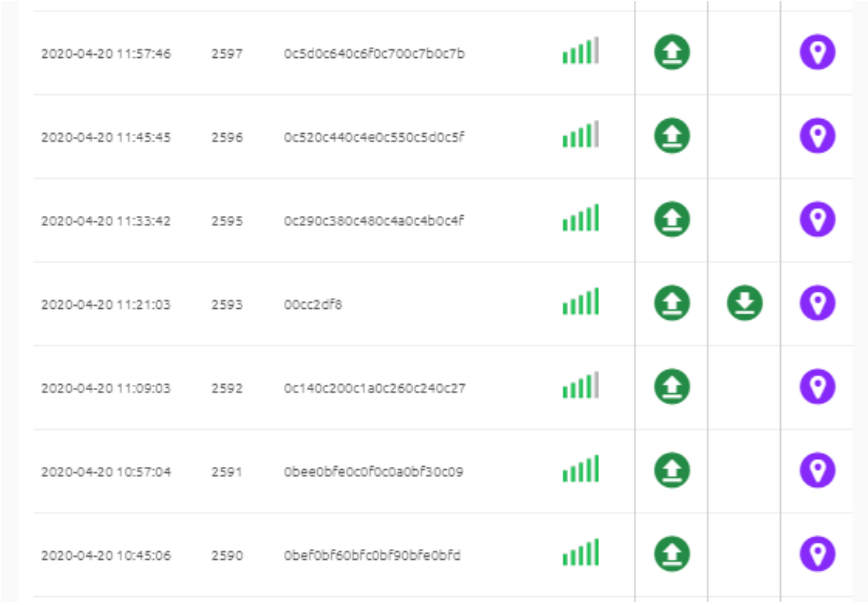


Figura 95. Ejemplo de mensajes recibidos. Fuente : elaboración propia.

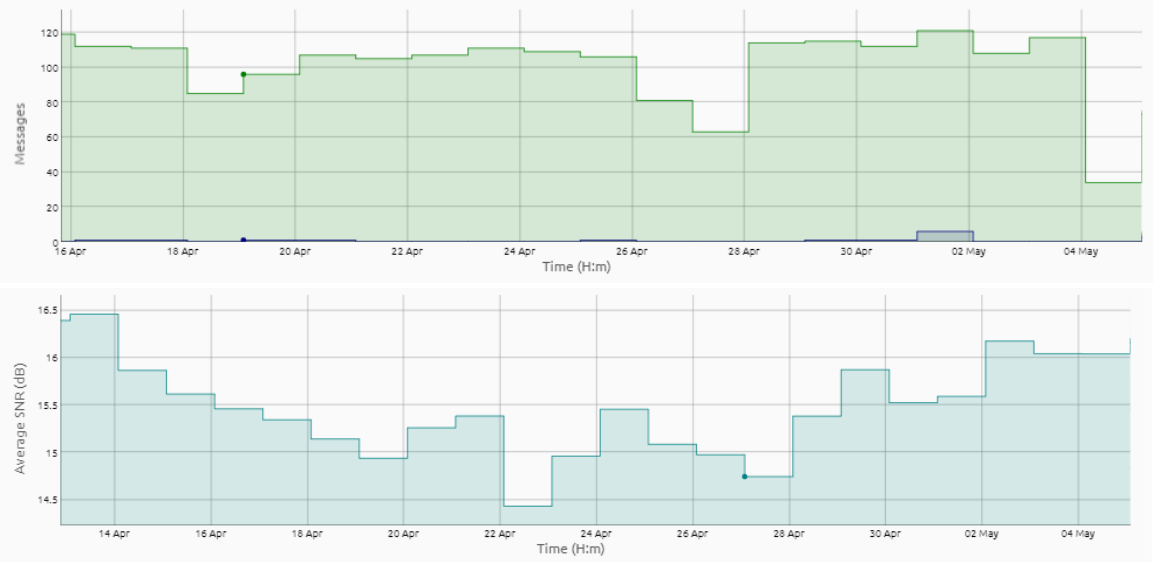


Figura 96. Histórico de recepción de mensajes. Fuente: elaboración propia.

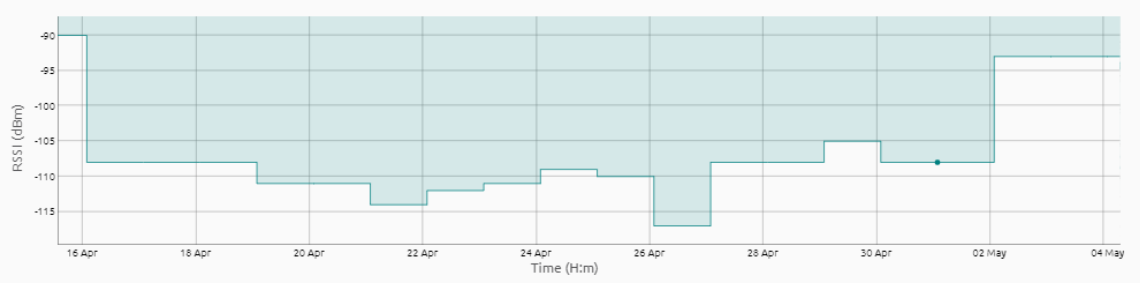


Figura 97. Nivel de señal media recibida, valor SNR y RSSI. Fuente: elaboración propia.

Durante el tiempo de funcionamiento, la cantidad de mensajes perdidos han sido aproximadamente de 5 a 10 al día con el dispositivo en el interior de la vivienda y entre 1 y 2 cuando se encontraba próximo a la ventana. Por este motivo, se recomienda la instalación de la antena en el exterior.

10.4. Comprobación de alarmas

Para la comprobación de alarmas, se empleó el sistema con el sensor de temperatura, se realizaron dos comprobaciones distintas:

- Se desconectó la alimentación del sensor para que la medida del sensor fuera 0 simulando una desconexión/rotura del sensor
- Se configuró un umbral máximo cercano a la temperatura ambiente, luego se calentó por encima de esta temperatura

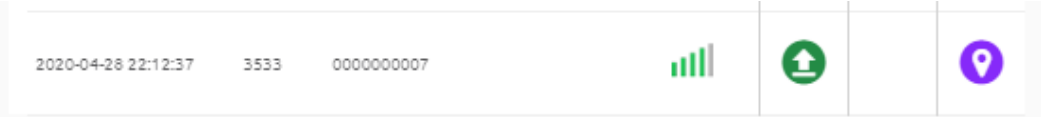


Figura 98. Ejemplo de recepción de alarma para la desconexión del sensor (lectura 0V). Fuente: elaboración propia.

10.5. Comprobación funcionamiento

Finalmente, tras comprobar cada uno de los aspectos más relevantes del sistema, se realizó una comprobación global del sistema a partir de la aplicación desarrollada:

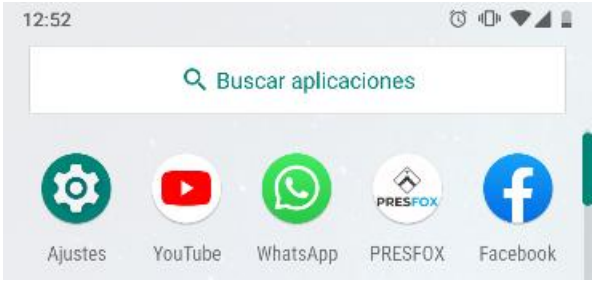


Figura 99. Apertura de la aplicación PRESFOX.

Tras abrir la aplicación, en caso de no tener activado el Bluetooth del dispositivo, mediante una ventana emergente permite su activación. Una vez dentro se pueden listar los dispositivos que han sido emparejados con el dispositivo móvil y seleccionar al que se quiere conectar (MiniFox)

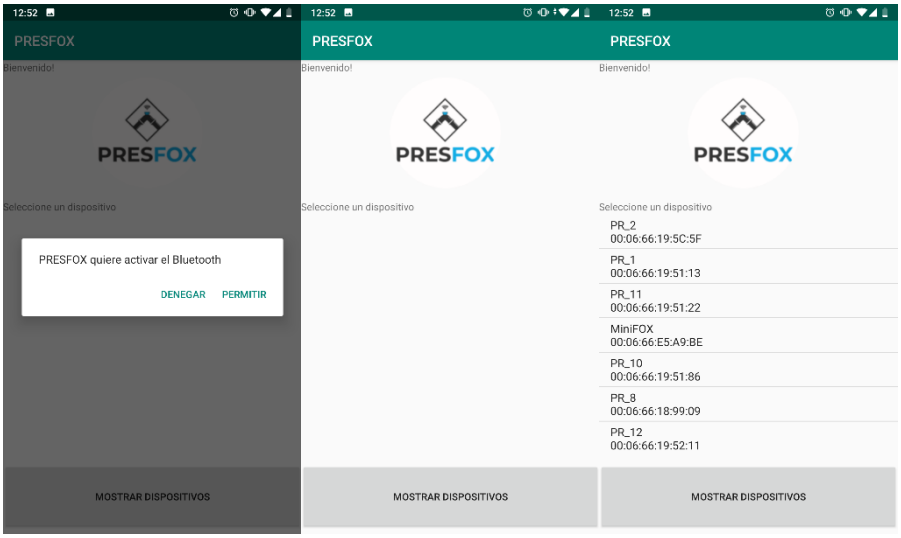


Figura 100. Pantalla inicial donde se realiza la conexión al dispositivo. Fuente: elaboración propia.

Tras seleccionar el dispositivo se realiza la vinculación, en caso de requerir un código de verificación (1234) este se introduce mediante una ventana emergente. Al realizarse la conexión se actualizan los parámetros de la aplicación a los almacenados en el dispositivo.

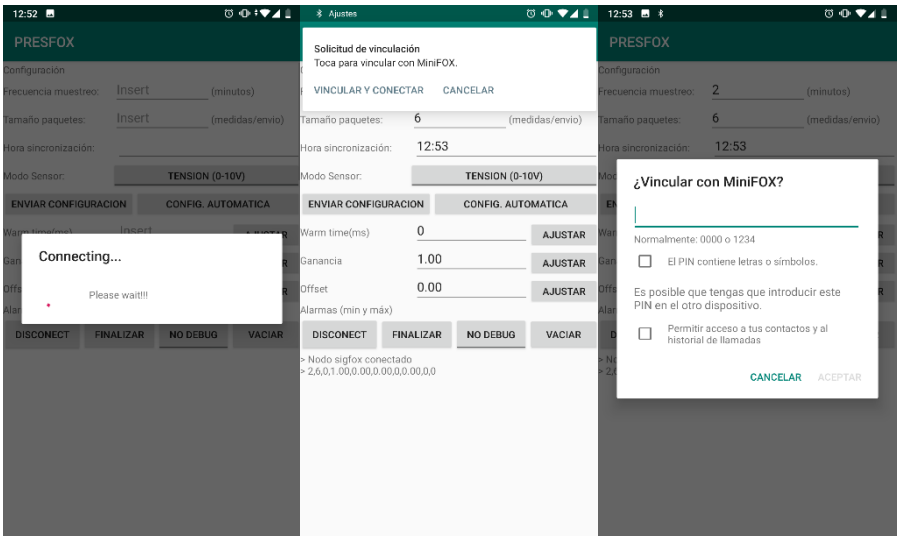


Figura 101. Vinculación al dispositivo MiniFox. Fuente: elaboración propia.

Tras la conexión se ha realizado una nueva configuración del dispositivo. Se han cambiado los parámetros de tiempo y la configuración del sensor.

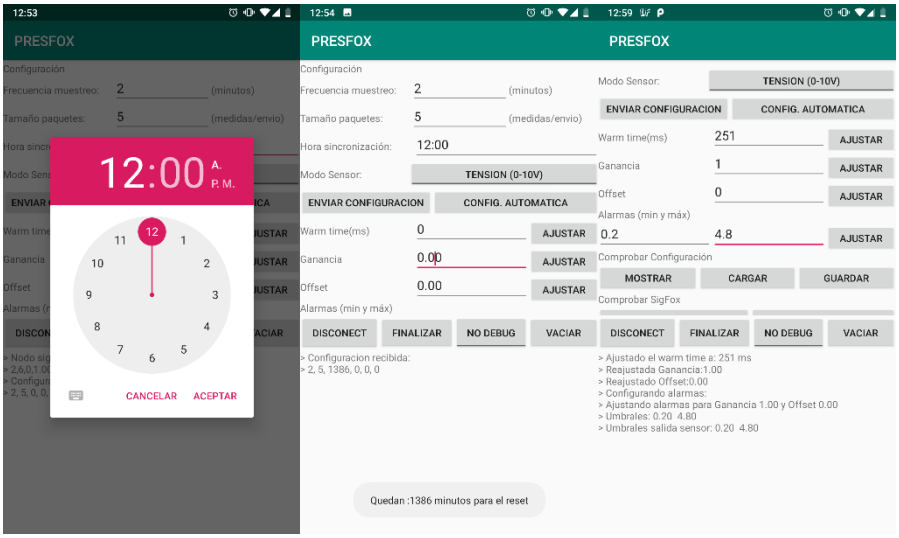


Figura 102. Ajuste de una nueva configuración al sistema. Fuente: elaboración propia.

A continuación, se ha testeado el módulo SigFox mostrando sus propiedades y realizando una prueba de cobertura. También se han testeado las lecturas de las ADCs mediante la aplicación y mediante el display.

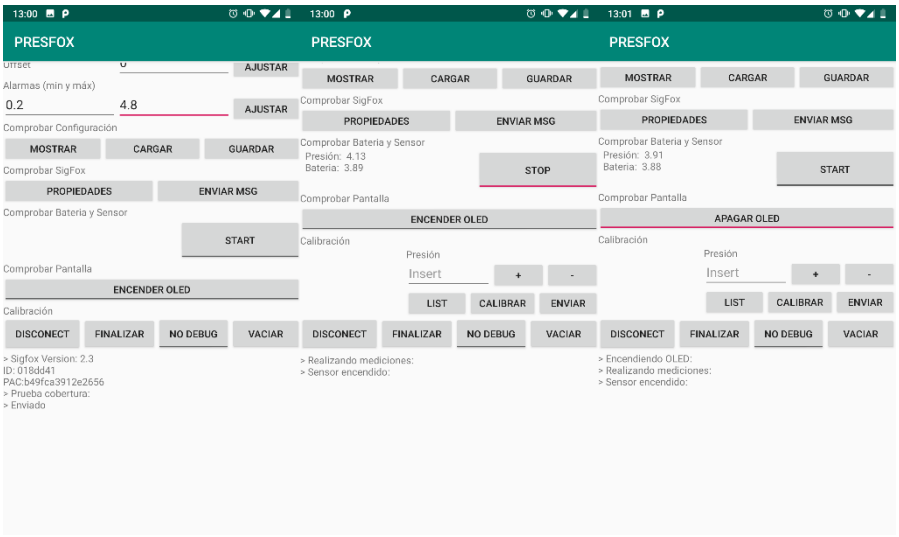


Figura 103. Comprobación del módulo SigFox, del sensor y del display. Fuente: elaboración propia.

Por último, se ha comprobado la interacción con la función calibrar, se han añadido varios puntos, se han corregido varios y finalmente se ha procedido a la calibración y al envío de los datos de calibración al servidor SigFox.

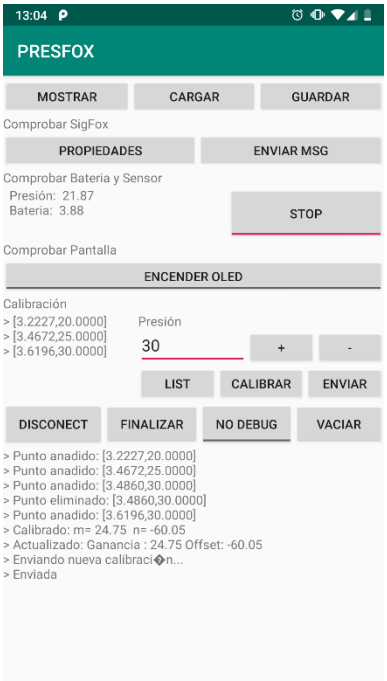


Figura 104. Proceso de calibración completa del sensor. Fuente: elaboración propia.

11. CÁLCULO DEL CONSUMO Y DIMENSIONAMIENTO DE LA BATERÍA

11.1. Cálculo del consumo

Para las mediciones del consumo se ha provisto del siguiente montaje:

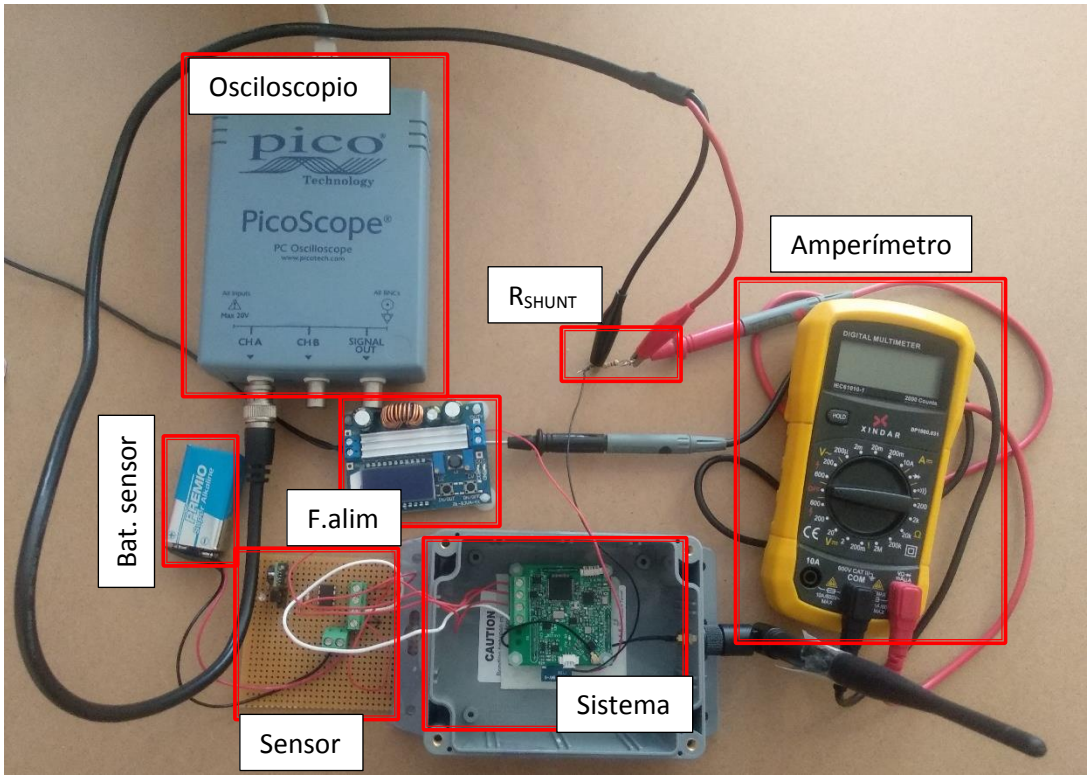


Figura 105. Montaje para la medición del consumo. Fuente: elaboración propia.

El consumo ha sido medido de dos formas, mediante una resistencia de shunt (1 Ohms) en la alimentación, midiendo la tensión caída con un osciloscopio y mediante un multímetro colocado como amperímetro.

Se tomado los seis perfiles de consumo más representativos

- Consumo en modo Sleep.
- Consumo en medición, envío y recepción.
- Consumo durante la espera de conexión Bluetooth.
- Consumo medio durante la conexión Bluetooth.
- Consumo medio durante un envío SigFox y conexión Bluetooth establecida.
- Consumo durante una recepción con el BT conectado.
- Consumo durante una recepción con el BT conectado.

Consumo en modo Sleep: para un consumo tan reducido, por debajo del mA el osciloscopio empleado PicoScope introduce más ruido en el display que la propia señal. No obstante, el valor medio si es calculado con exactitud, el multímetro tomaba una lectura de 0.052 mA equivalente a los 40.94µV medidos con el osciloscopio.

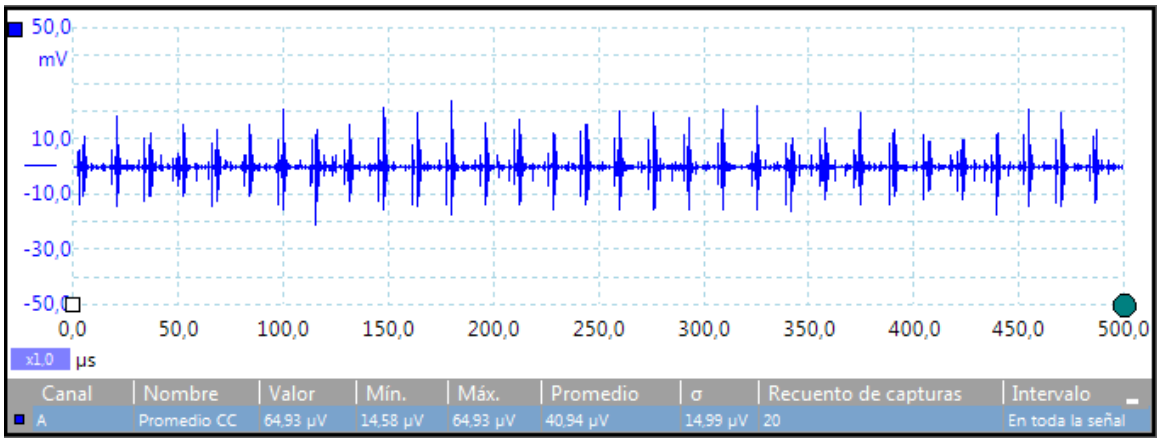


Figura 106. Consumo de corriente en Modo Deep Sleep, medido con una resistencia de shunt de 1 Ω. Fuente: elaboración propia

Consumo en medición, envío y recepción: en este caso el consumo medio oscila entre los 30-20 mA durante el envío y 24-18 mA durante la recepción.

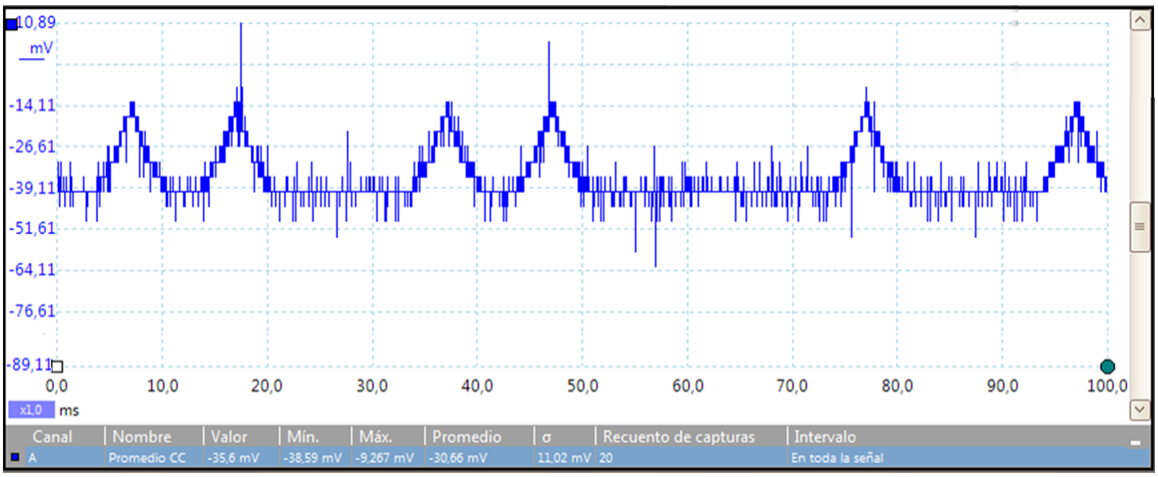


Figura 107. Consumo medio en el proceso de medición y envío. Fuente: elaboración propia

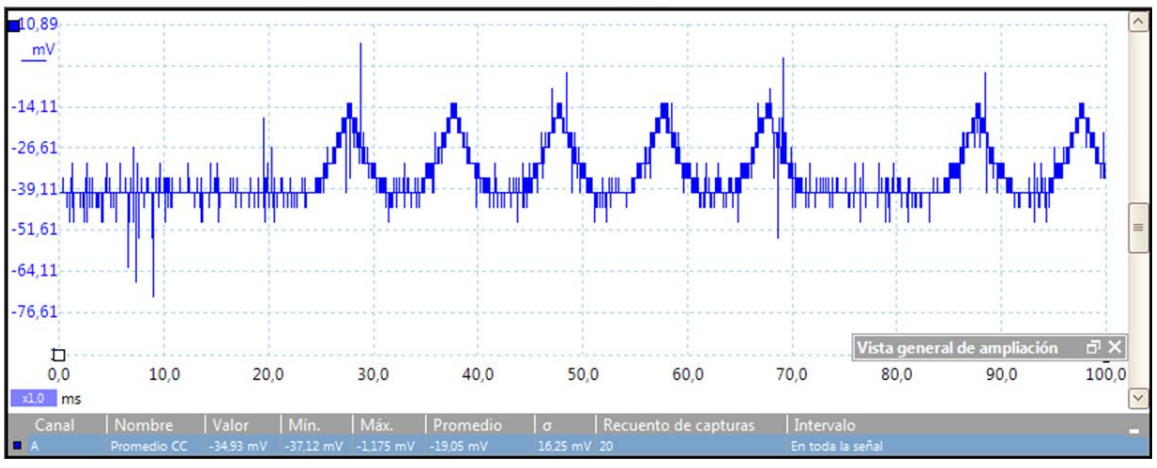


Figura 108. Consumo medio en el proceso de envío y recepción. Fuente: elaboración propia

Consumo durante la espera de conexión Bluetooth: durante la espera de conexión el consumo del sistema es de aproximadamente 25 mA.

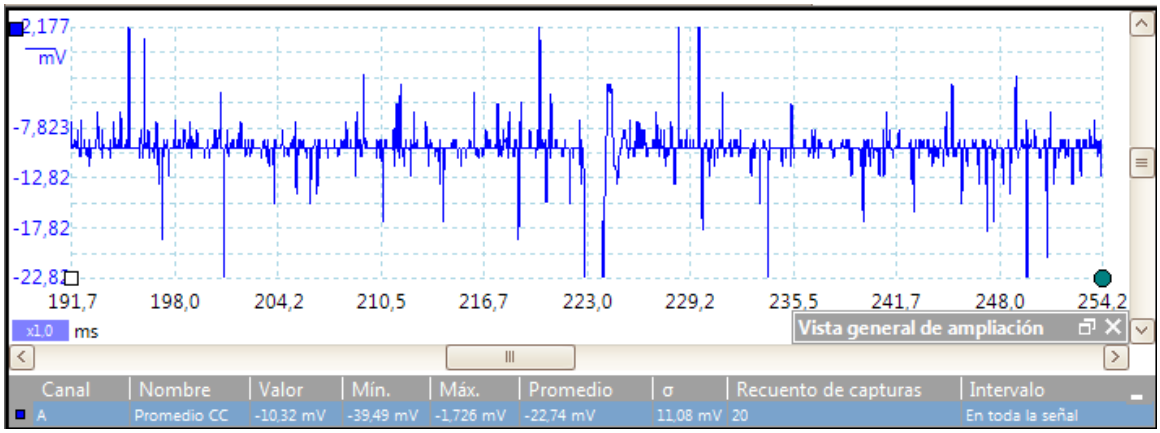


Figura 109. Consumo medio durante la espera de vinculación Bluetooth. Fuente: elaboración propia

Consumo medio durante la conexión Bluetooth: durante la conexión se alcanza un consumo medio de 35 mA.

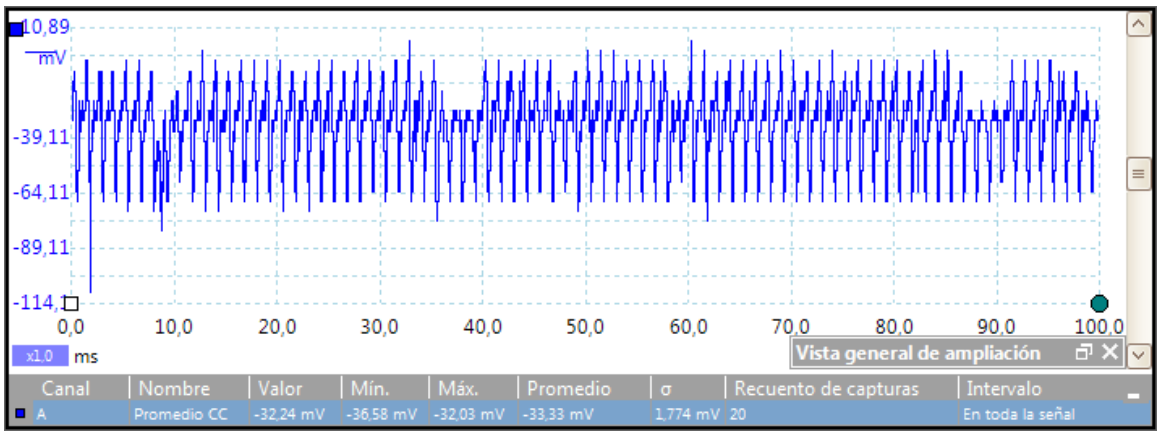


Figura 110. Consumo medio durante la vinculación Bluetooth. Fuente: elaboración propia

Consumo medio durante un envío SigFox y conexión Bluetooth establecida: durante este proceso se registra el máximo consumo entorno a los 60 mA.

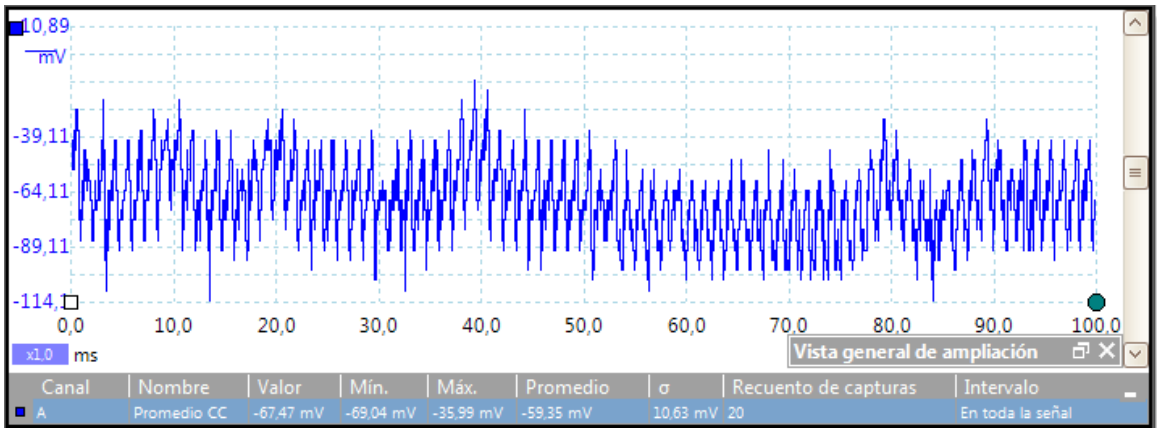


Figura 111. Consumo medio durante un envío estando de vinculado el Bluetooth. Fuente: elaboración propia

Consumo durante una recepción con el BT conectado: para esta situación el consumo oscila alrededor de los 55 mA.

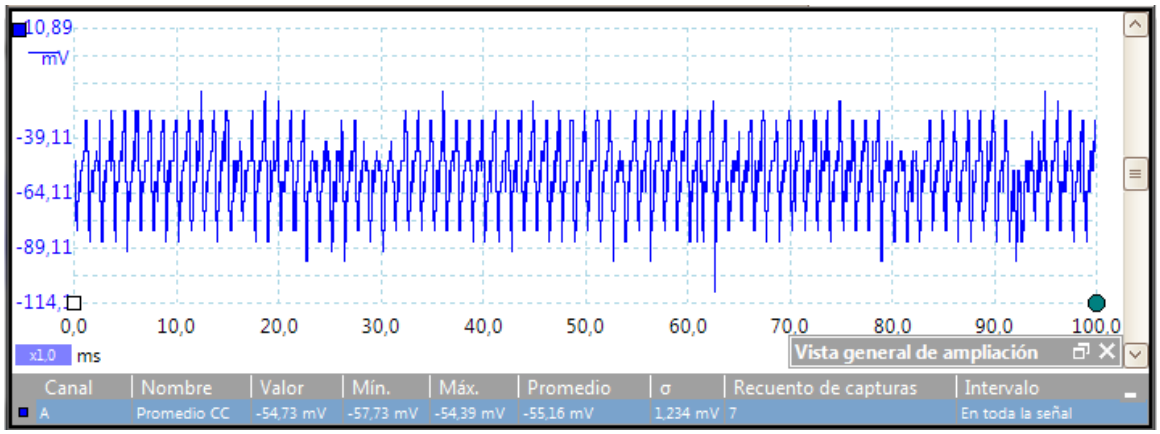


Figura 112. Consumo medio durante una recepción estando de vinculado el Bluetooth. Fuente: elaboración propia

Los consumos anteriores quedan simplificados en la siguiente tabla:

Tabla 24. Resumen de los consumos medios en las distintas operaciones. Fuente: elaboración propia.

Perfil	Duración (s)	Consumo (mA)
Modo Sleep	-	0.06
Medición y envío	7	30
Envío y recepción	60	25
Espera de conexión BT	30	25
Conexión BT	-	35
Conexión BT y enviando	7 s	60
Conexión BT y recepción	60	55

11.2. Dimensionamiento de la batería

Para el dimensionamiento de la batería se ha tenido en cuenta el cálculo del consumo medio diario. La mayor parte del consumo viene provocada por el envío y la recepción de mensajes vía SigFox. Como el dispositivo es programable y se puede configurar la tasa de envío se han evaluado distintos escenarios.

- **Escenario 1:** 120 mensajes diarios de 12 bytes, corresponden a mediciones cada 2 minutos y envíos cada 12 minutos, y un envío y recepción.
- **Escenario 2:** 48 mensajes diarios de 12 bytes, corresponden a mediciones cada 5 minutos y envíos cada 30 minutos, y un envío y recepción.
- **Escenario 3:** 16 mensajes diarios de 12 bytes, corresponden a mediciones cada 15 minutos y envíos cada 90 minutos, y un envío y recepción diario.
- **Escenario 4:** 4 mensajes diarios de 12 bytes, corresponden a mediciones cada 60 minutos y envíos cada 4 horas, y un envío y recepción diario.

Para la medición del sensor se ha estimado unos 5 mA de corriente media y un tiempo de Warmtime de 250 ms. Como siempre se mide cada minuto, corresponde un total de 360 s al día.

Tabla 25. Cálculo de consumo diario para los distintos escenarios. Fuente: elaboración propia.

	Medición sensor (5 mA)	Envío (30 mA)	Envío y recepción (25 mA)	Sleep (0.06mA)	Consumo medio diario (mA)
Escenario 1	360 s	840 s	60 s	85140 s	0.389
Escenario 2	360 s	336 s	60 s	85644 s	0.214
Escenario 3	360 s	112 s	60 s	85868 s	0.137
Escenario 4	360 s	28 s	60 s	85920 s	0.108

Por lo tanto, sin tener en cuenta ningún tipo de pérdida del sistema de almacenamiento, para garantizar una duración mínima de 2 años en el escenario más desfavorable, se requiere una una capacidad de:

$$C(mAh) = Consumo_{medio}(mA) \cdot t(h) = 0.4\,mA \cdot 2\,años \cdot 365\frac{d}{año} \cdot 24\frac{h}{d} = 7000\,mAh \qquad \text{Ec.33}$$

No obstante, las baterías sufren una descarga progresiva de las mismas, incluso sin consumo. Esta pérdida se computa mediante el coeficiente de descarga. De esta la duración de la batería se puede estimar mediante la siguiente fórmula:

$$\sum_{x=1}^{X} (K \cdot (1-K)^{x-1} \cdot C) + I_{media} \cdot X = C \qquad \text{Ec.34}$$

$Donde$:
$X = nº\,mes$
$K = coeficiene\ de\ descarga\ mensual$
$C = capacidad$
$I_{media} = consumo\ mensual$

El coeficiente de descarga varía en gran medida dependiendo de la tecnología de batería, por ejemplo, una pila no recargable de Li-$SOCL_2$ el coeficiente de descarga suele estar por debajo del 1% anual (0.08% mes). Mientras que para una batería recargable este coeficiente suelen ser mucho mayores, entorno al 0.35-2.5% (16) mensual para baterías ion-Litio o polímero de Litio o por encima del 20% mensual para baterías NiMH (a excepción de las baterías LSD Low Self Discharge, las que tienen pérdidas de un 0.15% mensual).

El cálculo del tiempo de vida será realizado para las siguientes opciones de baterías:

- Batería Li-$SOCL_2$, LTS 26500de 3.7V y 7.7 Ah, tamaño tipo D (61x33 mm)
- 4 (4P) x Batería ion-Litio INR-18650 20R de 3.7 V y 2 Ah (65x18 mm)
- 8 (4Px2S) x Batería Ni-MH LSD, Eneloop de 1.2 V y 2 Ah, tamaño tipo AA (50x14mm)

Tabla 26. Cálculo de duración de batería para los distintos escenarios. Fuente: elaboración propia.

	1 x LTS 26500	4 x INR-18650 20R	8 x Eneloop AA
Capacidad (Ah)	7.7	8	8
Voltage	3.7	3.7	3.6
Coeficiente descarga (%)	0.08	1.5	0.15
Autonomía 1 (0.4mA)	2y 1m 24d	1y 7m 10d	2y 2m 9d
Autonomía 2 (0.2mA)	4y 2m 17d	2y 5m 26d	4y 2m 17d
Autonomía 3 (0.15mA)	5y 6m 15d	2y 10m 18d	5y 5m 22d
Autonomía 4 (0.1mA)	8y 1m 3d	3y 5m 2d	7y 9m 27d

Como se puede observar, cuánto menor es el consumo, más importante es que el coeficiente de autodescarga sea lo más reducido posible. Para el prototipo inicial, se había propuesto la selección de una batería de Litio (ion Litio o de polímero de Litio) tras el presente análisis se ha demostrado como otras alternativas como las pilas de Li-$SOCl_2$ son una mejor elección para aumentar drásticamente su autonomía, reduciendo su capacidad.

No obstante, para el actual prototipo debido a la necesidad del empleo del cargador implementado (MCP73833) se ha seleccionado una batería formada por las pilas de ion-Litio. Ya que presentan un coeficiente de autodescarga menor que las de polímero de Litio (5% aprox.). Para garantizar al menos 2 años de autonomía se debe implementar un pack formado por 6 pilas (2 años, 1 mes y 8 días).

Sin embargo, para futuros desarrollos se plantea la sustitución de la batería recargable por una batería no recargable de $LiSOCl_2$. O la búsqueda de cargadores para baterías de Ni-MH.

12. CONCLUSIONES Y TRABAJO FUTURO

En el presente Trabajo Final de Máster, se ha diseñado una primera versión de un sistema para la monitorización continua de redes de distribución de aguas. Su monitorización permitirá la rápida detección y localización de las averías que puedan surgir. También permitirá un mayor control del uso actual del sistema, identificando las zonas con mayor riesgo a su saturación.

En primer lugar, se ha realizado un estudio de los requisitos de la aplicación, atendiendo a las características del entorno, los tipos de sensores utilizados y a su manipulación/configuración. A partir de los requisitos se han definido los distintos subsistemas que conformarán la solución: subsistema de comunicación, de adquisición, de interfaz con el operario, de alimentación y controlador.

En cuanto al subsistema de comunicación, se ha realizado un estudio de las principales tecnologías que han impulsado el IoT, seleccionando la tecnología SigFox como lo más apropiada, por sus características de bajo consumo y coste de implementación.

Para la selección de los distintos componentes principales se ha realizado un estudio técnico-económico. Entre los componentes seleccionados destacan el microcontrolador PIC24FJ128GC006, el transceptor SigFox ATA8520E, el módulo Bluetooth RN-42 y el display SSH116. A continuación, se ha procedido a su diseño, implementación y fabricación, mostrando un especial cuidado en el desarrollo de las conexiones de radiofrecuencia.

A partir del prototipo fabricado, se han programado tanto el microcontrolador como la aplicación móvil. En un primer lugar, se ha programado la interacción del microcontrolador con cada uno de los periféricos, una vez resuelta la comunicación y el desarrollo de todas las funciones individuales, se ha realizado una estructuración del código completo.

Tras la primera versión del código del microcontrolador se ha desarrollado una primera versión de la aplicación. Tras un testeo iterativo se fueron solucionando los distintos problemas surgidos y se fueron añadiendo funcionalidades hasta desarrollar una versión completa.

Con el prototipo funcional se han realizado una serie de pruebas para validar los aspectos más relevantes del dispositivo, las lecturas de la ADC, la calibración del sistema, la cobertura de la red SigFox, el funcionamiento general y cuantificar el consumo para poder realizar un diseño adecuado de las baterías. Se ha garantizado una autonomía de más de 2 años mediante el uso de baterías de ion-Litio INR-18650, y se han estudiado alternativas inicialmente no planteadas para el aumento de la autonomía hasta 8 años.

Todo ello ha permitido desarrollar un registrador de datos genérico de bajo coste (<50 €) de alta resolución (16 bits) y bajo consumo (<0.4 mA) que implementa una nueva tecnología inalámbrica LP-WAN, SIGFOX, para el envío automático de datos al servidor web.

Que puede ser fácilmente configurable y adaptable a todo tipo de sensores de presión comerciales mediante una aplicación móvil. Por lo tanto, se han alcanzado los objetivos del actual Trabajo Final de Máster.

A continuación, se plantean una serie de trabajos futuros, que podrían llevarse a cabo tras el desarrollo del prototipo inicial.

Se plantea la creación de un servidor web para la visualización de los datos de forma gráfica, pudiendo representar históricos para su correcto análisis y estudio. Desde el servidor web SigFox se pueden crear *callbacks* para el envío de datos a otros servidores mediante el método HTTP.

También sería de interés la creación de una aplicación móvil para dispositivos i-OS o incluso el desarrollo de una versión de ordenador que sustituyera la actual conexión Bluetooth por la conexión USB del microcontrolador.

Otro posible trabajo, como ya se ha comentado, sería la sustitución del sistema de carga de batería de Litio por un sistema de mayor autonomía como podría ser un cargador de baterías Ni-MH LSD o sustituyéndolo por un sistema no recargable de baterías Li-$SOCl_2$.

Finalmente, se plantea el desarrollo industrial del dispositivo, su implementación dentro de una caja con protección IP68 con conector exterior para el sensor y la antena. Tras el diseño, deberían iniciarse las certificaciones necesarias (grado de protección, compatibilidad electromagnética, marcado CE) para su futura comercialización.

13. REFERENCIAS

1. **Ticnegocios. [En línea] [Citado el: 25 de Noviembre de 2019.] https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-11-infraestructuras-i-redes-inalambricas/.**

2. Iot for All. [En línea] [Citado el: 26 de Noviembre de 2019.] https://www.iotforall.com/iot-connectivity-comparison-lora-sigfox-rpma-lpwan-technologies/.

3. Miranda, Javier Saiz. *Estudio en detalle de NB-IoT. Comparación con otras tecnologías LPWAN considerando diferentes patrones de tráfico.* s.l. : Trabajo Final de Master UOC-URL, 2019.

4. NXTIOT. [En línea] [Citado el: 2019 de Noviembre de 29.] https://www.nxtiot.io/index.php/2018/07/23/conectividad-ahora-y-mas-alla-explorando-las-conexiones-cat-m1-nb-iot-y-lpwan/.

5. *APPLICATION BASED COMPARISON OF DIFFERENT ANALOG TO DIGITAL CONVERTER ARCHITECTURES.* BHATIA, & Bhatia Lohit, Veepsa & Pandey, Neeta & ASOK, BHATTACHARYYA. 2, s.l. : International Journal of Engineering Science and Technology, 2010.

6. *High Accuracy Sigma-Delta Modulator implementations via Suboptimal Quasi-Sliding Mode Control.* Pilloni, Alessandro & Franceschelli, Mauro & Pisano, Alessandro & Usai, Elio. s.l. : 438-443. 10.1109/VSS.2018.8460315. , 2018.

7. Lai, Yuk M. 20 - Power Supplies. 659-684. *Power Electronics Handbook (Fourth Edition).* s.l. : Butterworth-Heinemann, 2018.

8. Schweber, Bill. DigiKey. *Understanding the Advantages and Disadvantages of Linear Regulators.* [En línea] https://www.digikey.com/en/articles/understanding-the-advantages-and-disadvantages-of-linear-regulators.

9. Díaz, Manuel J. Bellido. Normas Básicas y Recomendaciones en el Diseño de PCBs. 2015.

10. *Chapter 20: Transmission Lines. The ARRL Handbook for Radio Communications (87th ed.).* Silver, H. Ward , Wilson y Mark J. s.l. : The American Radio Relay League. , 2010. ISBN 978-0-87259-144-8.

11. *Aplicaciones y Teoría de Ingeniería de Microondas.* Castillo, Ebert & Castillo-Aranibar, Patricia & Polar, Manuel & Gonzales Fuentes, Lee & Zenteno, Efrain. 2014. .

12. Leal Elías, Miguel. Capitulo 3. Líneas de Transmisión, Guías de Ondas y Circuitos Pasivos. *ESTUDIO DE LAS PUESTAS A TIERRA EN CIRCUITOS MICROSTRIP A FRECUENCIAS DE MICROONDAS.* s.l. : E-Reding. Biblioteca de Ingeniería. Universidad de Sevilla.

13. Calculadora de impedancia de línea para CPGW. [En línea] http://wcalc.sourceforge.net/cgi-bin/coplanar.cgi.

14. Protocolo comunicación SSH1106. [En línea] http://www.farnell.com/datasheets/609753.pdf.

15. Android Studio. [En línea] https://developer.android.com/studio/intro.

16. ***Measuring Reversible and Irreversible Capacity Losses on Lithium-Ion Batteries.*** **Redondo-Iglesias, Eduardo, Venet, Pascal y Pelissier, Serge. s.l. : IEEE Vehicle Power and Propulsion Conference (VPPC). p. 7. , 2016 . doi:10.1109/VPPC.2016.7791723. ISBN 9.**

DOCUMENTO II:

PRESUPUESTO

1. DESCRIPCIÓN DEL PRESUPUESTO

1.1 Descripción de las unidades de obra

Para el desarrollo del Trabajo Final de Grado se han empleado un total de 350 h, la totalidad de las horas dedicadas se pueden agrupar en los siguientes grupos:

- **Estudios previos:** 20 horas incluye todos los estudios previos realizados para la definición de la solución
    - **Estudio de los requisitos del dispositivo:** 5 horas, tras un análisis del sector de aplicación, se han evaluado los requisitos a exigir al sistema.
    - **Definición del sistema**: 5 horas, tras el estudio de los requisitos se han definido las características del sistema.
    - **Estudio de las tecnologías de comunicación:** 10 horas, se ha seleccionado la tecnología de comunicación más apropiada para el sistema.
- **Desarrollo del sistema:** 110 horas, agrupa todas las operaciones requeridas para el desarrollo físico de la solución a partir de la idea desarrollada.
    - **Análisis técnico-económico de los componentes:** 15 horas, conlleva la selección de los componentes principales que forman cada uno de los subsistemas definidos
    - **Desarrollo de la PCB:** 80 horas, incluye la elaboración de la circuitería para cada uno de los subsistemas, así como la interconexión entre ellos. También incluye el diseño de la tarjeta impresa.
    - **Ensamblaje y soldadura de prototipos:** 15 horas, incluye el montaje y la soldadura de 3 prototipos.
- **Programación y validación del equipo:** 150 horas, agrupa las actividades necesarias para conseguir un equipo funcional a partir del desarrollo físico previo.
    - **Programación del microcontrolador:** 85 horas, incluye tanto la programación como las pruebas iniciales de testeo.
    - **Programación de la aplicación móvil:** 45 horas, incluye el desarrollo de un App mediante Android Studio.
    - **Test de validación del dispositivo:** 20 horas, incluye todas las pruebas de validación realizadas para asegurar el correcto funcionamiento del equipo desarrollado.
- **Elaboración de la documentación técnica:** 70 horas, incluye por una parte la búsqueda y aprendizaje de distintas tecnologías empleadas en el presente documento, y por otra parte la redacción del documento memoria, presupuesto a partir de los datos obtenidos en los apartados anteriores.

1.2 Cálculo del precio de los materiales empleados

Hemos de incorporar al presupuesto el gasto asociado a las herramientas y/o programas utilizados para el desarrollo del proyecto, así como el coste salarial de los involucrados en el proyecto.

- **Coste de amortización del portátil:** se ha estimado a partir de la amortización del portátil empleado (Acer Aspire 5750G, Intel Core i7-2670QM, 4GB DDR3), su precio asciende a un total de 630€, se ha presupuesto una vida útil de unos 5 años (60 meses), por otra parte la duración del trabajo final de grado se ha estimado en unos 6 meses por lo que el coste de su amortización se ha aproximado a 63€.
- **Licencia de Microsoft Office 2013:** A través del catálogo de Microsoft, se ha encontrado una oferta para el sector empresarial de unos 8,8€ al mes.
- **Programa MPLAB vIDE y el compilador CCS:** la IDE de Microchip es gratuita mientras que el compilador supone un coste total de 682,47€, a partir de este dato se ha estimado un coste mensual de 22.68 €
- **Programa Eagle de AutoDesk:** el precio de la licencia anual es de 133.10 €
- **Material electrónico:** incluye el uso de estaciones de soldadura, estaño, flux, cables, conectores, osciloscocopio, etc. Se ha estimado en 90€
- **Componentes y PCB del prototipado:** incluye todos los componentes necesarios para el ensamblaje de 3 prototipos. 97.81€ (desglosado en los anexos)
- **Material de oficina y reprografía:** dicho conste incluye el material de oficina empleado, además del coste de impresión de los distintos documentos, se ha estimado en unos 30€.
- **Ingeniero Industrial en formación:** su salario se ha estimado en 26,7€ por hora.
- **Consulta y seguimiento por parte del tutor:** en este caso se ha estimado en 35,4€ por hora.

2. PRESUPUESTO DEL PROYECTO

2.1 Cuadro de precios

2.1.1. Cuadro de precio de los operarios

Salario de los trabajadores			
Cód.	**U.M**	**Descripción**	**Precio**
M.O.1	h	Ingeniero Industrial en formación	26,7
M.O.2	h	Consulta y seguimiento tutor	35,4

2.1.2. Cuadro de precio de los materiales

Precio de los materiales			
Cód.	**U.M**	**Descripción**	**Precio**
M.1	u	Amortización del portátil	63,00
M.2	mes	Licencia Microsoft Office 2013	8,80
M.3	mes	MPLAB IDE y CCS compiles (3 meses)	22,68
M.4	mes	Licencia Autodesk Eagle (3 meses)	133,10
M.5	u	Material electrónico	90,00
M.6	u	Material de reprografía	30,00
M.7	u	Componentes 3 prototipos	97,81

2.1.3. Cuadro de precios descompuestos

U.O.1 Estudio previo					
Cód.	**U.M**	**Descripción**	**Rdto.**	**Precio**	**Importe**
M.O.1	h	Ingeniero Industrial en formación	20	26,70	534,00
M.O.2	h	Consulta y seguimiento tutor	10	35,40	350,40
%	-	Costes directos complementarios	0,02		17,61
%	-	Costes Indirectos	0,03		26,94
			Coste total:		928,95

U.O.2 Desarrollo físico del sistema					
Cód.	**U.M**	**Descripción**	**Rdto.**	**Precio**	**Importe**
M.O.1	h	Ingeniero Industrial en formación	110	26,70	2937,00
		Análisis técnico-económico de componentes	15		
		Desarrollo de la PCB	80		
		Ensamblaje y soldadura de prototipos	15		
M.O.2	h	Consulta y seguimiento tutor	20	35,40	708,00
M.4	u	Licencia Autodesk Eagle	1	399,30	399,30
M.5	u	Material electrónico	1	90,00	90
M.7	u	Componentes y PCB para prototipos	1	97,81	97,81
%	-	Costes directos complementarios	0,02		84,64
%	-	Costes Indirectos	0,03		129,50
			Coste total:		4446,25

U.O.3 Programación y validación del equipo					
Cód.	**U.M**	**Descripción**	**Rdto.**	**Precio**	**Importe**
M.O.1	h	Ingeniero Industrial en formación	150	26,70	4005,00
		Programación del microcontrolador	85		
		Programación de la aplicación móvil	45		
		Test de validación	20		
M.O.2	h	Consulta y seguimiento tutor	10	35,40	354,00
M.3	u	Programa MPLAB IDE y compilador CCS	1	67,70	67,70
%	-	Costes directos complementarios	0,02		88,53
%	-	Costes Indirectos	0,03		135,46
			Coste total:		4650,69

U.O.4 Elaboración de documentación técnica					
Cód.	**U.M**	**Descripción**	**Rdto.**	**Precio**	**Importe**
M.O.1	h	Ingeniero Industrial en formación	70	26,70	1869,00
M.O.2	h	Consulta y seguimiento tutor	5	35,40	177,00
M.2	u	Licencia Microsoft Office	1	52,80	52,80
%	-	Costes directos complementarios	0,02		41,98
%	-	Costes Indirectos	0,03		64,22
			Coste total:		2205,00

U.O.5 Otros gastos					
Cód.	**U.M**	**Descripción**	**Rdto.**	**Precio**	**Importe**
M.1	u	Amortización del ordenador portátil	1	63,00	63,00
M.5	u	Material de reprografía	1	30,00	30,00
%	-	Costes directos complementarios	0,02		1,86
%	-	Costes Indirectos	0,03		2,85
			Coste total:		97,71

2.1.4. Cuadro de precio unitario

Cód.	**Descripción**	**Precio**
U.O.1	Estudio previo	928,95
U.O.2	Desarrollo físico del sistema	4446,25
U.O.3	Programación y validación del sistema	4650,69
U.O.4	Elaboración de la documentación técnica	2205,0
U.O.5	Otros gastos	97,71

2.2 Presupuesto general del proyecto

		Cant.	Precio	Importe
U.O.1	Estudio previo	1	928,95	928,95
U.O.2	Desarrollo físico del sistema	1	4446,25	4446,25
U.O.3	Programación y validación del sistema	1	4650,69	4650,69
U.O.4	Elaboración de la documentación técnica	1	2205,0	2205,00
U.O.5	Otros gastos	1	97,71	97,71

Presupuesto Ejecución Material **12328,60**

13% Gastos generales.....1602,72

6% Beneficio Industrial.....739,72

Presupuesto Ejecución por Contrata **14767,20**

21% IVA.....3101,11

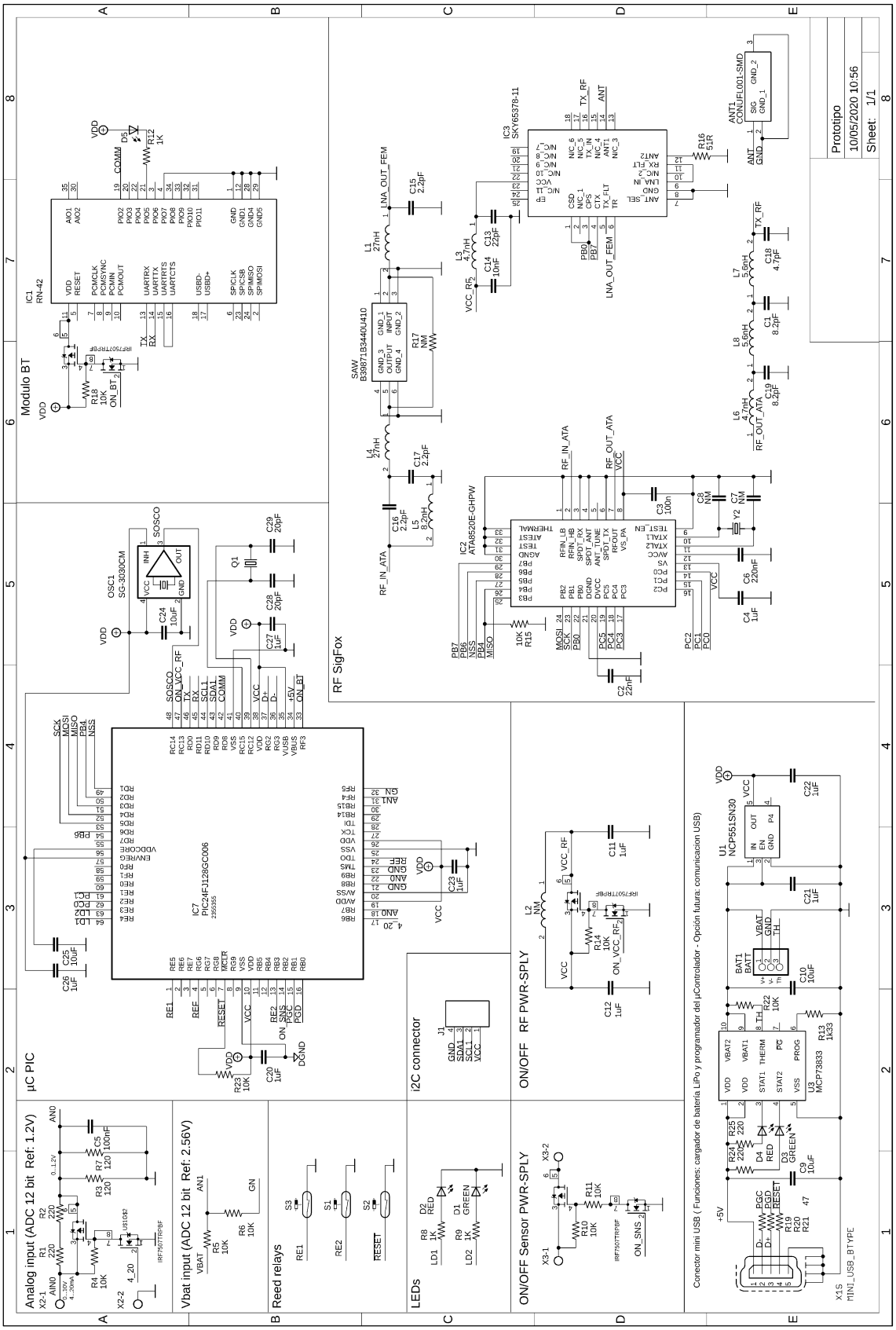
Presupuesto Total: **17868,31**

Asciende el presupuesto proyectado, a la expresada cantidad de:
DIECISIETE MIL OCHOCIENTOS SESENTA Y OCHO CON TREINTA Y UNO EUROS.

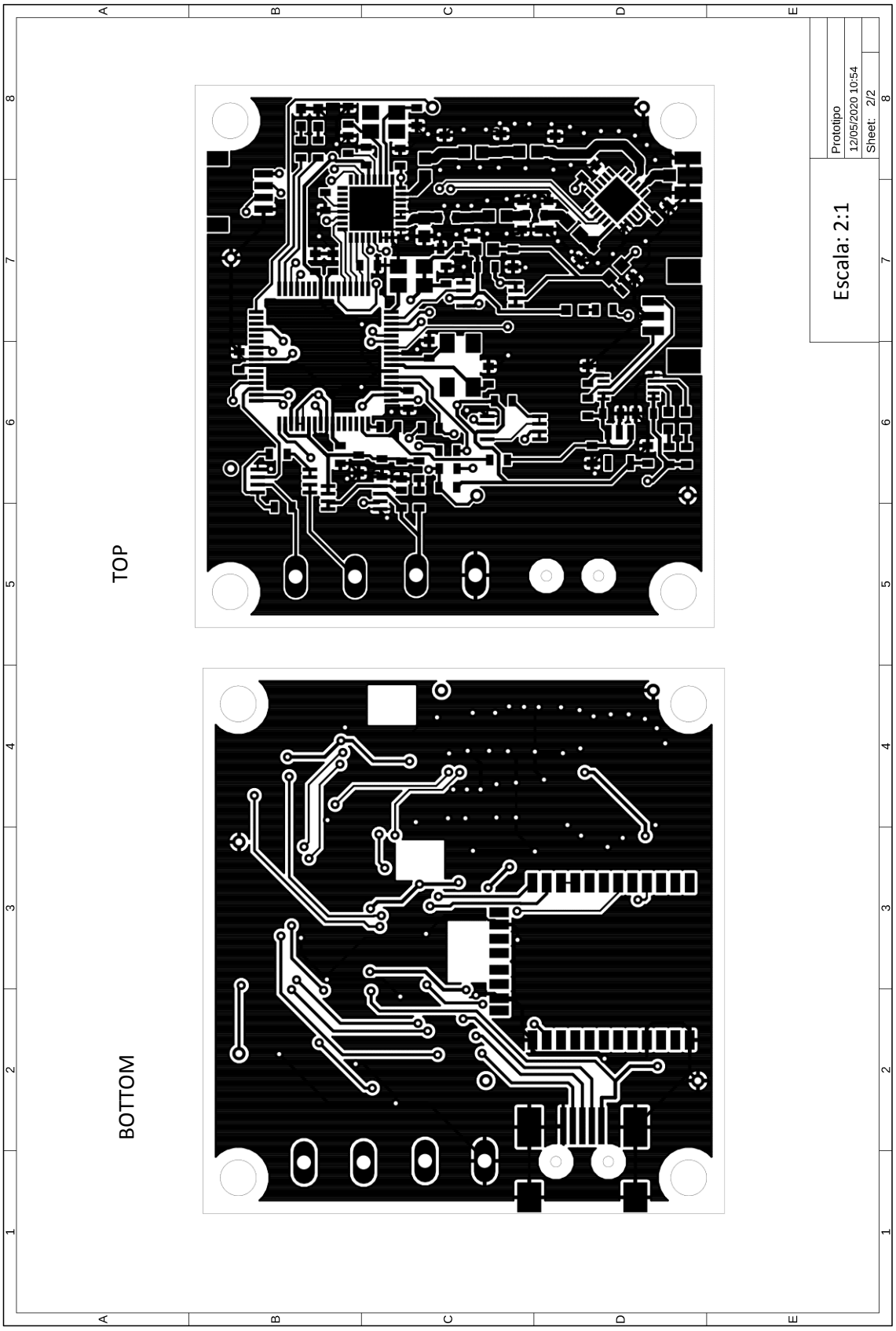
DOCUMENTO III:

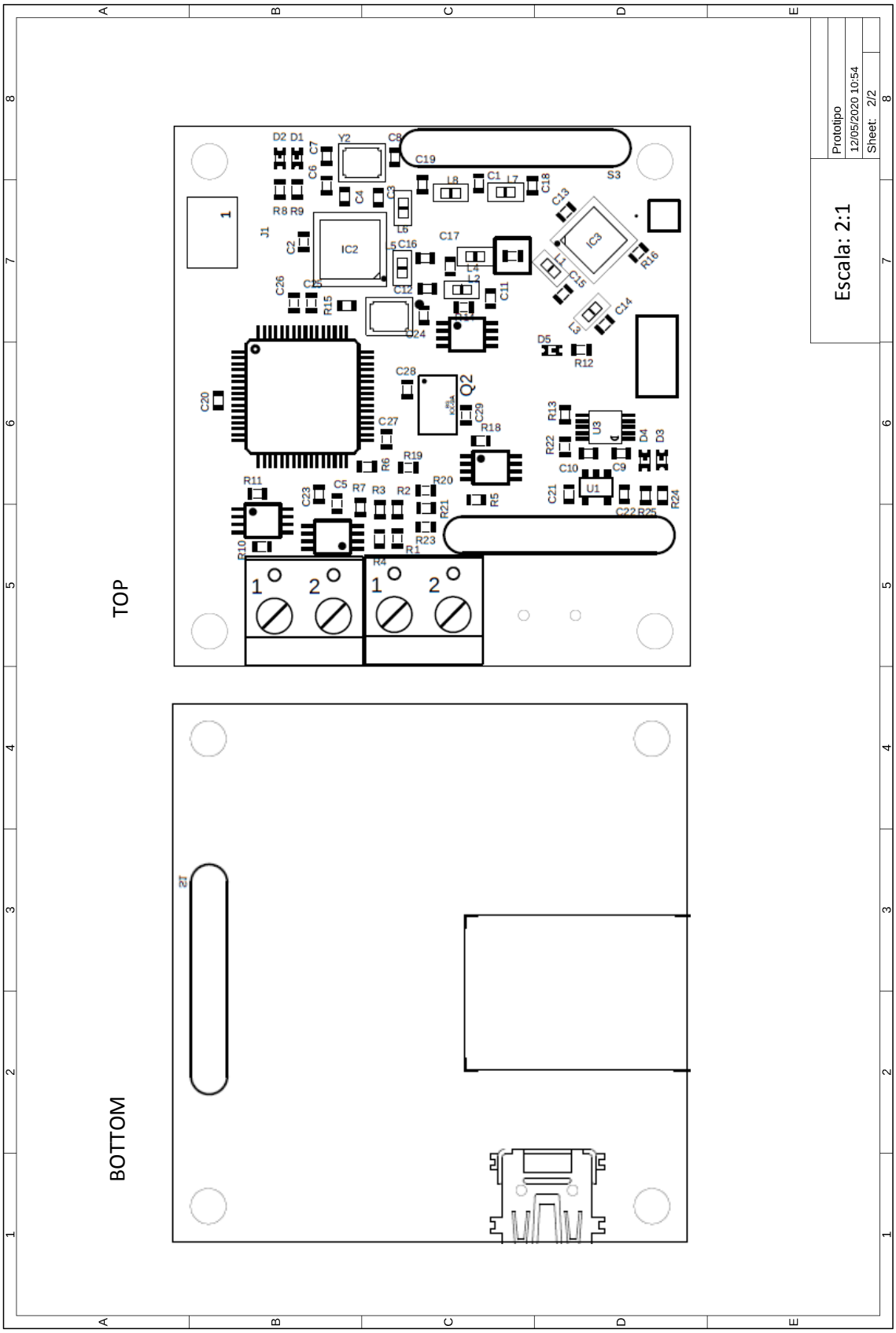
ANEXOS

1. ESQUEMÁTICO DEL PROTOTIPO



2. LAYOUT PCB




Prototipo
12/05/2020 10:54
Sheet: 2/2
Escala: 2:1
TOP
BOTTOM
IC2
IC3
U1
U3
Q2
S3
J1

3. LISTA DE MATERIALES

Cant.	Tipo	Identificador	Valor	Descripción	Precio	Total
2	Bobina	L1,L4	27nH	Chip inductor, SMD 0603	0,19	0,38
2	Bobina	L1,L3	4,7nH	Chip Inductor, SMD 0603, Ir=300mA, Rdc=0.2Ohm, +/- 0.3nH	0,125	0,25
2	Bobina	L7,L8	5,6nH	Chip Inductor, SMD 0603, Ir=300mA, Rdc=0.26Ohm, +/- 0.3nH	0,174	0,348
1	Bobina	L5	8,2nH	RF Inductor, 8.2nH, +/-5%, 0.4Ohm, 250mA, 0605	0,09	0,09
10	Resistencia	R4,R5,R6,R10,R11, R14,R15,R18,R2,R23	10 kOhms	Thin film resistor, SMD 0603, 5%, 100mW	0,05	0,5
2	Resistencia	R3,R7	120 Ohms	Thin film resistor, SMD 0603, 1%, 100mW	0,09	0,18
3	Resistencia	R8,R9,R12	1 kOhms	Thin film resistor, SMD 0603, 5%, 100mW	0,016	0,048
1	Resistencia	R13	1k33 Ohms	Thin film resistor, SMD 0603, 5%, 100mW	0,016	0,016
4	Resistencia	R1,R2,R24,R25	220 Ohms	Thin film resistor, SMD 0603, 1%, 100mW	0,021	0,084
3	Resistencia	R19,R20,R21	47 Ohms	Thin film resistor, SMD 0603, 5%, 100mW	0,016	0,048
1	Resistencia	R16	51 Ohms	Thin film resistor, SMD 0603, 5%, 100mW	0,016	0,016
2	Condensador	C3,C5	100 nF	Ceramic capacitor, SMD 0603, X7R, 16V, +/-10%	0,04	0,08
1	Condensador	C14	10nF	Ceramic capacitor, SMD 0603, X7R, 16V, +/-5%	0,09	0,09
4	Condensador	C9,C10,C24,C25	10uF	Ceramic capacitor, SMD 0603, X5R, 6.3V, +/-10%	0,221	0,884
9	Condensador	C4,C11,C12,C20,C21, C22,C23,C26,C27	1uF	Ceramic capacitor, SMD 0603, X5R, 6.3V, +/-10%	0,058	0,522
2	Condensador	C16,C17	2,2pF	Ceramic capacitor, SMD 0603, NP0, 50V, +/-2%	0,148	0,296
	Condensador	C26	220nF	Ceramic capacitor, SMD 0603, X5R, 10V, +/-10%	0,09	0
1	Condensador	C2	22nF	Ceramic capacitor, SMD 0603, X7R, 16V, +/-10%	0,05	0,05
1	Condensador	C13	22pF	Ceramic capacitor, SMD 0603, X7R, 16V, +/-5%	0,118	0,118
1	Condensador	C18	4,7pF	Ceramic capacitor, SMD 0603, NP0, 50V, +/-2%	0,056	0,056
2	Condensador	C1,C19	8,2pF	Ceramic capacitor, SMD 0603, NP0, 50V, +/-2%	0,125	0,25

Cant.	Tipo	Identificador	Valor	Descripción	Precio	Total
1	Conector	BAT1	CONECTOR	CONNECTOR_BATTERY_25MM	0,23	0,23
1	Conector	ANT1	CONUFL001-SMD	CONUFL001-SMD	0,221	0,221
1	Conector	X1	MINI_USB_BTYPE	32005-201	0,65	0,65
1	Integrado	IC2	ATA8520E-GHPW	QFN50P500X500X90-33N	2,3	2,3
1	Integrado	SAW	B39871B3440U410	B39871B3440U410	1,28	1,28
1	Integrado	U3	MCP73833MSOP10	MSOP10	0,536	0,536
1	Integrado	U1	NCP551SN30	SOT23-5	0,325	0,325
1	Integrado	IC7	PIC24FJ128GC006	TQFP64-10X10	3,65	3,65
3	Integrado	S1,S2,S3	REED-SWITCH-MK6-7	MK6-7	0,32	0,96
1	Integrado	IC1	RN-42		12,35	12,35
1	Integrado	IC3	SKY65378-11	QFN50P400X400X90-25N	1,28	1,28
2	Leds	D4,D2	red	Standard LED - SMD Red 2; 0603	0,15	0,3
1	Leds	D3	green	Standard LED - SMD Green 2; 0603	0,16	0,16
1	Leds	D1	ambar	Standard LED - SMD Ambar; 0603	0,15	0,15
4	Mosfet	U$1,U$2,U$3,U$4	IRF7507TRPBF	TSSOP8	0,423	1,692
1	Oscilador	Y2	RH100-11.0592	CRYSTAL_3.2X2.5	0,23	0,23
1	Oscilador	OSC1	SG-3030CM	OSC_3.2X2.5MM	0,65	0,65
				TOTAL		**31,27**

4. CÓDIGO MICROCONTROLADOR

4.1. Header.h

```
#fuses NOJTAG      // JTAG port is disabled
#fuses NOPROTECT   // Code protection is disabled
#fuses NOWRT       // Writes to program memory are allowed
#fuses ICSP1       // Emulator functions are shared with PGEC1/PGED1
#fuses NOWDT       // WDT is disabled; SWDTEN bit is disabled
#fuses WPOSTS16    // Watchdog Timer Postscaler Select bits-> 1:32,768 (NOT
USED)
#fuses NOIESO      // Internal External Switchover bit: IESO mode (Two-Speed
Start-up) is disabled
#fuses HS
#fuses PR
#fuses CKSNOFSM    // Clock switching is enabled, Fail-Safe Clock Monitor is
disabled
#fuses OSCIO       // OSCO Pin Configuration bit: OSCO/CLKO/RC15 functions as
port I/O (RC15)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#define ON( pin)   output_high(pin)
#define OFF(pin)   output_low( pin)
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

/*************************************************************************
                              DEFINICIÓN DE PINOUT
*************************************************************************/

//-----------------------------------LEDS-----------------------------------
#define LD1     PIN_E4
#define LD2     PIN_E3
// --------------------------PINES DE ACTIVACION-----------------------------
#define ON_BT       PIN_F3
#define ON_VCC_RF   PIN_C13
#define ON_SNS      PIN_B2
#define Ac_4_20     PIN_B6
#define GN          PIN_F5
//-------------------------ENTRADAS ANALOGICAS-------------------------
#define AN1         PIN_F4
#define AN0         PIN_B7 //B9
//-------------------------- INTERRUPTORES MAGNETICOS-----------------------
#define INT_REED_1  PIN_E5
#define INT_REED_2  PIN_B3
//-------------------------- PINES BLUETOOTH RN42(UART)----------------------
#define BT_TX       PIN_D0
#define BT_RX       PIN_D11
#define COMM        PIN_D8
//-------------------------- PINES SIGFOX ATA8520(SPI)-----------------------
#define PC0         PIN_E2  //NRESET
#define PC1         PIN_E1  // N_ON despierta cuando esta low
#define PB4         PIN_D2  // H_ON despierta cuando esta high
#define PB6         PIN_D6  // Pin de interrupcion
#define SIGFOX_SDI  PIN_D3
#define SIGFOX_SDO  PIN_D4
#define SIGFOX_SCLK PIN_D5
#define CS          PIN_D1
```

4.2. OLED.c

```
/*************************************************************************
*
      LIBRERÍA PARA EL DISPLAY
*
*************************************************************************/

//LCD
#ifndef SSH1106
const unsigned int8 text[51][5] =
{

   0x00, 0x00, 0x00, 0x00, 0x00, // SPACE
   0x00, 0x00, 0x5F, 0x00, 0x00, // !
   0x00, 0x03, 0x00, 0x03, 0x00, // "
   0x14, 0x3E, 0x14, 0x3E, 0x14, // #
   0x24, 0x2A, 0x7F, 0x2A, 0x12, // $
   0x43, 0x33, 0x08, 0x66, 0x61, // %
   0x36, 0x49, 0x55, 0x22, 0x50, //&
   0x00, 0x05, 0x03, 0x00, 0x00, // '
   0x00, 0x1C, 0x22, 0x41, 0x00, // (
   0x00, 0x41, 0x22, 0x1C, 0x00, //)
   0x14, 0x08, 0x3E, 0x08, 0x14, // *
   0x08, 0x08, 0x3E, 0x08, 0x08, // +
   0x00, 0x50, 0x30, 0x00, 0x00, //,
   0x08, 0x08, 0x08, 0x08, 0x08, // -
   0x00, 0x60, 0x60, 0x00, 0x00, // .
   0x20, 0x10, 0x08, 0x04, 0x02, // /
   0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
   0x04, 0x02, 0x7F, 0x00, 0x00, // 1
   0x42, 0x61, 0x51, 0x49, 0x46, // 2
   0x22, 0x41, 0x49, 0x49, 0x36, // 3
   0x18, 0x14, 0x12, 0x7F, 0x10, // 4
   0x27, 0x45, 0x45, 0x45, 0x39, // 5
   0x3E, 0x49, 0x49, 0x49, 0x32, // 6
   0x01, 0x01, 0x71, 0x09, 0x07, // 7
   0x36, 0x49, 0x49, 0x49, 0x36, // 8
   0x26, 0x49, 0x49, 0x49, 0x3E, // 9
   0x00, 0x36, 0x36, 0x00, 0x00, // :
   0x00, 0x56, 0x36, 0x00, 0x00, //;

   0x08, 0x14, 0x22, 0x41, 0x00, // <
   0x14, 0x14, 0x14, 0x14, 0x14, // =
   0x00, 0x41, 0x22, 0x14, 0x08, // >
   0x02, 0x01, 0x51, 0x09, 0x06, // ?
   0x3E, 0x41, 0x59, 0x55, 0x5E, // @
   0x7E, 0x09, 0x09, 0x09, 0x7E, // A
   0x7F, 0x49, 0x49, 0x49, 0x36, // B
   0x3E, 0x41, 0x41, 0x41, 0x22, // C
   0x7F, 0x41, 0x41, 0x41, 0x3E, // D
   0x7F, 0x49, 0x49, 0x49, 0x41, // E
   0x7F, 0x09, 0x09, 0x09, 0x01, // F
   0x3E, 0x41, 0x41, 0x49, 0x3A, // G
   0x7F, 0x08, 0x08, 0x08, 0x7F, // H
   0x00, 0x41, 0x7F, 0x41, 0x00, // I
   0x30, 0x40, 0x40, 0x40, 0x3F, // J
   0x7F, 0x08, 0x14, 0x22, 0x41, // K
   0x7F, 0x40, 0x40, 0x40, 0x40, // L
   0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
   0x7F, 0x02, 0x04, 0x08, 0x7F, // N
   0x3E, 0x41, 0x41, 0x41, 0x3E, // O
   0x7F, 0x09, 0x09, 0x09, 0x06, // P
   0x1E, 0x21, 0x21, 0x21, 0x5E, // Q
   0x7F, 0x09, 0x09, 0x09, 0x76
}; // R
```

```
const unsigned int8 text2[44][5]= {
   0x26, 0x49, 0x49, 0x32, // S
   0x01, 0x01, 0x7F, 0x01, 0x01, // T
   0x3F, 0x40, 0x40, 0x40, 0x3F, // U
   0x1F, 0x20, 0x40, 0x20, 0x1F, // V
   0x7F, 0x20, 0x10, 0x20, 0x7F, // W
   0x41, 0x22, 0x1C, 0x22, 0x41, // X
   0x07, 0x08, 0x70, 0x08, 0x07, // Y
   0x61, 0x51, 0x49, 0x45, 0x43, // Z
   0x00, 0x7F, 0x41, 0x00, 0x00, // [
   0x02, 0x04, 0x08, 0x10, 0x20, // \
   0x00, 0x00, 0x41, 0x7F, 0x00, // ]
   0x04, 0x02, 0x01, 0x02, 0x04, // ^
   0x40, 0x40, 0x40, 0x40, 0x40, // _
   0x00, 0x01, 0x02, 0x04, 0x00, // `
   0x20, 0x54, 0x54, 0x54, 0x78, // a
   0x7F, 0x44, 0x44, 0x44, 0x38, // b
   0x38, 0x44, 0x44, 0x44, 0x44, // c
   0x38, 0x44, 0x44, 0x44, 0x7F, // d
   0x38, 0x54, 0x54, 0x54, 0x18, // e
   0x04, 0x04, 0x7E, 0x05, 0x05, // f
   0x08, 0x54, 0x54, 0x54, 0x3C, // g
   0x7F, 0x08, 0x04, 0x04, 0x78, // h
   0x00, 0x44, 0x7D, 0x40, 0x00, // i
   0x20, 0x40, 0x44, 0x3D, 0x00, // j
   0x7F, 0x10, 0x28, 0x44, 0x00, // k
   0x00, 0x41, 0x7F, 0x40, 0x00, // l
   0x7C, 0x04, 0x78, 0x04, 0x78, // m
   0x7C, 0x08, 0x04, 0x04, 0x78, // n
   0x38, 0x44, 0x44, 0x44, 0x38, // o
   0x7C, 0x14, 0x14, 0x14, 0x08, // p
   0x08, 0x14, 0x14, 0x14, 0x7C, // q
   0x00, 0x7C, 0x08, 0x04, 0x04, // r
   0x48, 0x54, 0x54, 0x54, 0x20, // s
   0x04, 0x04, 0x3F, 0x44, 0x44, // t
   0x3C, 0x40, 0x40, 0x20, 0x7C, // u
   0x1C, 0x20, 0x40, 0x20, 0x1C, // v
   0x3C, 0x40, 0x30, 0x40, 0x3C, // w
   0x44, 0x28, 0x10, 0x28, 0x44, // x
   0x0C, 0x50, 0x50, 0x50, 0x3C, // y
   0x44, 0x64, 0x54, 0x4C, 0x44, // z
   0x00, 0x08, 0x36, 0x41, 0x41, //
   0x00, 0x00, 0x7F, 0x00, 0x00, //|
   0x41, 0x41, 0x36, 0x08, 0x00, //
   0x02, 0x01, 0x02, 0x04, 0x02
}; // ~

#define oLED     0x78
unsigned int8 display[1024];
/*****************************************************************/
void oLED_command(unsigned int8 address,unsigned int8 command) {
   i2c_start();
   i2c_write(address);
   i2c_write(0x00);  //the next byte will be command
   i2c_write(command);
   i2c_stop();
}
/*****************************************************************/
void  oLED_init(unsigned int8 address) {
   oLED_command (address, 0xAE) ; // SSH1106_DISPLAYOFF
   oLED_command (address, 0xA8);  // SSH1106_SETMULTIPLEX
   oLED_command (address, 0x3F);
   oLED_command (address, 0xD3); //SSH1106_SETDISPLAYOFFSET
   oLED_command (address, 0x00);  //no offset
   oLED_command (address, 0x40); //SSH1106_SETSTARTLINE
   oLED_command (address, 0xA0); //SSH1106_SEGREMAP
   oLED_command (address, 0xA1); //--set segment re-map 127 to 0
   oLED_command (address, 0xC0); //SSH1106_COMSCANINC
```

```
    oLED_command (address, 0xC8); //SSH1106_COMSCANDEC
    oLED_command (address, 0xDA); //SSH1106_SETCOMPINS
    oLED_command (address, 0x12);
    oLED_command (address, 0x81); //SSH1106_SETCONTRAST
    oLED_command (address, 0xFF);
    oLED_command (address, 0xA4); //SSH1106_DISPLAYALLON_RESUME
    oLED_command (address, 0xA6) ; // SSH1106_NORMALDISPLAY
    oLED_command (address, 0xD5);  //SSH1106_SETDISPLAYCLOCKDIV
    oLED_command (address, 0x80);
    oLED_command (address, 0x8D); //SSH1106_CHARGEPUMP
    oLED_command (address, 0x14) ;   // using internal VCC
    oLED_command (address, 0xAF) ; //SSH1106_DISPLAYON
    oLED_command (address, 0x20) ;  //SSH1106_MEMORYMODE
    oLED_command (address, 0x00) ;  // 0x00 horizontal addressing
}
/*********************************************************************/
void oLED_write(unsigned int8 address) {
    unsigned int16 i;
    oLED_command (address, 0x21); //SSH1106_COLUMNADDR
    oLED_command (address, 0x00);        // Column start address
    oLED_command (address,  127);        // Column end address
    oLED_command (address, 0x22); //SSH1106_PAGEADDR
    oLED_command (address, 0x00);        // Start Page address
    oLED_command (address,    7);        // End Page address
    i2c_start ();
    i2c_write (address) ;
    i2c_write (0x40) ;           //the next byte will be data
        for(i = 0; i < 1024; i++)
        {
           i2c_write(display[i]);
        }
    i2c_stop ();
}
/*********************************************************************/
void oLED_clear() {
    memset(display,0x00,1024);
}
/*********************************************************************/
void oLED_dark() {
    memset(display,0xFF,1024);
}
/*********************************************************************/
void oLED_pixel(int16 x,int16 y)
{
 int16 nt;
 int16 pagina;
 int16 binarydigit;
    pagina = y /8;
    binarydigit= y-(pagina*8);
    nt = display[pagina*128+x];
    nt |= 1 << binarydigit;
    display[pagina*128+x] = nt;
}
/*********************************************************************/
void oLED_text(int x, int y, char* textptr, int size)
{
    unsigned int8 i, j, k, l, m;     // Loop counters
    unsigned int8 pixelData[5];       // Stores character data

    for(i= 0; textptr[i] != '\0'; ++i, ++x) { // Loop through the passed string
       if(textptr[i] < 'S') // Checks if the letter is in the first text array
                memcpy(pixelData, text[textptr[i]-' '], 5);
       else if (textptr[i] <= '~') // Check if the letter is in the second array
                memcpy (pixelData, text2[textptr[i]-'S'], 5) ;
            else
                memcpy (pixelData, text[0], 5); // DEFAULT to space
       if(x + 5 * size >= 128) {  // Performs character wrapping
          x = 0;       // Set x at far left position
```

```
            y += 7*size + 1;     // Set y at next position down
         }
         for(j = 0; j < 5; ++j, x += size) {  // Loop through character byte data
            for(k = 0; k< 7 * size; ++k) {   // Loop through the vertical pixels
               if(bit_test(pixelData[j], k)) { // Check if the pixel should be set
                  for(l = 0; l<size; ++l) {  // The next two loops change the
                     // character's size
                     for(m = 0; m<size; ++m) {
                        oLED_pixel (x + m, y + k * size + l); // Draws the pixel
                     }
                  }
               }
            }
         }
      }
   }

   #endif
```

4.3. Código configuración ATA8520E

```
#include <24FJ128GC006.h>
#include "Header.h"
#use delay(clock= 11059200)
#use rs232 (UART1, baud=115200, stream=BT)
#define SPI_MODE_0  (SPI_L_TO_H | SPI_XMIT_L_TO_H)

// VARIABLES
unsigned int8 Mjvs, Mnvs;
unsigned int8 nPAC[16]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
unsigned int8 nPAC0,nPAC1,nPAC2,nPAC3,nPAC4,nPAC5,nPAC6,nPAC7;
unsigned int8 nPAC8,nPAC9,nPAC10,nPAC11,nPAC12,nPAC13,nPAC14,nPAC15;
unsigned int8 SSM_st,ATM_st,SIG_st1,SIG_st2;
unsigned int8 nID0,nID1,nID2,nID3;
int1 status=1;

void get_status_ATA8520E(void){
    delay_ms(1);
    OFF(CS);
    delay_ms(1);
    spi_write(0x0A);
    spi_write(0x00);
    SSM_st=spi_read(0x00);
    ATM_st=spi_read(0x00);
    SIG_st1=spi_read(0x00);
    SIG_st2=spi_read(0x00);
    ON(CS);
}
void main(void){

int i=0;
ON(PB4);   // Wake-up
OFF(PC1);
delay_ms(1); // Reinicio
OFF(PC0);
delay_ms(10);
ON(PC0);
delay_ms(20);
OFF(CS);
delay_ms(1);

setup_spi(SPI_MASTER|SPI_SS_DISABLED|SPI_MODE_0|SPI_CLK_DIV_256);
spi_write(0x06);   // Leer versión SigFox
spi_write(0x00);
Mjvs=spi_read(0x00);
Mnvs=spi_read(0x00);
get_status_ATA8520E();
delay_ms(100);      //TX and RX configure
```

```
ON(CS);
delay_ms(100);
OFF(CS);
delay_ms(100);
spi_write(0x1A);
spi_write(0xD0);    //TX
spi_write(0x9C);
spi_write(0xBE);
spi_write(0x33);
spi_write(0x08);    //RX
spi_write(0xE6);
spi_write(0xD3);
spi_write(0x33);

while(input(PB6)!=0);
get_status_ATA8520E();
delay_ms(100);       //Syst. Config
ON(CS);
delay_ms(100);
OFF(CS);
delay_ms(100);
spi_write(0x11);
spi_write(0x00);
spi_write(0x01);
spi_write(0x02);
spi_write(0x3B);  // EU mode  uplink and downlink
while(input(PB6)!=0);
get_status_ATA8520E();
delay_ms(100);       //RSSI Conf
ON(CS);
delay_ms(100);
OFF(CS);
delay_ms(100);
spi_write(0x24);
spi_write(0x0A);  // 5dB=10=0x0A
while(input(PB6)!=0);
get_status_ATA8520E();
delay_ms(100);       //RESET
ON(CS);
delay_ms(100);
OFF(CS);
delay_ms(100);
spi_write(0x01);

while(input(PB6)!=0);
get_status_ATA8520E();

while(1){

       ON(LD2);
       delay_ms(1000);
}

}
```

4.4. Código principal

```
#include <24FJ128GC006.h>
#include "Header.h"
#device ADC=12  //Para la ADC de pipeline
#include <string.h>
#use delay(clock= 11059200)

/*********************************************************************
                          DEFINICION DE REGISTROS
*********************************************************************/
```

```
//--------------------------TYPE OF RESET-------------------------

#word RCON = 0x740
//--------------------------ADC PIPELINE-------------------------

#word REF_BUF1 = 0x0672 // Addres register refer. Volt. BUF1 to pipeline
//--------------------------ADC SigDelta 16 bits-------------------------

#word REF_BUF2 = 0x0674   // Addres register reference voltage BUF2
#word IFS6    = 0x0090 //interrupccion fin
#word SD1RESH = 0x04D6 // Valor ADC
/*********************************************************************
                                DEFINICIÓN DE INTERFACES
*********************************************************************/
//--------------------------I2C OLED-------------------------
#use i2c(Master,Fast=400000, i2c1)
#include "OLED.c"
//--------------------------UART BLUETOOTH-------------------------

#pin_select U1TX =  BT_TX
#pin_select U1RX =  BT_RX
#use rs232 (UART1, baud=115200, stream=BT)

// -------------------------SPI SIGFOX MODULE ATA8520E-------------------------

#pin_select SDI1 = SIGFOX_SDI
#pin_select SDO1 = SIGFOX_SDO
#pin_select SCK1OUT = SIGFOX_SCLK

#define SPI_MODE_0  (SPI_L_TO_H | SPI_XMIT_L_TO_H)
#define SPI_MODE_1  (SPI_L_TO_H)
#define SPI_MODE_2  (SPI_H_TO_L)
#define SPI_MODE_3  (SPI_H_TO_L | SPI_XMIT_L_TO_H)
// -------------------------ADC SigDelta 16 bits-------------------------

#define SD_CON1  SDADC_ENABLED|SDADC_GAIN_1|SDADC_DITHER_HIGH
|SDADC_SVREF_SVSS|SDADC_BW_NORMAL
#define SD_CON2  SDADC_CHOPPING_ENABLED|SDADC_INT_EVERY_SAMPLE|
SDADC_RES_UPDATED_EVERY_INT |SDADC_ROUND_16_BITS
#define SD_CON3  SDADC_CLOCK_DIV_2|SDADC_OSR_1024|SDADC_CLK_FRC


/*********************************************************************
                                VARIABLES GLOBALES
*********************************************************************/

//--------------------------SAVE FLASH-------------------------

unsigned int32 orgUmb = 0x00010000;
//--------------------------CONTADORES DE INTERRUPCION-------------------------
int16 count_day=0, count_to_measure=0, num_measures=0;
//----------------------PARAMETROS DE CONFIGURACION-------------------------
int16 int_day=1440,int_to_measure=2,measures_to_send=6;
unsigned int8 min_alarm =0, max_alarm=255;
unsigned int8 warm_time=25;
int1 bool_select_4_20=0;
float ganancia=1,offset=0;
int1 debug=0;
//----------------------BUFFER DE MEDICIONES A ENVIAR-------------------------
int16 buff_measures[840];
//----------------------AUXILIAR CONVERT FLOAT TO 4 BYTES----------------------
union {
    float f;
    unsigned int8 b[4];
}y;
//----------------------CALIBRACION ADC SD-------------------------

int16 SDerrOffset, SDmaxValue;
```

```
//----------------------------RTC--------------------------------

rtc_time_t dtm,alm;
//---------------------SIGFOX ATA8520E---------------------------

unsigned int8 SSM_st,ATM_st,SIG_st1,SIG_st2;
unsigned int8 rx_SigF_buffer[8];
unsigned int8 Mjvs, Mnvs,nID3,nID2,nID1,nID0,nPAC[16];

/*****************************************************************************
                                FUNCIONES AUXILIARES
*****************************************************************************/
//---------------------------------------------------------------------
//                       PANTALLA OLED
//---------------------------------------------------------------------

void Print_measure_oLED(float medida){

    char txt[100];
    oLED_clear();
    oLED_write(oLED);
    sprintf(txt,"Medida del sensor");
    oLED_text(0,0,txt,1);
    sprintf(txt,"%.3f mmHg",medida);
    oLED_text(0,20,txt,2);
    sprintf(txt,"E.T.S.I.I. - U.P.V.");
    oLED_text(10,50,txt,1);
    oLED_write(oLED);
}

//---------------------------------------------------------------------
//                       ACTUALIZAR CONFIGURACION
//---------------------------------------------------------------------

void establish_configuration(unsigned int8 config[8]){

    unsigned int32 aux1,aux2;
    aux1= (((unsigned int32)config[0])<<24) + ( ((unsigned int32)config[1])<<16)
+ ( ((unsigned int32)config[2])<<8)+ (((unsigned int32)config[3]));
    aux2= (((unsigned int32)config[4])<<24) + ( ((unsigned int32)config[5])<<16)
+ ( ((unsigned int32)config[6])<<8)+ (((unsigned int32)config[7]));

    //AUX1
    int_to_measure=((aux1&0xFFE00000)>>21);
    measures_to_send=((aux1&0x001FF800)>>11);
    int_day=((aux1&0x000007FF));

    //AUX2
    bool_select_4_20=(aux2>>31);
    min_alarm=((aux2&0x7F800000)>>23);
    max_alarm=((aux2&0x007F8000)>>15);
    warm_time=((aux2&0x00007F80)>>7);
}

//---------------------------------------------------------------------
//                       WRITE/READ IN PROGRAM MEMORY
//---------------------------------------------------------------------

void read_configuration_flash(void){
    unsigned int8 config_flash[24];
    unsigned int32 aux[2];

    read_program_memory(orgUmb, config_flash, 24);
        aux[0] =
make32(config_flash[0],config_flash[1],config_flash[2],config_flash[4]);
        aux[1] =
make32(config_flash[5],config_flash[6],config_flash[8],config_flash[9]);
```

```
        int_to_measure=((aux[0]&0xFFE00000)>>21);
        measures_to_send=((aux[0]&0x001FF800)>>11);
        int_day=((aux[0]&0x000007FF));
        bool_select_4_20=(aux[1]>>31);
        min_alarm=((aux[1]&0x7F800000)>>23);
        max_alarm=((aux[1]&0x007F8000)>>15);
        warm_time=((aux[1]&0x00007F80)>>7);

        y.b[0]=config_flash[10];
        y.b[1]=config_flash[12];
        y.b[2]=config_flash[13];
        y.b[3]=config_flash[14];

        ganancia=y.f;

        y.b[0]=config_flash[16];
        y.b[1]=config_flash[17];
        y.b[2]=config_flash[18];
        y.b[3]=config_flash[20];
        offset=y.f;
}
void write_configuration_flash(void){
    unsigned int8 config_flash[24];
    unsigned int32 aux[2];

    aux[0]=  (((unsigned int32) int_to_measure)<<21)+(((unsigned int32)
measures_to_send)<<11)+((unsigned int32) int_day);
    aux[1]=  (((unsigned int32)  bool_select_4_20)<<31) + (((unsigned int32)
min_alarm)<<23)+(((unsigned int32) max_alarm)<<15)+(((unsigned int32)
warm_time)<<7);

    config_flash[0]= (aux[0]&0xFF000000)>>24;  //=make8(aux,3);
    config_flash[1]= (aux[0]&0x00FF0000)>>16;
    config_flash[2]= (aux[0]&0x0000FF00)>>8;
    config_flash[3]=0x00;
    config_flash[4]= aux[0]&0x000000FF;
    config_flash[5]=(aux[1]&0xFF000000)>>24;
    config_flash[6]=(aux[1]&0x00FF0000)>>16;
    config_flash[7]=0x00;
    config_flash[8]=(aux[1]&0x0000FF00)>>8;
    config_flash[9]=aux[1]&0x000000FF;

    y.f=ganancia;

    config_flash[10]=y.b[0];
    config_flash[11]=0x00;
    config_flash[12]=y.b[1];
    config_flash[13]=y.b[2];
    config_flash[14]=y.b[3];

    y.f=offset;
    config_flash[15]=0x00;
    config_flash[16]=y.b[0];
    config_flash[17]=y.b[1];
    config_flash[18]=y.b[2];
    config_flash[19]=0x00;
    config_flash[20]=y.b[3];

    erase_program_memory(orgUmb);
    write_program_memory(orgUmb,  config_flash, 24);
}


//-------------------------------------------------------------------
//                      BLUETOOTH AUXILIAR FUNCTIONS
//-------------------------------------------------------------------

unsigned int8 RSP(unsigned int16 ms){
```

```
    unsigned int32 timeout;
    unsigned int8 rcv[256];
    unsigned int8 cc;
    memset( rcv, 0, 256);
    cc=0;
    trama:
        timeout= (ms>0) ? (ms*100) : 1;
        while(!kbhit(BT) && ((--timeout)>0))     delay_us(10);
        if(kbhit(BT)) rcv[cc++]=getch(BT);
        else return cc;
    goto trama;
}

inline void configuracion_inicial_BT(void){
    unsigned int8 n;
    ON(ON_BT);
    delay_ms(2000);
    if(debug){
        ON(LD2);OFF(LD1);delay_ms(200);
        OFF(LD2);ON(LD1);delay_ms(200);
        ON(LD2);OFF(LD1);delay_ms(200);
        OFF(LD2);ON(LD1);delay_ms(200);
        ON(LD2);OFF(LD1);delay_ms(200);
    }
    n = RSP(500);
    fprintf(BT,"$$$");
    n = RSP(500);
    fprintf(BT,"SF,1\r");
    n = RSP(500);
    fprintf(BT,"SA,0\r");
    n = RSP(500);
    fprintf(BT,"SN,MiniFOX\r");
    n = RSP(500);
    fprintf(BT,"R,1\r");
    n = RSP(500);
}

//-----------------------------------------------------------------------
//                       ADC SIGMA DELTA 16 BITS: SENSOR
//-----------------------------------------------------------------------

int16 read_SDadc(void){
    for(unsigned int8 i= 0; i<6; i++){
                bit_clear(IFS6,9);
                while(!bit_test(IFS6,9));
            }
    return SD1RESH;
}
void inicializacion_calibracion_ADC_SD(void){
   REF_BUF2=0xB083; // Configuracion a 3.072 V y salida pin 4
   delay_ms(100);
   set_sd_adc_channel(0);
   setup_SD_adc(SD_CON1, SD_CON2,SD_CON3); //Configure ADC SD 16 bits
   set_sd_adc_calibration(SDADC_START_CALIBRATION_MODE);
   SDerrOffset = read_SDadc();
   set_sd_adc_calibration(SDADC_END_CALIBRATION_MODE);
   set_sd_adc_channel(SDADC_REFERENCE);
   SDmaxValue  = read_SDadc()-SDerrOffset;
   set_sd_adc_channel(0);

}
void sleep_ADC_SD(void){
   REF_BUF2=0x3043; //Apagar
   setup_SD_adc(SDADC_DISABLED,0,0); //Configure ADC SD 16 bits
}
int16 LeerSensor_ADC(void){
  int32 tmp;
```

```
  int16 tmp16;
  int16 SDvalor;
  SDvalor = read_SDadc();
  tmp16= SDvalor - SDerrOffset;
  tmp=((int32)(SDvalor - SDerrOffset))*0x7FFF/SDmaxValue;
      if(tmp >  32767) tmp=32767;
      else if(tmp < -32768) tmp= -32768;
  return (int16) tmp;;
}
float LeerSensor_P(void){
  float V,P;
  int16 SDvalor;
  SDvalor = LeerSensor_ADC();
  V = ((float)(3.072/0x7FFF))*SDvalor;
  if(bool_select_4_20) V =((V/60.0)*1000.0);
  else V=V*(500.0/60.0);
  P=ganancia*V+offset;
  return P;
}

//-------------------------------------------------------------------
//                      ADC PIPELINE 12 BITS: BATTERY
//-------------------------------------------------------------------

void inicializacion_adc_pipeline(void){
    REF_BUF1=0xB003; // Configuracion a 3.072 V interna
    setup_adc_ports(sAN11,VSS_INTREF);
    setup_adc(ADC_CLOCK_DIV_32);
    set_adc_channel(11);
}

void sleep_adc_pipeline(void){
    REF_BUF1=0x3043; // Apagar
    setup_adc(ADC_OFF);
}
int16 read_battery_level_ADC(void){
    int32 Lvalor;
    int16 level;

    Lvalor=0;
    for(int16 c=0;c<256;c++)
        {
            while(!adc_done());
            Lvalor+=read_adc();
        }
    level=(unsigned int16)((Lvalor>>8)&0x0FFF);
    return level;
}
float read_battery_level(void){
    int16 level;
    float level_V;
    level=read_battery_level_ADC();
    level_V=((float) level)*(3.072/(0x0FFF));
    level_V=level_V*2;
    return level_V;
}

//-------------------------------------------------------------------
//                      SIGFOX MODULE ATA8520E
//-------------------------------------------------------------------

void wake_up_and_reset_ATA8520E(void){
    ON(CS); // asegurar que no esta seleccionado
    ON(PB4); OFF(PC1);  // wake-up
    delay_ms(1);
    OFF(PC0);    // RESET
    delay_ms(10);
    ON(PC0);
```

```
    delay_ms(20);
    OFF(PB4);ON(PC1); //Para no volverlo a despertar
}
void get_status_ATA8520E(void){
    delay_ms(1);
    OFF(CS);
    delay_ms(1);
    spi_write(0x0A);
    spi_write(0x00);
    SSM_st=spi_read(0x00);
    ATM_st=spi_read(0x00);
    SIG_st1=spi_read(0x00);
    SIG_st2=spi_read(0x00);
    ON(CS);
}
void Load_TX_Buffer(unsigned int8 *m,unsigned int8 size){
    delay_ms(1);        //LOAD TX BUFFER
    OFF(CS);
    delay_ms(1);
    spi_write(0x07);
    spi_write(size);  // N bytes
    for (int i=0;i<size;i++){
        spi_write(m[i]);
    }
    ON(CS);
}

void Read_RX_Buffer(void){
    delay_ms(1);        //READ TX BUFFER
    OFF(CS);
    delay_ms(1);
    spi_write(0x10);
    spi_write(0x00);
    for(int i=0;i<8;i++){
        rx_SigF_buffer[i]=spi_read(0x00);
    }
    ON(CS);
}
void Send_TX_Buffer(void){
    delay_ms(1);        //SEND DATA
    OFF(CS);
    delay_ms(1);
    spi_write(0x0D);
    delay_ms(100);
    while(input(PB6)!=0);  //Wait to finish
    ON(CS);
    get_status_ATA8520E();
}
void Send_TX_Buffer_and_Receive(void){
    delay_ms(1);        //SEND DATA
    OFF(CS);
    delay_ms(1);
    spi_write(0x0E);
    delay_ms(100);
    while(input(PB6)!=0);  //Wait to finish
    ON(CS);
    get_status_ATA8520E();
}
void envio_buff_ATA8520E(int16 *buff_measures,int16 num_measures){
    int16 j=0,i=0;
    unsigned int8 x;
    unsigned int8 bufferByte[12];

    while(j<num_measures){
        x=((num_measures-j)>=6)?6:(num_measures-j);
        for (i=0;i<x;i++){
            bufferByte[2*i]= (int8) ((buff_measures[j+i]& 0xFF00)>>8);
            bufferByte[2*i+1]= (int8) (buff_measures[j+i]& 0x00FF);
```

```
            }
            j=j+i;
            Load_TX_Buffer(&bufferByte,x*2);
            Send_TX_Buffer();
        }
    }
    int1 envio_y_recepcion_ATA8520E(unsigned int16 battery){
        unsigned int8 msg[4];
        int1 ok=0;
        msg[0]=0x00;
        msg[1]=0xCC;
        msg[2]=make8(battery,1);
        msg[3]=make8(battery,0);
        delay_ms(1);
        Load_TX_Buffer(&msg,4);
        Send_TX_Buffer_and_Receive();

        if(SIG_st1!=0x3E){  //If data is received
                Read_RX_Buffer();
                ok=1;
            }
    return ok;
    }
    void enviar_calibracion_ATA8520(float m,float n){
        unsigned int8 msg[11];
        msg[0]=0x00;
        msg[1]=0x00;
        msg[2]=0xFF;  //Calibration identifier
        y.f=m;
        msg[3]=y.b[3];
        msg[4]=y.b[2];
        msg[5]=y.b[1];
        msg[6]=y.b[0];
        y.f=n;
        msg[7]=y.b[3];
        msg[8]=y.b[2];
        msg[9]=y.b[1];
        msg[10]=y.b[0];
        Load_TX_Buffer(&msg,11);
        Send_TX_Buffer();
    }

    void envio_alarma_ATA8520E(unsigned int16 measure, int1 type){
        unsigned int8 msg[5];
        msg[0]=0x00;
        msg[1]=0x00;
        msg[2]=(unsigned int8) type;
        msg[3]=make8(measure,1);
        msg[4]=make8(measure,0);
        Load_TX_Buffer(&msg,5);
        Send_TX_Buffer();
    }
    void sleep_ATA8520E(void){
        OFF(PB4);ON(PC1); //Para asegurar no volverlo a despertar
        delay_ms(1);
        OFF(CS);
        delay_ms(1);
        spi_write(0x05);
        ON(CS);
    }
    void identifier_ATA8520E(void){
        OFF(CS);
        delay_ms(1);
        spi_write(0x06);
        spi_write(0x00);
        Mjvs=spi_read(0x00);
        Mnvs=spi_read(0x00);
        ON(CS);
```

```
    delay_ms(1);
    OFF(CS);
    delay_ms(1);
    spi_write(0x12);
    spi_write(0x00);
    nID0=spi_read(0x00);
    nID1=spi_read(0x00);
    nID2=spi_read(0x00);
    nID3=spi_read(0x00);
    ON(CS);
    delay_ms(1);
    OFF(CS);
    delay_ms(1);
    spi_write(0x0F);
    spi_write(0x00);
    for(int i=0;i<16;i++){
        nPAC[i]=spi_read(0x00);
    }
    ON(CS);

}

/*************************************************************************
                    FUNCIONES PRINCIPALES DE EJECUCION
*************************************************************************/

//-----------------------------------------------------------------
//                      SECUENCIA DE MEDICION Y ENVIO
//-----------------------------------------------------------------

#INT_RTC
void RTC_isr(){  //Medicion del sensor y envio de datos
    int16 measure;

    inicializacion_calibracion_ADC_SD();  //Utilizada para leer del sensor
    inicializacion_adc_pipeline();        //Utilizada para leer el nivel de
bateria

    if(debug) output_toggle(LD2);

    count_day++;
    count_to_measure++;
    if(count_day==int_day) reset_cpu();  //Reset

    if(bool_select_4_20) ON(Ac_4_20);   //Primero ajusto la salida por seguridad
    else OFF(Ac_4_20);
    delay_ms(10);
    ON(ON_SNS);  //Enciendo el sensor
    delay_ms((((unsigned int16)warm_time)*10+1));
    measure=LeerSensor_ADC();
    OFF(ON_SNS); //Apago sensor

    sleep_adc_pipeline();
    sleep_ADC_SD();

    if((measure>>8)<=min_alarm){
        wake_up_and_reset_ATA8520E();
        while(input(PB6)!=0);  //Espero a que se despierte
        get_status_ATA8520E();
        envio_alarma_ATA8520E(measure,0);  // 0 < umbral y 1 > umbral
        sleep_ATA8520E();
    }
    else if((measure>>8)>=max_alarm){
        wake_up_and_reset_ATA8520E();
        while(input(PB6)!=0);  //Espero a que se despierte
        get_status_ATA8520E();
        envio_alarma_ATA8520E(measure,1);
        sleep_ATA8520E();
```

```
        }

    if(count_to_measure==int_to_measure){

        count_to_measure=0;
        buff_measures[num_measures]=measure;
        num_measures++;
        if(num_measures==measures_to_send){
            num_measures=0;
            wake_up_and_reset_ATA8520E();
            while(input(PB6)!=0);  //Espero a que se despierte
            get_status_ATA8520E();
            if(debug) ON(LD1);
            envio_buff_ATA8520E(&buff_measures,measures_to_send);
            if(debug) OFF(LD1);
            sleep_ATA8520E();
            }
    }

}

//-------------------------------------------------------------------
//                      CONFIGURACION AUTOMATICA
//-------------------------------------------------------------------

void configuracion_automatica(void){

    int8 k=0;
    int16 bat_level;
    OFF(GN);
    delay_ms(1);
    bat_level=read_battery_level_ADC;
    output_float(GN);
    if(debug){
        ON(LD2);OFF(LD1);
        delay_ms(100);
        while(k<10){
            output_toggle(LD2); output_toggle(LD1);
            delay_ms(100);
            k++;
        }
        OFF(LD1),OFF(LD2);
    }

    wake_up_and_reset_ATA8520E();
    while(input(PB6)!=0);  //Espero a que se despierte
    get_status_ATA8520E();
    if(envio_y_recepcion_ATA8520E(bat_level)){
        establish_configuration(rx_SigF_buffer);
        write_configuration_flash();
        fprintf(BT,"Se ha recibido\n\r");
    }
}

//-------------------------------------------------------------------
//                      MENU CONFIGURACION POR BLUETOOTH
//-------------------------------------------------------------------

void configuration_mode(void){

    char lectura_BT;
    unsigned int8 config[8];
    float P,bat;
    unsigned int1 on_display=0,on_conf_aut=0;
    unsigned int8 i=0,k=0;
    unsigned int1 measure_ON =0,display_ON=0;
    unsigned int16 time_ms=0;
    float min_alar_float,max_alar_float;
```

```
    float x;
    float Xi[100],Yi[100];
    double Sx,Sy,Sxx,Sxy,m,n;

    if(debug){
        ON(LD1);delay_ms(500);
        OFF(LD1);delay_ms(100);
        ON(LD1);delay_ms(500);
        OFF(LD1);delay_ms(100);
        ON(LD1);delay_ms(500);
        OFF(LD1);delay_ms(100);
    }

    configuracion_inicial_BT();
    delay_ms(1000);
    if(debug){
        OFF(LD1);
        delay_ms(1000);
        ON(LD1);
    }
    time_ms=0;
    while(!input(COMM)){    // Esperar a la conexion
        if(!input(INT_REED_1)){
            on_display=1;
            goto end_bluetooth;
        }
        if(!input(INT_REED_2)){
            on_conf_aut=1;
            goto end_bluetooth;
        }
        delay_ms(1);
        time_ms++;
        if(time_ms>30000) goto end_bluetooth;

    }
    fprintf(BT,"Nodo sigfox conectado\n\r");
    min_alar_float=( ((float) (((unsigned int16)min_alarm)<<8) )
       *((bool_select_4_20)?1000.0/60.0:500.0/60.0)
       *( 3.072 / ((float)0x7FFF) )*ganancia +offset);
    max_alar_float=( ((float) (((unsigned int16)max_alarm)<<8) )
       *((bool_select_4_20)?1000.0/60.0:500.0/60.0)
       *( 3.072 / ((float)0x7FFF) )*ganancia +offset);

if(debug){
fprintf(BT,"%u,%u,%u,%f,%f,%f,%u,%f,%u,%u\n\r",int_to_measure,measures_to_send,b
ool_select_4_20,ganancia,offset,min_alar_float,min_alarm,
max_alar_float,max_alarm,warm_time*10);
}
fprintf(BT,"$,%u,%u,%u,%f,%f,%f,%f,%u\n\r",int_to_measure,measures_to_send,bool_
select_4_20,ganancia,offset,min_alar_float,max_alar_float,warm_time*10);

    delay_ms(1000);
    while(input(COMM)){
        if(!input(INT_REED_1)){
            on_display=1;
            break;
        }
        if(!input(INT_REED_2)){
            on_conf_aut=1;
            break;
        }
        if(kbhit(BT))    {
            lectura_BT=getch(BT);
            delay_ms(10);
            if(lectura_BT=='C'){
                i=0;
                while(i<8){
                    if(kbhit(BT)){
```

```
                config[i]=getch(BT);
                i++;
            }
        }
        fprintf(BT,"Configuracion recibida:\n\r");
        establish_configuration(config);
        fprintf(BT,"%d, %d, %d, %d, %d, %d\n\r",int_to_measure,
        measures_to_send, int_day,bool_select_4_20,min_alarm,max_alarm);
        write_configuration_flash();
    }
    else if(lectura_BT=='w'){
        while(!kbhit(BT));
        warm_time=getch(BT);
        fprintf(BT,"Ajustado el warm time a: %d ms\n\r",warm_time*10+1);
    }
    else if(lectura_BT=='G'){
        i=0;
        while(i<4){
            if(kbhit(BT)){
                y.b[i]=getch(BT);
                i++;
            }
        }
        fprintf(BT,"Valor recibido:%x %x %x %x\n\r",
                            y.b[0],y.b[1],y.b[2],y.b[3]);
        fprintf(BT,"Reajustada Ganancia:%f\n\r",y.f);
        ganancia=y.f;
    }
    else if(lectura_BT=='N'){

        i=0;
        while(i<4){
            if(kbhit(BT)){
                y.b[i]=getch(BT);
                i++;
            }
        }
        fprintf(BT,"Valor recibido:%x %x %x %x\n\r",
                             y.b[0],y.b[1],y.b[2],y.b[3]);
        fprintf(BT,"Reajustado Offset:%f\n\r",y.f);
        offset=y.f;
    }
    else if(lectura_BT=='L'){
        fprintf(BT,"Configurando alarmas:\n\r");
        i=0;
        while(i<4){
            if(kbhit(BT)){
                y.b[i]=getch(BT);
                i++;
            }
        }
        min_alar_float=y.f;
        i=0;
        while(i<4){
            if(kbhit(BT)){
                y.b[i]=getch(BT);
                i++;
            }
        }
        max_alar_float=y.f;
        fprintf(BT,"Ajustando alarmas para Ganancia %f y Offset %f\n\r"
                ,ganancia,offset);
        fprintf(BT,"Umbrales: %f  %f\n\r"
                ,min_alar_float,max_alar_float);
        min_alar_float=(min_alar_float-offset)/ganancia;// en V/mA Sens
        max_alar_float=(max_alar_float-offset)/ganancia;//
        fprintf(BT,"Umbrales salida sensor: %f  %f\n\r"
                ,min_alar_float,max_alar_float);
```

```
        if(bool_select_4_20){       // convertir a V a la entrada ADC
            min_alar_float=(min_alar_float)*60.0/1000.0;
            max_alar_float=(max_alar_float)*60.0/1000.0;
        }
        else{
            min_alar_float=(min_alar_float)*60.0/500.0;
            max_alar_float=(max_alar_float)*60.0/500.0;
        }
        fprintf(BT,"Umbrales entrada ADC (V): %f  %f\n\r"
               ,min_alar_float,max_alar_float);
        // convertir a bit: 0x7FFF=3.072 V
        min_alar_float= min_alar_float*((0x7FFF)/3.072);
        max_alar_float= max_alar_float*((0x7FFF)/3.072);
        fprintf(BT,"Umbrales entrada ADC (bits): %f  %f\n\r"
               ,min_alar_float,max_alar_float);
        // Cambiar la resolucion de 16 a 8 bits
        min_alarm=(unsigned int8) (min_alar_float/256.0);
        if(min_alarm<(min_alar_float*256.0)) min_alarm++;
        max_alarm=(unsigned int8) (max_alar_float/256.0);
        fprintf(BT,"Umbrales entrada ADC (bits): %u  %u\n\r"
               ,min_alarm,max_alarm);
    }
    else if(lectura_BT=='P'){
        fprintf(BT,"Configuracion Actual:\n\r");
        fprintf(BT,"Frecuencia %u minutos\n\rTamano paquete: %u
                    medidas/envio\n\r",int_to_measure,measures_to_send);
        (bool_select_4_20)?fprintf(BT,"Modo corriente seleccionado\n\r")
                          :fprintf(BT,"Modo tension seleccionado\n\r");
        fprintf(BT,"Warm time: %u ms\n\r",warm_time*10+1);
        fprintf(BT,"Ganancia: %f",ganancia);
        (bool_select_4_20)?fprintf(BT,"mmHg/mA\n\r")
                          :fprintf(BT,"mmHg/V\n\r");
        fprintf(BT,"Offset: %f mmHg\n\r",offset);
        fprintf(BT,"Min Alarmas: %f mmHg\n\r",
          ( ((float) (((unsigned int16)min_alarm)<<8) )*
          ((bool_select_4_20)?1000.0/60.0:500.0/60.0)*
          ( 3.072 / ((float)0x7FFF) )*ganancia +offset));
        fprintf(BT,"Max Alarmas: %f mmHg\n\r",
          ( ((float) (((unsigned int16)max_alarm)<<8) )*
          ((bool_select_4_20)?1000.0/60.0:500.0/60.0)*
          ( 3.072 / ((float)0x7FFF) )*ganancia +offset));
    }
    else if(lectura_BT=='R'){
        read_configuration_flash();
        fprintf(BT,"Leida configuracion de la flash:\n\r");
    }
    else if(lectura_BT=='W'){
        write_configuration_flash();
        fprintf(BT,"Escrita configuracion en la flash:\n\r");
    }
    else if(lectura_BT=='+'){
        i=0;
        while(i<4){
            if(kbhit(BT)){
                y.b[i]=getch(BT);
                i++;
            }
        }

        if(k<99){
            x=LeerSensor_P();
            fprintf(BT,"Punto anadido: [%.4f,%.4f]\n\r",x,y.f);
            fprintf(BT,"+[%.4f,%.4f]\n\r",x,y.f);
            Xi[k]=x;
            Yi[k]=y.f;
            k++;
        }
        else fprintf(BT,"Error: Alcazado numero mḁmo\n\r");
```

```
}
else if(lectura_BT=='-'){
    if(k>0){
     fprintf(BT,"Punto eliminado: [%.4f,%.4f]\n\r",Xi[k-1],Yi[k-1]);
     fprintf(BT,"--\n\r");
     k--;
    }
}

else if(lectura_BT=='l'){
    fprintf(BT,"Listado de puntos:\n\r");
        for(i=0; i<k; i++){
          fprintf(BT,"[%.4f,%.4f]\n\r",Xi[i],Yi[i]);
        }
}

else if(lectura_BT=='f'){
    Sx=0;
    Sy=0;
    Sxx=0;
    Sxy=0;
    for(i=0; i<k; i++){
        Sx += Xi[i];
        Sy += Yi[i];
        Sxx += Xi[i]*Xi[i];
        Sxy += Xi[i]*Yi[i];
    }
    m= (k*Sxy - Sx*Sy ) /(k*Sxx -Sx*Sx);
    n= (Sxx*Sy - Sx*Sxy ) /(k*Sxx -Sx*Sx);
    fprintf(BT,"Calibrado: m= %f  n= %f\n\r",m,n);
}

else if(lectura_BT=='s'){

 ganancia = (float) m;
 offset= (float) n;
 fprintf(BT,"Actualizado: Ganancia : %f Offset:
           %f\n\r",ganancia,offset);
 printf(BT,"Enviando nueva calibraci□.\n\r");
 wake_up_and_reset_ATA8520E();
 while(input(PB6)!=0);  //Espero a que se despierte
 get_status_ATA8520E();
 enviar_calibracion_ATA8520(ganancia,offset);
 sleep_ATA8520E();
 fprintf(BT,"Enviada\n\r");
}

else if((lectura_BT=='M')&&(measure_ON==0)){
    fprintf(BT,"Realizando mediciones:\n\r");
    //Primero ajusto la salida por seguridad
    (bool_select_4_20)?ON(Ac_4_20):OFF(Ac_4_20);
    delay_ms(10);
    ON(ON_SNS); OFF(GN);  //ON el sensor y el divisor de la bateria
    delay_ms(200);
    fprintf(BT,"Sensor encendido:\n\r");
    measure_ON=1; // Start to measure
    }

else if((lectura_BT=='F')&&(measure_ON==1)){
    measure_ON=0;  // Stop to measure
    OFF(ON_SNS);output_float(GN);//OFF sensor y divisor de bateria
    fprintf(BT,"Sensor apagado:\n\r");
}
else if(lectura_BT=='1'){

    wake_up_and_reset_ATA8520E();
    identifier_ATA8520E();
    delay_ms(1);
```

```
        sleep_ATA8520E();
        fprintf(BT,"Sigfox Version: %d.%d\n",Mjvs,Mnvs);
        fprintf(BT,"ID: %x%x%x%x\n",nID3,nID2,nID1,nID0);
        fprintf(BT,"PAC:");
        for(i=0;i<8;i++){
            fprintf(BT,"%x",nPAC[i]);}
        fprintf(BT,"\n\r");
    }
    else if(lectura_BT=='2'){
        fprintf(BT,"Prueba cobertura:\n\r");
        unsigned int8 msg[]={0xCA,0xFE,0xCA,0xFE,0xCA,
                             0xFE,0xCA,0xFE,0xCA,0xFE,0x00};
        wake_up_and_reset_ATA8520E();
        while(input(PB6)!=0);  //Espero a que se despierte
        get_status_ATA8520E();
        Load_TX_Buffer(&msg,11);
        Send_TX_Buffer();
        sleep_ATA8520E();
        fprintf(BT,"Enviado\n\r");

    }
    else if(lectura_BT=='D'){
        if(debug){
        debug=0;
        fprintf(BT,"Modo Debug: Desactivado:\n\r");
        }
        else{
        debug=1;
        fprintf(BT,"Modo Debug Activado:\n\r");
        }

    }
    else if(lectura_BT=='O'){
        display_ON=1;

        fprintf(BT,"Encendiendo OLED:\n\r");
        oLED_init(oLED);
        oLED_dark();
        oLED_write(oLED);
        fprintf(BT,"Realizando mediciones:\n\r");
        //Primero ajusto la salida por seguridad
        (bool_select_4_20)?ON(Ac_4_20):OFF(Ac_4_20);
        delay_ms(10);
        ON(ON_SNS);  //Enciendo el sensor
        delay_ms(500);
        fprintf(BT,"Sensor encendido:\n\r");

        P=LeerSensor_P();
        Print_measure_oLED(P);
    }
    else if(lectura_BT=='X'){
        display_ON=0;
        fprintf(BT,"Apagando OLED:\n\r");
        OFF(ON_SNS); // Apago sensor
        fprintf(BT,"Sensor apagado:\n\r");
        oLED_clear();
        oLED_write(oLED);
    }
    else if(lectura_BT=='A'){
        configuracion_automatica();
        fprintf(BT,"Terminado:\n\r");
        fprintf(BT,"SSM: %x ATM: %x SIG1: %x SIG2: %x\n\r",
                    SSM_st,ATM_st,SIG_st1,SIG_st2);
        fprintf(BT,"RX: %x %x %x %x ",rx_SigF_buffer[0]
               ,rx_SigF_buffer[1],rx_SigF_buffer[2],rx_SigF_buffer[3]);
        fprintf(BT,"%x %x %x %x\n\r",rx_SigF_buffer[4],
                 rx_SigF_buffer[5],rx_SigF_buffer[6],rx_SigF_buffer[7]);
    }
```

```
            else if(lectura_BT=='S'){
                fprintf(BT,"Saliendo...:\n\r");
                delay_ms(1000);
                break;
            }
        }
        if((measure_ON)||(display_ON)){
            P=LeerSensor_P();
            if(measure_ON){
                bat=read_battery_level();
                fprintf(BT,"*%f,%f,\n\r",P,bat);
            }
            if(display_ON){
                Print_measure_oLED(P);
            }
        }
    }
    end_bluetooth:
    OFF(LD1);
    OFF(ON_BT); // Apago el BT

    if(on_conf_aut)  configuracion_automatica();
    if(on_display){
        (bool_select_4_20)?ON(Ac_4_20):OFF(Ac_4_20);//Ajuste salida
        delay_ms(10);
        ON(ON_SNS); OFF(GN);  //Enciendo el sensor y el divisor de la bateria
        oLED_init(oLED);
        oLED_dark();
        oLED_write(oLED);
        delay_ms(3000);
        while(input(INT_REED_1)){
            P=LeerSensor_P();
            oLED_clear();
            oLED_write(oLED);
            Print_measure_oLED(P);
            delay_ms(300);
        }
        oLED_clear();
        oLED_write(oLED);
    }

    OFF(ON_SNS);output_float(GN); // Apago sensor y divisor de la bateria

}


/*************************************************************************
*
                                FUNCION MAIN
*
*************************************************************************/


void main(void){

    // ************Configuracion y estado Inicial*************************
    //LEDS
    OFF(LD1);OFF(LD2);

    //Bluetooth
    OFF(ON_BT); // Lo mantengo apagado

    //Interruptores magneticos
    set_pullup(true,INT_REED_1);
    set_pullup(true,INT_REED_2);

    // ATA8520-E
    setup_spi(SPI_MASTER|SPI_SS_DISABLED|SPI_MODE_0|SPI_CLK_DIV_192);
```

```
ON(CS); // No seleccionado
ON(PC0); // No reset
OFF(PB4); // No despertar PB4=0 y PC1=1;
ON(PC1);

// Sensor
OFF(ON_SNS); //Apago la alimentacion al sensor
OFF(Ac_4_20); //Por defecto 0-10 V menos peligroso
bool_select_4_20=0;
output_float(GN); // High impedance pin GN

// ADCs
inicializacion_calibracion_ADC_SD();  //Utilizada para leer del sensor
inicializacion_adc_pipeline();        //Utilizada para nivel de bateria

// ***********************RESET TYPE*********************************

unsigned int16 ResetCause= RCON;
RCON &= 0b0011110100101100;


if((ResetCause & (1<<RESTART_MCLR))||(ResetCause & (1<<RESTART_POWER_UP))){
    read_configuration_flash();
    int_day=1440;
    configuration_mode();
    //Escribir configuracione en la EEPROM
    }
else if(ResetCause & (1<<RESTART_SOFTWARE)){
    // Leer configuraci□e la EEPROM
    read_configuration_flash();
    int_day=1440;
    int16 bat_level;
    OFF(GN);
    delay_ms(1);
    bat_level=read_battery_level_ADC;
    output_float(GN);

    wake_up_and_reset_ATA8520E();
    while(input(PB6)!=0);  //Espero a que se despierte
    get_status_ATA8520E();
    if(envio_y_recepcion_ATA8520E(bat_level)){
        if(rx_SigF_buffer[7]&0x01){   // = 1 -> actualizar la configuracion
            establish_configuration(rx_SigF_buffer);
            write_configuration_flash();
        }
    }
    sleep_ATA8520E();


    if(debug){
        ON(LD1);ON(LD2);delay_ms(200);
        OFF(LD1);delay_ms(200);
        ON(LD1);delay_ms(200);
        OFF(LD1);delay_ms(200);
        ON(LD1);ON(LD2);delay_ms(200);
        OFF(LD1);delay_ms(200);
        ON(LD1);delay_ms(200);
        OFF(LD1);delay_ms(200);
        ON(LD1);ON(LD2);delay_ms(200);
        OFF(LD1),OFF(LD2);
    }
}

if(debug){
    ON(LD2);ON(LD1);delay_ms(200);
    OFF(LD2);delay_ms(200);
    ON(LD2);delay_ms(200);
    OFF(LD2);delay_ms(200);
```

```
        ON(LD2);delay_ms(200);
        OFF(LD1);OFF(LD2);
    }

    // RTC
    dtm.tm_year=19;
    dtm.tm_mon=9;
    dtm.tm_mday=7;
    dtm.tm_wday=2;
    dtm.tm_hour=9;
    dtm.tm_min=0;
    dtm.tm_sec=0;

    rtc_write(&dtm);
    setup_rtc(RTC_ENABLE, 0);
    memcpy(&alm,&dtm,sizeof(dtm));
    alm.tm_min=alm.tm_min+10;
    rtc_alarm_write(&alm);
    setup_rtc_alarm(RTC_ALARM_ENABLE | RTC_CHIME_ENABLE, RTC_ALARM_MINUTE, 2);
    enable_interrupts(int_rtc);

    set_pullup(false,INT_REED_1);
    set_pullup(false,INT_REED_2);



while(1){
    delay_ms(1);
    //************** ACTIVAR EL MODO SLEEP***********
    sleep_adc_pipeline();
    sleep_ADC_SD();
    sleep(DEEP_SLEEP|WAKE_FROM_RTCC);
}

}
```

5. CÓDIGO APLICACIÓN

5.1. Android manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.minisigfoxapp">

    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.BLUETOOTH" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Configuration"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

5.2. Programación de las distintas interfaces

5.2.1. Actividad inicial

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="173dp"
        android:layout_height="158dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginStart="118dp"
        android:layout_marginTop="24dp"
        app:srcCompat="@mipmap/ic_launcher_round" />

    <TextView
        android:id="@+id/txPdev"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Bienvenido!" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/txPdev"
        android:layout_marginTop="179dp"
        android:text="Seleccione un dispositivo" />
```

```
    <ListView
        android:id="@+id/ListPdev"
        android:layout_width="match_parent"
        android:layout_height="343dp"
        android:layout_below="@+id/txPdev"
        android:layout_marginTop="199dp" />

    <Button
        android:id="@+id/BttPdev"
        android:layout_width="match_parent"
        android:layout_height="96dp"
        android:layout_below="@id/ListPdev"
        android:layout_alignBottom="@id/ListPdev"
        android:layout_marginTop="11dp"
        android:layout_marginBottom="-106dp"
        android:text="Mostrar dispositivos" />
</RelativeLayout>
```

5.2.2. Actividad configuración

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Configuration">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_marginTop="2dp"
        android:orientation="vertical">

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="360dp">

            <LinearLayout
                android:id="@+id/scroll"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <LinearLayout
                    android:id="@+id/config_layout"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:orientation="vertical">

                    <TextView
                        android:id="@+id/txt1"
                        android:layout_width="match_parent"
                        android:layout_height="20dp"
                        android:text="Configuración" />
                    <LinearLayout
                        android:layout_width="match_parent"
                        android:layout_height="40dp"
                        android:orientation="horizontal"
                        android:visibility="visible">

                        <TextView
                            android:id="@+id/txt2"
                            android:layout_width="50dp"
                            android:layout_height="match_parent"
                            android:layout_weight="1"
                            android:gravity="center_vertical"
```

```
            android:text="Frecuencia muestreo:"
            tools:text="Frecuencia muestreo:" />

        <EditText
            android:id="@+id/id_Freq"
            android:layout_width="30dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="number" />

        <TextView
            android:id="@+id/txt3"
            android:layout_width="20dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="(minutos)"
            tools:text="(minutos)" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/txt4"
            android:layout_width="50dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Tamaño paquetes:"
            tools:text="Tamaño paquetes:" />

        <EditText
            android:id="@+id/id_TamP"
            android:layout_width="30dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="number" />

        <TextView
            android:id="@+id/txt5"
            android:layout_width="20dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="(medidas/envio)"
            tools:text="(medidas/envio)" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/txt12"
            android:layout_width="40dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
```

```
            android:text="Hora sincronización:"
            tools:text="Hora sincronización:" />

        <EditText
            android:id="@+id/etChooseTime"
            android:layout_width="140dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:ems="10"
            android:inputType="time" />


    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView4"
            android:layout_width="309dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Modo Sensor:"
            tools:text="Modo Sensor:" />

        <ToggleButton
            android:id="@+id/id_mode_sensor"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="ToggleButton"
            android:textOff="TENSION (0-10V)"
            android:textOn="CORRIENTE (4-20mA)" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btSend"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="ENVIAR CONFIGURACION" />

        <Button
            android:id="@+id/id_AutoConf"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Config. Automatica" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView0"
            android:layout_width="150dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Warm time(ms)"
```

```
            tools:text="Warm time(ms)" />

        <EditText
            android:id="@+id/id_warmtime"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="numberDecimal" />

        <Button
            android:id="@+id/id_setwarmtime"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ajustar" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView"
            android:layout_width="150dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Ganancia"
            tools:text="Ganancia" />

        <EditText
            android:id="@+id/id_gain"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="numberDecimal" />

        <Button
            android:id="@+id/id_setgain"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ajustar" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView5"
            android:layout_width="150dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center_vertical"
            android:text="Offset"
            tools:text="Offset" />

        <EditText
            android:id="@+id/id_offset"
```

```
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="numberDecimal" />

        <Button
            android:id="@+id/id_setoffset"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ajustar" />

    </LinearLayout>

    <TextView
        android:id="@+id/txt11"
        android:layout_width="match_parent"
        android:layout_height="20dp"
        android:text="Alarmas (min y máx)" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <EditText
            android:id="@+id/min_alarm"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Min alarma"
            android:inputType="numberDecimal" />

        <EditText
            android:id="@+id/max_alarm"
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Max alarma"
            android:inputType="numberDecimal" />

        <Button
            android:id="@+id/id_setalarm"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ajustar" />

    </LinearLayout>

    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="20dp"
        android:text="Comprobar Configuración"
        tools:text="Comprobar Configuración" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/id_print_conf"
```

```
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Mostrar" />

            <Button
                android:id="@+id/id_read_conf"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Cargar" />

            <Button
                android:id="@+id/id_save_conf"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="guardar" />
        </LinearLayout>

    </LinearLayout>

    <LinearLayout
        android:id="@+id/SigFox_Layout"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView6"
            android:layout_width="wrap_content"
            android:layout_height="20dp"
            android:gravity="center_vertical"
            android:text="Comprobar SigFox"
            tools:text="Comprobar SigFox" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:orientation="horizontal">

            <Button
                android:id="@+id/id_prop_SigFox"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Propiedades" />

            <Button
                android:id="@+id/id_send_SigFox"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Enviar MSG" />

        </LinearLayout>

    </LinearLayout>

    <LinearLayout
        android:id="@+id/Bat_Sens_Layout"
        android:layout_width="match_parent"
        android:layout_height="80dp"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView7"
            android:layout_width="wrap_content"
```

```
            android:layout_height="20dp"
            android:gravity="center_vertical"
            android:text="Comprobar Bateria y Sensor"
            tools:text="Comprobar Bateria y Sensor" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:orientation="horizontal">

            <TextView
                android:id="@+id/id_signals_ouput"
                android:layout_width="250dp"
                android:layout_height="match_parent" />

            <ToggleButton
                android:id="@+id/id_measure"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="ToggleButton"
                android:textOff="START"
                android:textOn="STOP" />

        </LinearLayout>

    </LinearLayout>

    <LinearLayout
        android:id="@+id/Pant_Layout"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView8"
            android:layout_width="wrap_content"
            android:layout_height="20dp"
            android:gravity="center_vertical"
            android:text="Comprobar Pantalla"
            tools:text="Comprobar Pantalla" />

        <ToggleButton
            android:id="@+id/id_OLED"
            android:layout_width="match_parent"
            android:layout_height="40dp"
            android:layout_weight="1"
            android:text="ToggleButton"
            android:textOff="ENCENDER OLED"
            android:textOn="APAGAR OLED" />


    </LinearLayout>

    <LinearLayout
        android:id="@+id/Calib_Layout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/textView9"
            android:layout_width="wrap_content"
            android:layout_height="20dp"
            android:gravity="center_vertical"
            android:text="Calibración"
            tools:text="Calibración" />
        <LinearLayout
```

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="horizontal">

<TextView
    android:id="@+id/id_Pcalib_list"
    android:layout_width="150dp"
    android:layout_height="wrap_content" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <TextView
        android:id="@+id/textView10"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="Presión"
        tools:text="Presión" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <EditText
            android:id="@+id/id_pressureCalib"
            android:layout_width="133dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:ems="10"
            android:hint="Insert"
            android:inputType="numberDecimal" />

        <Button
            android:id="@+id/id_addPoint"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="+" />
        <Button
            android:id="@+id/id_deletePoint"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="-" />


    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/id_ListCalib"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:text="List" />

        <Button
            android:id="@+id/id_CalculCalib"
            android:layout_width="wrap_content"
```

```
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Calibrar" />

            <Button
                android:id="@+id/id_sendCalib"
                android:layout_width="wrap_content"
                android:layout_height="match_parent"
                android:layout_weight="0"
                android:text="Enviar" />


        </LinearLayout>
    </LinearLayout>
</LinearLayout>
</LinearLayout>
</ScrollView>

<LinearLayout
    android:id="@+id/Exit_Layout"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btDisconect"
        android:layout_width="wrap_content"
        android:layout_height="50dp"
        android:layout_weight="1"
        android:text="Disconect" />

    <Button
        android:id="@+id/id_close"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Finalizar" />

    <ToggleButton
        android:id="@+id/id_debug"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="ToggleButton"
        android:textOff="NO DEBUG"
        android:textOn="DEBUG" />

    <Button
        android:id="@+id/id_clear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Vaciar" />
</LinearLayout>

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="150dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">

        <TextView
            android:id="@+id/myLabel"
            android:layout_width="match_parent"
```

```
                    android:layout_height="wrap_content" />
            </LinearLayout>
        </ScrollView>
    </LinearLayout>
</RelativeLayout>
```

5.3. Código de las distintas actividades

Para la simplificación del código, y que sea más fácilmente interpretable por el lector, se han reducido las estructuras repetidas, las estructuras de creación de *Listerners*:

```
        Botón.setOnClickListener(newView.OnClickListener() {
            @Override
            public void onClick(View v)
            {
                funcion();
            }
        });
    }
```

Se han sustituido por:

```
Botón.setOnClickListener( → funcion());
```

La estructura de función, para la utilización de la conexión Bluetooth:

```
    private void funcion()
    {
        if (btSocket!=null){
            try{
                Conjunto de acciones ...
            }catch (IOException e){
                msg("Error");
            }
        }
    }
```

Ha sido simplificada por:

```
private void funcion(→){
       Conjunto de acciones ...
}
```

5.3.1. Actividad inicial

```
package com.example.minisigfoxapp;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.widget.TextView;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.Set;
import android.os.Bundle;
```

```
public class MainActivity extends AppCompatActivity {

    //widgets
    Button btnPaired;
    ListView devicelist;
    //Bluetooth
    private BluetoothAdapter myBluetooth = null;
    private Set<BluetoothDevice> pairedDevices;
    public static String EXTRA_ADDRESS = "device_address";


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Calling widgets
        btnPaired = (Button)findViewById(R.id.BttPdev);
        devicelist = (ListView)findViewById(R.id.ListPdev);

        //if the device has bluetooth
        myBluetooth = BluetoothAdapter.getDefaultAdapter();

        if(myBluetooth == null)
        {
            //Show a mensag. that the device has no bluetooth adapter
            Toast.makeText(getApplicationContext(), "Bluetooth Device Not
Available", Toast.LENGTH_LONG).show();

            //finish apk
            finish();
        }
        else if(!myBluetooth.isEnabled())
        {
            //Ask to the user turn the bluetooth on
            Intent turnBTon = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(turnBTon,1);
        }

        btnPaired.setOnClickListener(→ pairedDevicesList());

    private void pairedDevicesList()
    {
        pairedDevices = myBluetooth.getBondedDevices();
        ArrayList list = new ArrayList();

        if (pairedDevices.size()>0)
        {
            for(BluetoothDevice bt : pairedDevices)
            {
                list.add(bt.getName() + "\n" + bt.getAddress()); //Get the
device's name and the address
            }
        }
        else
        {
            Toast.makeText(getApplicationContext(), "No Paired Bluetooth Devices
Found.", Toast.LENGTH_LONG).show();
        }

        final ArrayAdapter adapter = new
ArrayAdapter(this,android.R.layout.simple_list_item_1, list);
        devicelist.setAdapter(adapter);
        devicelist.setOnItemClickListener(myListClickListener); //Method called
when the device from the list is clicked

    }
```

```
    private AdapterView.OnItemClickListener myListClickListener = new
AdapterView.OnItemClickListener()
    {
        public void onItemClick (AdapterView<?> av, View v, int arg2, long arg3)
        {
            // Get the device MAC address, the last 17 chars in the View
            String info = ((TextView) v).getText().toString();
            String address = info.substring(info.length() - 17);

            // Make an intent to start next activity.
            Intent i = new Intent(MainActivity.this, Configuration.class);

            //Change the activity.
            i.putExtra(EXTRA_ADDRESS, address); //this will be received at
Configuration Activity
            startActivity(i);
        }
    };
}
```

5.3.2. Actividad configuración

```
package com.example.minisigfoxapp;

import androidx.appcompat.app.AppCompatActivity;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.util.TypedValue;
import android.view.Menu;
import android.view.MenuItem;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.TimePicker;
import android.widget.Toast;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.os.AsyncTask;
import android.os.Handler;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.IOException;
import java.util.Calendar;
import java.util.UUID;
import android.os.Bundle;
import android.widget.ToggleButton;

public class Configuration extends AppCompatActivity {

    Button btnDis, btSend,btAutoConf,btSetWarmTime,btSetGain;
    Button btSetOffset,btPrintConf,btReadConf,btSaveConf,btclose,btSetAlarm;
    Button btPropSigFox, btEnviarSigFox;
    Button bt_menu_config,bt_clear,bt_debug;
    Button bt_addPCalib,bt_deletePCalib,bt_ListCalib,bt_calCalib,bt_SendCalib;
    EditText PresCalib;
    EditText Freq,TamP;
    EditText warmtime,gain,offset,minAlarm,maxAlarm;
    ToggleButton ModeSensor,btOLED,btmeasure;
    EditText chooseTime;
```

```
    LinearLayout config_layout;
    TextView signals_ouput,Pcalib_list;

    String address = null;
    private ProgressDialog progress;
    BluetoothAdapter myBluetooth = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;

    TextView IdBufferIn;
    Handler bluetoothIn;
    private StringBuilder DataStringIN = new StringBuilder();
    private ConnectedThread MyConexionBT;
    final int handlerState = 0;
    String bufferOUT="";

    //SPP UUID. Look for it
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent newint = getIntent();
        address = newint.getStringExtra(MainActivity.EXTRA_ADDRESS); //receive
                                            the address of the bluetooth device
        setContentView(R.layout.activity_configuration);

        //call the widgtes
        chooseTime=(EditText)findViewById(R.id.etChooseTime);
        config_layout = (LinearLayout)findViewById(R.id.config_layout);
        btnDis = (Button)findViewById(R.id.btDisconect);
        IdBufferIn = (TextView) findViewById(R.id.myLabel);
        signals_ouput=(TextView)findViewById(R.id.id_signals_ouput);
        btSend= (Button)findViewById(R.id.btSend) ;
        Freq= (EditText) findViewById(R.id.id_Freq);
        TamP= (EditText) findViewById(R.id.id_TamP);
        ModeSensor=(ToggleButton) findViewById(R.id.id_mode_sensor);
        btSetAlarm=(Button)findViewById(R.id.id_setalarm);
        minAlarm=(EditText)findViewById(R.id.min_alarm);
        maxAlarm=(EditText)findViewById(R.id.max_alarm);
        btAutoConf=(Button)findViewById(R.id.id_AutoConf);
        warmtime=(EditText)findViewById(R.id.id_warmtime);
        gain=(EditText) findViewById(R.id.id_gain);
        offset=(EditText)findViewById(R.id.id_offset);
        btSetWarmTime=(Button)findViewById(R.id.id_setwarmtime);
        btSetGain= (Button) findViewById(R.id.id_setgain);
        btSetOffset=(Button)findViewById(R.id.id_setoffset);
        btPrintConf=(Button) findViewById(R.id.id_print_conf);
        btReadConf=(Button)findViewById(R.id.id_read_conf);
        btSaveConf=(Button)findViewById(R.id.id_save_conf);
        btOLED=(ToggleButton)findViewById(R.id.id_OLED);
        btmeasure=(ToggleButton)findViewById(R.id.id_measure);
        btclose=(Button) findViewById(R.id.id_close);
        btPropSigFox=(Button)findViewById(R.id.id_prop_SigFox);
        btEnviarSigFox= (Button)findViewById(R.id.id_send_SigFox);
        bt_clear=(Button)findViewById(R.id.id_clear);
        bt_debug=(ToggleButton)findViewById(R.id.id_debug);
        bt_addPCalib=(Button)findViewById(R.id.id_addPoint);
        bt_deletePCalib=(Button)findViewById(R.id.id_deletePoint);
        bt_ListCalib=(Button)findViewById(R.id.id_ListCalib);
        bt_calCalib=(Button)findViewById(R.id.id_CalculCalib);
        bt_SendCalib=(Button)findViewById(R.id.id_sendCalib);
        PresCalib=(EditText)findViewById(R.id.id_pressureCalib);
        Pcalib_list=(TextView)findViewById(R.id.id_Pcalib_list);
```

```
bluetoothIn = new Handler() {
    public void handleMessage(android.os.Message msg) {
        if (msg.what == handlerState) {

            String readMessage = (String) msg.obj;
            DataStringIN.append(readMessage);
            //IdBufferIn.setText(readMessage);
            int endOfLineIndex=1;
            while(endOfLineIndex >0) {
                endOfLineIndex = DataStringIN.indexOf("\r");
                if (endOfLineIndex > 0) {
                    String dataInPrint = DataStringIN.substring(0,
                                                endOfLineIndex);
                    if(dataInPrint.startsWith("*")){
                        String buffer2 =
                        dataInPrint.substring(1,dataInPrint.length());
                        String[] buffer3=buffer2.split(",");
                        signals_ouput.setText("  Presión:  " +
                         buffer3[0] + "\n  Bateria:  " + buffer3[1] );
                    }
                    else if(dataInPrint.startsWith("+")){
                        String buffer4=Pcalib_list.getText().toString();
                        Pcalib_list.setText((buffer4+">
                      "+dataInPrint.substring(1,dataInPrint.length())));
                    }
                    else if(dataInPrint.startsWith("-")){
                        String buffer4=Pcalib_list.getText().toString();
                        buffer4 = buffer4.substring(0,
                                                buffer4.lastIndexOf(">"));
                        Pcalib_list.setText(buffer4);
                    }
                    else if(dataInPrint.startsWith("$")){

                        String[] buffer3=dataInPrint.split(",");
                        Freq.setText(buffer3[1]);
                        TamP.setText(buffer3[2]);
                        ModeSensor.setChecked(buffer3[3].contains("1"));
                        Calendar calendar= Calendar.getInstance();
                        chooseTime.setText(String.format("%02d:%02d",
                                  calendar.get(Calendar.HOUR_OF_DAY),
                                  calendar.get(Calendar.MINUTE)));
                        gain.setText(buffer3[4]);;
                        offset.setText(buffer3[5]);;
                        minAlarm.setText(buffer3[6]);;
                        maxAlarm.setText(buffer3[7]);;
                        warmtime.setText(buffer3[8]);
                    }
                    else {
                        String buffer = IdBufferIn.getText().toString();
                        IdBufferIn.setText(buffer + "> " + dataInPrint);
                    }
                    DataStringIN.delete(0, endOfLineIndex + 1);
                }
            }
        }
    }
};

new ConnectBT().execute(); //Call the class to connect



chooseTime.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){
        TimePickerDialog timePickerDialog=new
           TimePickerDialog(Configuration.this,new
```

```
                                                TimePickerDialog.OnTimeSetListener(){
                        @Override
                        public void onTimeSet(TimePicker timePicker,int hourDay, int
                                                                                minutes){

    chooseTime.setText(String.format("%02d:%02d",hourDay,minutes));

                    }
                },0,0,false);

                   timePickerDialog.show();
                }
            });

            bt_clear.setOnClickListener(→ IdBufferIn.setText(""));
            bt_addPCalib.setOnClickListener(AddPCalib());
            bt_deletePCalib.setOnClickListener(DeletePCalib());
            bt_ListCalib.setOnClickListener(→ ListCalib());
            bt_calCalib.setOnClickListener(→ CalcCalib());
            bt_SendCalib.setOnClickListener(→ SendCalib());
            btPropSigFox.setOnClickListener(→ PropSigFox());
            btEnviarSigFox.setOnClickListener(→ EnviarSigFox());
            btAutoConf.setOnClickListener(→ AutoConfiguration());
            btSetWarmTime.setOnClickListener(→ SendWarmTime());
            btSetGain.setOnClickListener(→ SendGain());
            btSetAlarm.setOnClickListener(→ SendAlarms());
            btSetOffset.setOnClickListener(→ SendOffset());
            btPrintConf.setOnClickListener(→ PrintConfiguration());
            btReadConf.setOnClickListener(→ ReadConfiguration());
            btSaveConf.setOnClickListener(→ SaveConfiguration());
            btOLED.setOnClickListener(→ OLED_change());
            btmeasure.setOnClickListener(→ measure_change());
            btSend.setOnClickListener(→ SendConfiguration();
            btclose.setOnClickListener(→ CloseConfiguration());
            bt_debug.setOnClickListener(→ ChangeDebug());
            btnDis.setOnClickListener(→ Disconnect());

    //----------------------------------------------------------------------
    //                FUNCIONES
    //----------------------------------------------------------------------
        private void Disconnect(→){
            btSocket.close(); //close connection
            finish(); //return to the first layout
        }

        private void measure_change(→){
            if(btmeasure.isChecked())
                  btSocket.getOutputStream().write("M".getBytes());
             else btSocket.getOutputStream().write("F".getBytes());
        }

        private void OLED_change(→){
            if(btOLED.isChecked())
                  btSocket.getOutputStream().write("O".getBytes());
            else btSocket.getOutputStream().write("X".getBytes());
        }

        private void CloseConfiguration(→){
                    btSocket.getOutputStream().write("S".getBytes());   //
        }

        private void AutoConfiguration(→){
                    btSocket.getOutputStream().write("A".getBytes());   //
        }
        private void DeletePCalib(→) {
                    btSocket.getOutputStream().write("-".getBytes());   //
        }
```

```
private void ListCalib(→){
            btSocket.getOutputStream().write("l".getBytes());  //
}
private void CalcCalib(→){
            btSocket.getOutputStream().write("f".getBytes());  //
}
private void SendCalib(→){
            btSocket.getOutputStream().write("s".getBytes());  //
}
private void AddPCalib(→){
            btSocket.getOutputStream().write("+".getBytes());
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }

            float f=Float.parseFloat(PresCalib.getText().toString());
            int bits=Float.floatToIntBits(f);
            byte[] bytes =new byte[4];
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);
}
private void EnviarSigFox(→){
            btSocket.getOutputStream().write("2".getBytes());  //
}
private void PropSigFox(→){
            btSocket.getOutputStream().write("1".getBytes());  //
}
private void PrintConfiguration(→){
            btSocket.getOutputStream().write("P".getBytes());  //
}
private void SaveConfiguration(→){
            btSocket.getOutputStream().write("W".getBytes());  //
}
private void ReadConfiguration(→){
            btSocket.getOutputStream().write("R".getBytes());  //
}
private void ChangeDebug(→){
            btSocket.getOutputStream().write("D".getBytes());  //
}
private void SendWarmTime(→){
            btSocket.getOutputStream().write("w".getBytes());
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }

            float f=Float.parseFloat(warmtime.getText().toString());
            int bits=(int) (f/10);
            byte[] bytes =new byte[1];
            bytes[0]= (byte) (bits & 0xff);
            btSocket.getOutputStream().write(bytes);
}
private void SendGain(→){
            btSocket.getOutputStream().write("G".getBytes());
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }
            float f=Float.parseFloat(gain.getText().toString());
            int bits=Float.floatToIntBits(f);
```

```
            byte[] bytes =new byte[4];
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);
    }
    private void SendAlarms(→)
    {
            btSocket.getOutputStream().write("L".getBytes());
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }
            float f=Float.parseFloat(minAlarm.getText().toString());
            int bits=Float.floatToIntBits(f);
            byte[] bytes =new byte[4];
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);

            f=Float.parseFloat(maxAlarm.getText().toString());
            bits=Float.floatToIntBits(f);
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);
    }
    private void SendOffset(→){
            btSocket.getOutputStream().write("N".getBytes());
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }

            float f=Float.parseFloat(offset.getText().toString());
            int bits=Float.floatToIntBits(f);
            byte[] bytes =new byte[4];
            bytes[0]= (byte) (bits & 0xff);
            bytes[1]= (byte) ((bits >>8) & 0xff);
            bytes[2]= (byte) ((bits >>16) & 0xff);
            bytes[3]= (byte) ((bits >>24) & 0xff);
            btSocket.getOutputStream().write(bytes);
    }
    private void SendConfiguration(→)
    {

            btSocket.getOutputStream().write("C".getBytes());  //Conf
            try {
                Thread.sleep(400);
            }catch(InterruptedException e){
                msg("Error");
            }
            int Frequency= Integer.parseInt(Freq.getText().toString());
            int Tam_packet=Integer.parseInt(TamP.getText().toString());
            int MSensor = (ModeSensor.isChecked()) ? 1 : 0;

            if((Frequency > 1440) || (Frequency < 0)) Frequency = 1440;
            if((Tam_packet > 840) || (Tam_packet < 0)) Tam_packet=840;
            if((TimeR > 1440) || (TimeR < 0)) TimeR=1440;

            // Calculo de los minutos que quedan para el RESET
```

```
                String s_time=chooseTime.getText().toString();
                String[] s_time2=s_time.split(":");

                int hora=Integer.parseInt(s_time2[0]);
                int minutos=Integer.parseInt((s_time2[1]));

                Calendar calendar= Calendar.getInstance();
                int currentHour=calendar.get(Calendar.HOUR_OF_DAY);
                int currentMinute= calendar.get(Calendar.MINUTE);

                int minutes_to_reset=(hora*60+minutos)-
                                        (currentHour*60+currentMinute);
                if(minutes_to_reset<0) minutes_to_reset=minutes_to_reset+1440;

                msg("Quedan :"+ String.valueOf(minutes_to_reset)+ " minutos para
                        el reset");

                byte[] config=new byte[8];
                int[] aux=new int[2];

                aux[0]=  (((Frequency&0x0000FFFF)<<21)+
                   ((Tam_packet&0x0000FFFF)<<11)+(minutes_to_reset&0x0000FFFF));

                aux[1]=  (MSensor&0x00000001)<<31;

                config[0]= (byte) ((aux[0]&0xFF000000)>>24);
                config[1]= (byte) ((aux[0]&0x00FF0000)>>16);
                config[2]= (byte) ((aux[0]&0x0000FF00)>>8);
                config[3]= (byte) ((aux[0]&0x000000FF));

                config[4]= (byte) ((aux[1]&0xFF000000)>>24);
                config[5]= (byte) ((aux[1]&0x00FF0000)>>16);
                config[6]= (byte) ((aux[1]&0x0000FF00)>>8);
                config[7]= (byte) ((aux[1]&0x000000FF));

                for(int i=0;i<8;i++){
                    btSocket.getOutputStream().write(config[i]);
                }
    }
    // fast way to call Toast
    private void msg(String s)
    {
        Toast.makeText(getApplicationContext(),s,Toast.LENGTH_LONG).show();
    }

    private class ConnectBT extends AsyncTask<Void, Void, Void>  // UI thread
    {
        private boolean ConnectSuccess = true; //if it's here, it's almost
                                               connected

        @Override
        protected void onPreExecute()
        {
            progress = ProgressDialog.show(Configuration.this, "Connecting...",
               "Please wait!!!");  //show a progress dialog
        }

        @Override
        protected Void doInBackground(Void... devices) //while the progress
                    dialog is shown, the connection is done in background
        {
            try
            {
                if (btSocket == null || !isBtConnected)
                {
                    myBluetooth = BluetoothAdapter.getDefaultAdapter();//get the
                                                          mobile bluetooth device
                    BluetoothDevice dispositivo =
```

```
                    myBluetooth.getRemoteDevice(address);//connects to the
                                device's address and checks if it's available
                    btSocket =
                  dispositivo.createInsecureRfcommSocketToServiceRecord(myUUID);
                   //create a RFCOMM (SPP) connection
                    BluetoothAdapter.getDefaultAdapter().cancelDiscovery();
                    btSocket.connect();//start connection
                    MyConexionBT = new ConnectedThread(btSocket);
                    MyConexionBT.start();

                }
            }
            catch (IOException e)
            {
                ConnectSuccess = false;//if the try failed, you can check the
                                       exception here
            }
            return null;
        }
        @Override
        protected void onPostExecute(Void result) //after the doInBackground, it
                                                  checks if everything went fine
        {
            super.onPostExecute(result);

            if (!ConnectSuccess)
            {
                msg("Connection Failed. Is it a SPP Bluetooth? Try again.");
                finish();
            }
            else
            {
                msg("Connected.");
                isBtConnected = true;
            }
            progress.dismiss();
        }
    }


    //Crea la clase que permite crear el evento de conexion
    private class ConnectedThread extends Thread
    {
        private final InputStream mmInStream;
        private final OutputStream mmOutStream;

        public ConnectedThread(BluetoothSocket socket)
        {
            InputStream tmpIn = null;
            OutputStream tmpOut = null;
            try
            {
                tmpIn = socket.getInputStream();
                tmpOut = socket.getOutputStream();
            } catch (IOException e) { }
            mmInStream = tmpIn;
            mmOutStream = tmpOut;
        }

        public void run()
        {
            byte[] buffer = new byte[256];
            int bytes;

            // Se mantiene en modo escucha para determinar el ingreso de datos
            while (true) {
                try {
                    bytes = mmInStream.read(buffer);
```

```
                    String readMessage = new String(buffer, 0, bytes);
                    // Envia los datos obtenidos hacia el evento via handler
                    bluetoothIn.obtainMessage(handlerState, bytes, -1,
                      readMessage).sendToTarget();
                } catch (IOException e) {
                    break;
                }
            }
        }
        //Envio de trama
        public void write(String input)
        {
            try {
                mmOutStream.write(input.getBytes());
            }
            catch (IOException e)
            {
                //si no es posible enviar datos se cierra la conexión
                Toast.makeText(getBaseContext(), "La Conexión fallo",
                     Toast.LENGTH_LONG).show();
                finish();
            }
        }
    }


}
```