Universidad Carlos III de Madrid

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

TRABAJO DE FIN DE GRADO

APLICACIÓN SOFTWARE EMBEBIDA PARA LA MEDIDA DEL CONSUMO DE ENERGÍA EN NODOS INALÁMBRICOS DE REDES DE SENSORES

AUTOR: IVÁN GARRIDO VEGA

TUTOR: EMILIO OLÍAS RUIZ

LEGANÉS, 26 DE SEPTIEMBRE DE 2017

AGRADECIMIENTOS

En primer lugar quería agradecer a mi tutor Emilio Olías Ruiz, a Celia López Ongil y a José Ángel Miranda Calero, por la ayuda prestada durante el proyecto. También dar las gracias al resto de compañeros de IoT, en especial a Óscar Escobar Muñoz, pues nos hemos ayudado en distintas apartados de nuestros respectivos proyectos, y sí, también hemos sufrido juntos. Y no me olvido ni mucho menos de los técnicos de laboratorio de la universidad, quienes nos han sacado de más de un apuro.

Por supuesto agradecer a mis padres y a mi familia, por el apoyo y el cariño que siempre me han dado. Siempre han apostado y lo han dado todo por mi futuro. El primer sustento emocional y económico que me ha permitido andar por este camino.

También a aquellos que ya no están aquí, que nunca olvidaré, un motivo más para seguir esforzándome cada día. Ya que el esfuerzo es el punto común de todos aquellos que han mejorado la tecnología para salvar vidas.

Especial mención también a mis compañeros y amigos de la universidad. Hemos sufrido, reído y aprendido juntos. Sin vosotros, habría sido un camino mucho más arduo. Con vosotros, he descubierto ese apoyo mutuo, compañerismo y trabajo en equipo con el que se alcanzan las mayores metas.

A mi novia Marta, que también ha soportado tanto mi presencia como mi ausencia. Por esas largas tardes y noches de estudio y trabajo codo con codo. No sé lo que me deparará el futuro, tan sólo sé que quiero que estés en él, y yo en el tuyo.

A los que cada semana que pasaba he visto menos. Los mismos con los que he crecido y he pasado muchos buenos y malos momentos, por supuesto, más de los primeros. Hablo, claro está, de mis amigos. Haré que merezca la pena cada minuto que no he podido estar con vosotros.

Por último, a todos aquellos que han estado presentes de alguna manera u otra en vida y hayan aportado algo positivo imperceptible en mí, pero cuya suma ha hecho más fácil que llegara hasta aquí.

RESUMEN

Hoy en día cada vez se ven más aplicaciones del "Internet de las Cosas" (IoT). Una de sus múltiples facetas son las redes de sensores inteligentes en infraestructuras, que recogen información del entorno, la transmiten y procesan. En el caso de que sean inalámbricos y se alimenten por medio de una batería, el consumo es algo crítico. Un buen análisis del consumo de la red permitirá optimizar las rutinas de comunicación y elegir la batería adecuada.

En este proyecto se ha dedicado el esfuerzo en elaborar el software necesario para la correcta medida de consumo de una red de sensores inalámbricos dedicada a la seguridad.

Dando unos primeros pasos en este proyecto encontrará el estudio general realizado sobre el análisis del consumo de este tipo de aplicaciones. A continuación, se hablará de todo el proceso de diseño seguido desde los elementos más simples hasta la estructura de medida final completa, junto con el software asociado. En los últimos capítulos, se detallan los resultados obtenidos para el caso de estudio en concreto y las conclusiones derivadas de los mismos.

Con todo ello se tratará de mostrar la experiencia adquirida en este campo, además de la importancia de conocer el consumo del sistema implementado. Ya que, en definitiva, este análisis es el paso previo a la necesaria optimización de toda aplicación en cuanto a ahorro de energía, algo que siempre debe tenerse en cuenta y más aún en el futuro.

ABSTRACT

Nowadays, there are more a more “Internet of Things” (IoT), applications. One of its many facets are intelligent sensor networks in infrastructures, which collect information from the environment, and then transmit and process it. In the case that they are wireless and they need to be powered by a battery, consumption is a critical aspect. A good analysis of consumption will allow to optimize the communication routines and to choose the right battery.

In this project I have put a lot of effort in design a software for the correct measurement of consumption of a network of wireless sensors dedicated to security.

Taking a first steps in this project, we could see the general study on the analysis of the consumption of this type of applications. Next, we will discuss the whole design process, from the simplest elements to the complete final measurement structure, along with the associated software. Finally, we discuss the results obtained for the specific case of study and the conclusions derived from them.

In this project I tried to show the experience gained in this field, besides the importance of knowing the consumption of the implemented system. Since, in short, this analysis is the step prior to the necessary optimization of every application related to energy saving. That is something that should always be taken into account, and more so in the future.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

LISTADO DE ACRÓNIMOS

ACK	Acknowledge
ADC	Analogic Digital converter
AES	Advanced Encryption Standard
API	Application Programming Interface
BLE	Bluetooth Low Energy
CA2	Complemento a 2
CPU	Central Processing Unit
DEN	Denominator
E/S	Entrada/Salida
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEP	Functional End Point
FSC	Full Scale Current
FSV	Full Scale Voltage
GMR	Giant Magnetoresistance Effect
HAL	Hardware Abstraction Level
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IoT	Internet of Things
ISM	Industrial, Scientific and Medical
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
MIPS	Millones de Instrucciones Por Segundo
NWK	Network Layer
PA	Power Amplifier
PC	Personal Computer
PCB	Printed Circuit Board
PCB	Printed Circuit Board
PHR	PHY Header
PHY	Physical Layer
PLL	Phase Locked-Loop
PSDU	PHY Service Data Unit
PWM	Pulse-Width Modulation
PWR	Power

RC	Resistencia Capacitor
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RX	Receive/ Receiver/ Reception
SCL	Serial Clock
SDA	Serial Data
SHR	Synchronization Header
SMA	SubMiniature version A
SMBUS	System Management Bus
SoT	State of Charge
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TVS	Transient-Voltage-Suppression
TX	Transmission
UART	Universal Asynchronous Receiver-Transmitter
USART	Universal Synchronous and Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
WPAN	Wireless Personal Area Network

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

Aunque lleva ya algunos años, hoy el término de IoT [1](*Internet Of Things*) está dándose cada vez más a conocer. IoT comprende desde los *wearables*[2] *que controlan el pulso*, electrodomésticos como la nevera que hace la lista de la compra, los smartphones que nos acompañan en el día a día, hasta los vehículos capaces decirnos la mejor ruta según el estado de las carreteras, o dónde hay una plaza disponible para aparcar.

Los dispositivos inteligentes están aumentado en gran medida de año en año. En 2010 ya había una media de 1,84 dispositivos conectados por persona en el mundo. En 2015 ascendió a 3,57 (unos 30000 millones). Es más, aun siendo algo optimistas, algunos estudios hablan de 50.000 millones en 2020.

Una de las áreas de estas nuevas tecnologías son las redes de sensores inalámbricas. Tanto si es para uso doméstico como si se trata de usos industriales, estas redes de nodos [3] cada vez poseen más funcionalidades. Sin embargo, dado que su alimentación es limitada, ya que depende de una batería o conjunto de baterías al no disponer de cableado, es de vital importancia para un funcionamiento eficiente que tecnología y uso de la energía vayan evolucionando a la par.

Por tanto, se va a buscar siempre el mínimo consumo posible con la que se sea capaz de llevar a cabo de forma eficaz las funciones para las que se ha creado la red, así como una monitorización del consumo de dichas aplicaciones y del Estado de Carga (*State of Charge*) (**SoC**) de las baterías.

Es este último punto el eje principal de este proyecto. Un sistema implementado en una red de nodos dada que sea capaz de monitorizar el consumo de cada nodo para poder estimar el SoC de cada una de las baterías. Con ello, si bien se espera que con un uso eficiente y la elección de la tecnología adecuada las baterías puedan durar varios años, podremos anticiparnos al agotamiento de la batería para mantener en todo momento la red activa.

Por último, con un análisis del consumo para distintos modos de funcionamiento y estados de cada tipo de nodo, será posible optimizar las rutinas de los mismos para reducir aún más la energía utilizada y aumentar así la duración de las respectivas fuentes.

1.2. OBJETIVOS

Se pretende implementar un sistema de medida de consumo de corriente para una red de nodos ya definida, formado por placas Atmega256rfr2 Xplained Pro. Ello lleva consigo los siguientes objetivos parciales:

- Estudio de las distintas tecnologías de baterías y medida de consumo en redes inalámbricas.
- Conocimiento del estándar IEE.802.4 y de la red de nodos de Atmega.
- Conseguir un dispositivo capaz de medir consumo para distintos modos de funcionamiento, que sea compatible con la placa Xplained Pro.

[1] IoT fue un término acuñado por Kevin Ashton cuando trabajaba en el centro de Auto-ID del MIT para un sistema de RFID, haciendo referencia a un conjunto de sensores conectado a internet. [1]

[2] Los llamados *wearables* son todos aquellos dispositivos electrónicos que podemos llevar puestos e interactúan con nosotros y con el entorno. Gafas inteligentes de realidad aumentada, *smartwatchs* (relojes inteligentes) o ropa que mide las constantes vitales del cuerpo, son ejemplos de ello.

[3] El nodo es un punto terminal de una red, o cualquiera de sus intersecciones [2]

- Obtención de la información sobre el consumo para distintos modos de funcionamiento y representación gráfica de los mismos.
- Análisis de los resultados, proponiendo métodos para reducir el consumo, aplicados a la red del proyecto.
- Desarrollar una aplicación que permita al usuario monitorizar el consumo y configurar el sistema para hacer sus propias mediciones.

1.3. METODOLOGÍA DEL PROYECTO. TRABAJO EN EQUIPO

He considerado hacer mención en este apartado sobre las peculiaridades que ha tenido este proyecto respecto al desarrollo común de otros trabajos.

Este TFG forma parte de un conjunto de propuestas enfocadas al desarrollo de una red de seguridad con sensores inalámbricos. Se nos repartió a un grupo de alumnos los distintos aspectos que formarían la red. En principio estábamos divididos de la siguiente manera:

- George Sebastian Roma: RTOS
- Iker Barrondo Martin: detector de gases
- Adrián Neira Robledo: cámara infrarroja
- Óscar Escobar Muñoz: análisis del consumo
- Iván Garrido Vega: análisis del consumo

El objetivo era, entre otros, llevar a cabo nuestro proyecto siendo conscientes de desarrollar cada uno una parte de un todo, hacia un objetivo común. Así, se han hecho reuniones de seguimiento conjuntas en las que cada uno ha expuesto a los tutores y al resto de compañeros los avances conseguidos hasta el momento, además de poner en común en una carpeta compartida los archivos generados.

Por otro lado, Óscar y yo nos hemos apoyado el uno al otro en nuestros respectivos proyectos, al estar altamente relacionados entre sí. No han podido desarrollarse el uno sin el otro, pues la implementación del sensor, con la creación de los esquemáticos, PCB y diseño final del circuito de conexión son pasos esenciales para el funcionamiento de los drivers. Por otro lado, sin una aplicación adecuada no sería posible configurar el dispositivo de medida ni adquirir los datos.

En definitiva, este proyecto no sólo ha supuesto un reto individual sino también un paso más hacia la experiencia del trabajo en equipo, algo que se aproxima más a la realidad en las empresas actuales.

1.4. MEDIOS UTILIZADOS

A continuación se detallan los medios utilizados durante el transcurso del proyecto:

- **OrCAD Pspice y Layout (Versión 15.7):** para la elaboración de los esquemáticos y diseño del rutado y colocación de los componentes necesarios para la fabricación de la placa y montaje del circuito impreso **PCB** (*Printed Circuit Board*) del sensor de corriente PAC1710.
- **Chip Manager de Microchip (Versión 4.16.8.):** para obtener una referencia en cuanto al funcionamiento del sensor, así como una primera toma de medidas de consumo.
- **Atmel Studio (Versión 7.0) y Visual Studio (Version 7.0):** elaboración de los drivers del PAC1710 en el sistema embebido, programación de la placa que constituye el nodo.
- **Matlab (Versión R2010.a) y Excel 2013:** principalmente para un primer análisis de los datos obtenidos.
- **Smart RF Packet Sniffer, de Texas Instruments (Versión 2.18):** este software permite visualizar los paquetes de datos enviados por el nodo.

1.5. ESTADO DE LA TÉCNICA

En este apartado se explicarán las tecnologías utilizadas en la red de sensores inalámbricos, las placas utilizadas y los protocolos bajo los que se rige. Por otro lado se mencionarán algunas de las baterías que se emplean típicamente en estas aplicaciones, los principales métodos de reducción del consumo y algunos de los métodos de medida de consumo existentes.

1.5.1. REDES DE SENSORES INALÁMBRICOS [3] [4] [5] [6]

La red que se emplea en el proyecto consiste en un conjunto de nodos inalámbricos formado por placas **ATMEGA256rfr2 Xplained Pro**, que siguen el estándar de dispositivos de comunicación RF **IEEE 802.15.4** y el protocolo de red de baja potencia **Atmel lightweight mesh**.

Esta tecnología proporciona una red inalámbrica de baja potencia, simple y de bajo coste adaptada para aplicaciones de control remoto, seguridad, automatización de edificios y equipos didácticos.

Una descripción detallada del estándar IEEE y el protocolo de red se puede encontrar en el anexo, en el capítulo dedicado al marco regulador A.5. Por otro se hablará más delante de las características de la placa, en el apartado 2.2..

1.5.2. BATERÍAS UTILIZADAS EN REDES NODOS INALÁMBRICOS [7] [8] [9]

Las baterías que están actualmente en el mercado se pueden dividir en *baterías no recargables* o primarias y *baterías recargables* o secundarias, en las que las baterías de **Níquel-Cadmio (Ni-Cd)**, **Hidruro metálico (Ni-MH)** e **Ión de Litio (Li-on)** son las más utilizadas.

Otras tecnologías actualmente en estudio en este campo son las **baterías imprimibles**, las **baterías de estado** sólido y los **supercondesadores**, aún en desarrollo.

La *Tabla 1.1.* resume los pros y los contras de las tecnologías disponibles para la alimentación de los dispositivos IoT.

TECNOLOGÍA	PROS	CONTRAS
TECNOLOGÍAS BÁSICAS		
Baterías no recargables	conveniencia Costo	Contaminantes Corta duración (no recargables)
Baterías recargables (Ión-litio, Ni-Cd, Ni-MH)	Recargable	Ciclos de carga-descarga limitada factible con la recolección de energía
OTRAS TECNOLOGÍAS		
Baterías imprimibles	Fácil proceso de fabricación de células personalizable (tensión, capacidad, tamaño) delgada y flexible	Se pueden dañar a 40-50ºC No es una tecnología suficientemente madura
Baterías de estado sólido	Fácil integración (IC) fácil de miniaturizar delgada y flexible	Baja densidad de potencia No es una tecnología suficientemente madura
Supercondensadores	ciclos de carga-descarga "ilimitado"	Alta autodescarga (25%/día)

Tabla 1.1. Tecnologías para la alimentación de los dispositivos IoT: sus pros y contras.

Debido a sus características y a su disponibilidad en el mercado actual, se ven a continuación con más detalle las **baterías recargables**.

COMPARATIVA DE BATERÍAS RECARGABLES [10] [11] [12] [13] [14]

En primer lugar hay que definir algunas características de las baterías que pueden ser desconocidas:

- *Voltaje nominal por célula:* las baterías están formadas por celdas que proporcionan un nivel determinado de energía. Puede variar entre células, por lo que sueles ser necesario un circuito regulador adicional.
- *Autodescarga*: descarga de la batería en reposo, cuando no tiene ninguna carga conectada en bornes.
- *Capacidad*: es la cantidad de energía que puede almacenar, se mide en Amperios-Hora Ah. Si por ejemplo la capacidad es de 75 Ah, la betería podría aportar 7,5 A en 10 horas, 75 A en una hora, etc. **Se hablará más delante de esta característica, a la hora de analizar el consumo medido.**
- *Densidad de energía:* carga por unidad de peso y volumen.
- *Efecto memoria:* fenómeno por el cual se reduce la capacidad de la batería al cargarse la batería tras una descarga incompleta.

Ni-Cd: níquel cadmio

Las baterías de **níquel cadmio (Ni-Cd)** son baterías recargables de uso doméstico e industrial. Cada vez se utilizan menos debido a su efecto memoria y al cadmio, que es muy contaminante.

Ni-Mh: níquel e hidruro metálico

Las baterías de **níquel e hidruro metálico (Ni-MH)** son recargables y similares a las de níquel-cadmio (Ni-Cd), pero al no contener cadmio no son tan perjudiciales para el medio ambiente.

Li-ion: iones de litio

Las baterías de **ion-Litio**, pese a ser más caras y presentar un ciclo de vida algo menor, tienen mayor potencia nominal por celda, mayor capacidad y mayor densidad de energía, además de no presentar apenas autodescarga ni efecto memoria.

Son la tecnología de baterías de elección para la mayoría de estos dispositivos inalámbricos de ultra baja potencia.

En la *tabla 1.2.* se muestra un resumen comparativo de las baterías recargabes estudiadas:

CARACTERÍSTICAS	Ni-Cd	NiMH	LITIO / IÓN LITIO
VOLTAJE NOMINAL POR CÉLULA	1,2V	1,2V	3,7 V
PRECIO	MÁS BARATAS	MEDIO	MÁS CARAS (40%+)
CICLOS DE VIDA (DURACIÓN)	1000 CICLOS	400-600 CICLOS	500 (MENOR)
DESCARGA	ALTA DESCARGA	MENOR QUE NI-CD	(LINEAL)
AUTODESCARGA	1-3% DÍA	2-8% DÍA	APENAS AUTODESCARGA 20%+MES Y 6% EN LI-ON
RECARGA	VARIAS HORAS	+ QUE Ni-Cd	+ RÁPIDA
CAPACIDAD	BUENA	MÁS QUE NI-CD	MAYOR CAPACIDAD
DENSIDAD DE ENERGÍA	150 W-h/l	40% MÁS QUE NI-CD	620 W-h/L (La mayor)(4 veces más que Ni)
EFECTO MEMORIA	SI	NO	NO
RECICLAJE	RECICLABLE, CONTAMINA	DESECHABLE, NO CONTAMINA	DESECHABLE, NO CONTAMINA
TEMPERATURA DE OPEACIÓN	BUENA	ALGO MEJOR QUE Ni-Cd	PUEDEN EXPLOTAR SI SE SOBRECALIENTAN Y PEOR RENDIMIENTO EN FRÍO
USOS	Cada vez menor debido al efecto memoria y a la contaminación Uso doméstico e industrial (drones, aviones)	Sustitutas principales de las NiMh	Tecnología de baterías de elección para la mayoría de dispositivos inalámbricos de ultra baja potencia

Tabla 1.2. Comparativa de baterías recargables

1.5.3. ESTRATEGIAS PARA CONSEGUIR BAJO CONSUMO Y MÁXIMA DURACIÓN [15]

El funcionamiento de un sensor inalámbrico bidireccional (capaz de transmitir y recibir datos) puede segmentarse en una serie de actividades, cada una de las cuales requiere un determinado nivel de potencia durante un periodo de tiempo específico. Las actividades más comunes son:

- **Modos activos alta energía**
  - Activación→toma de una medida→ procesamiento de datos.

  - Encendido del amplificador de potencia de RF→transmisión del mensaje→apagado del amplificador de potencia de RF.
  - Encendido del receptor→recepción→procesamiento de datos
- **Modos inactivos o baja energía**

La duración de la batería puede aumentarse utilizando los siguientes enfoques para reducir la corriente media consumida:

- **Reducir la corriente del modo activo:** reducir el consumo en todas las operaciones activas (recepción, transmisión y procesamiento de datos, entre otras).
- **Reducir el tiempo de modo activo:** reducir el tiempo empleado en las operaciones de CPU o RF para que el sistema pueda estar en los estados de baja energía con más frecuencia.
- **Modificando los protocolos de comunicación:** lo que reduce a su vez el tiempo de modo activo.

Algunos dispositivos, como es el caso de la placa utilizada, disponen de distintos modos de sueño a elegir (Ver apartado 2.2.2.b), siendo posible elegir el más indicado para cada situación y disminuir aún más el consumo.

a. Reducir la Corriente del modo Activo

- **Bajar la frecuencia de reloj de la CPU:** con frecuencias más bajas se reduce la corriente.
- **Cerrar recursos no utilizados:** la aplicación puede poner permanentemente los periféricos y relojes no utilizados en los modos de bajo consumo disponibles.
- **Utilizar la integración de nivel de chip:** reduce el consumo de corriente del sistema global debido a un rendimiento mejorado y una conmutación de E/S (Entrada/Salida).
- **Reducir la potencia de transmisión:** a menor potencia de transmisión es menor el consumo, pero también disminuye la distancia a la que se puede transmitir

b. Reducir el tiempo en modo activo

- **Programar actividades alrededor del evento de conexión:** con ello aumenta el tiempo en el que el dispositivo puede estar en modo *Deep-Sleep* (Sueño profundo, estado de mínima energía). Sin embargo, debe evitarse realizar cualquier otra acción en paralelo a la comunicación, ya que aumentaría la corriente de pico. Como se verá más adelante, esto afecta adversamente a la duración de algunos tipos de baterías.
- **Tasas de barrido de sensor más bajas:** si no se detecta ningún uso o actividad del sensor durante algún tiempo, el dispositivo puede reducir la velocidad a la que busca la actividad de la red para poder entrar en modos de baja potencia durante más tiempo. Por ejemplo, escanear sensores alternos en lugar de todos los sensores para detectar un movimiento puede ahorrar energía.
- **Bajar las tasas de muestreo de datos:** la frecuencia de muestreo del ADC en el dispositivo debe reducirse a la frecuencia mínima requerida para reproducir efectivamente la señal analógica que se muestrea. Además, la resolución del canal también se puede reducir para disminuir el tiempo de conversión y así ahorrar energía.
- **Sniffing[4]:** se utiliza para reducir el consumo de energía del dispositivo, ya que el receptor puede ponerse en espera entre ciclos de *sniff*.

c. Modificar los protocolos de comunicación

- **Reducción del consumo por ciclo de trabajo:** la idea aquí es que no hay necesidad de que todos los nodos tengan su radio activa en todo momento. Los nodos desactivarán su radio

[4] Sniffing (oler) es un proceso de escucha de tipos específicos de comandos que se producen periódicamente, como por ejemplo paquetes de datos que espera recibir un sensor de otro en un intervalo fijo.

cuando no hay actividad de red y se despertarán dependiendo del esquema elegido, reduciendo así el consumo.
Dado que los nodos están durmiendo para conservar la energía, no siempre podemos usar las mismas rutas al enviar un mensaje, por lo que deben crearse diferentes protocolos.

- **Compresión:** comprimir los datos antes de ser transmitido.

1.5.4. MÉTODOS DE MEDIDA

En este apartado se exponen tres de los métodos más utilizados para la medida de consumo de la batería.

a. Medir la tensión en la descarga

Se alimenta la unidad con baterías cargadas al máximo y se mide el periodo que permanece en funcionamiento. La tensión se obtiene durante el proceso, pero hay que tener en cuenta que, por ejemplo, en el caso de las baterías de litio, la tensión no es lineal, teniendo una caída muy abrupta en sus últimos instantes. [12]

Se trata de un método poco preciso debido al comportamiento no lineal de muchos tipos de baterías con respecto al voltaje como las de ión Litio en un rango de valores intermedio. [16][20]

b. Coulomb counting

Es calificado como el método más preciso para la estimación del SoC debido a que es posible la medida directa de la corriente de carga/ descarga de la batería. Básicamente, este método integra en el tiempo la intensidad que carga y/o descarga la batería, dando como resultado la carga almacenada. Para ello es necesario conocer el estado inicial (C_1), por lo que necesita apoyarse de algún otro método.

La fórmula para calcular la capacidad relativa (C_x) en un intervalo de tiempo ($t_2 - t_1$) conocido es la siguiente (*ecuación 1*):

$$Cx = \int_{t_1}^{t_2} I(t)dt$$

Ecuación 1. Capacidad relativa

Donde (I) es la corriente, (C) es la variación de la capacidad de la batería respecto a un tiempo inicial (t_1) y un tiempo final (t_2).

Considerando la corriente de descarga negativa y la de carga positiva, capacidad resultante (C_f) será la resta de (C_x) a la capacidad inicial (Ci) en (t_1)(*ecuación 2*):

$$C_f = Ci + C_x$$

Ecuación 2. Capacidad final

El estado de carga SoC de la batería será la relación entre la capacidad tras ese tiempo y la capacidad nominal en la batería (Cn), dato proporcionado en la hoja de características del fabricante (*ecuación 3*):

$$SoC = \frac{C_f}{C_n}$$

Ecuación 3. Estado de Carga (SoC)

Sin embargo, este modelo se ve afectado por el consumo irregular de corriente. Al basarse en una integración, los errores son directamente proporcionales a la diferencia entre el tiempo inicial y final. [12] [16][20][22][20]

Para medir la corriente fluyente pueden utilizarse varios sensores [20]:

- **Resistencia "shunt" (resistencia de medida):** mide la diferencia de tensión en una resistencia de valor bajo, de alta precisión, que no cause pérdidas de potencia alta. *Para valores de intensidad muy bajos no es muy preciso.*
- **Efecto "Hall":** consiste en el uso de transductores para medir la corriente. Son susceptibles a ruido y *no aceptan intensidades altas además de ser muy caros*.
- **GMR:** Sensores magnetoresistivos. Son *más caros* pero obtienen un nivel de señal más alto y son más estable a altas temperaturas. Su sensibilidad es mayor.

c. *Mediante la resistencia interna de la batería*

La degradación de los componentes internos de la batería debido a los ciclos de carga y descarga, tiene una relación directa con el valor de impedancia interna (IR). La IR tiene una componente eléctrica debida a los materiales y contactos y una componente iónica debida a factores electroquímicos como la conductividad de los electrolitos. [12] [18]

Para una primera aproximación suele ser suficiente conocer la parte resistiva de la impedancia interna. Para ello, se pueden aplicar distintos niveles de corriente ($I_1, I_2 \dots I_n$) y medir la tensión en circuito abierto en cada nivel. La tensión ($V_1, V_2 \dots V_n$) de la batería disminuirá a mayor demanda de corriente. Así la resistencia se puede calcular como [12] (*Ecuación 4.*):

$$R = \frac{V_2 - V_1}{I_2 - I_1}$$

Ecuación 4. Medida de la IR

Estas pruebas se comparan con las especificaciones de la célula, pero para los cálculos se utilizan los datos medidos.[18]

d. *Combinación de técnicas*

Debido a que todas las técnicas tienen sus limitaciones, algunos autores combinan Coulomb Counting con la medida de tensión.

Mientras que mediante Coulomb Counting se obtiene la carga relativa de la batería, con la monitorización de la tensión se calibra el SoC cuando se acerca a uno de sus extremos ya que es en los puntos limites (0-100% SoC) donde la medida de tensión es más significativa. [12].

1.4.5. PROCEDIMIENTO DE MEDIDA Y ANÁLISIS DE CONSUMO DE NODOS INALÁMBRICOS

a. *Medidas a realizar* [12] [13] [16] [21] [18]

Un perfil de carga común de dispositivos que se comunican por RF puede simplificarse para tener 4 estados: sueño, pre-procesamiento, **RX/T**X (*Reception/Transmission*) y post-procesamiento, donde varía la corriente.

Aunque en la mayor parte del tiempo el microprocesador se encontrará en un estado de baja energía (si es que hemos desarrollado bien la aplicación), los eventos de conexión y la activación de los distintos dispositivos que lo componen producirán picos de corriente que aumentarán significativamente el valor promedio.

Aunque en la mayoría de artículos leídos se enfocan en medir estrictamente la corriente consumida sin tener en cuenta elementos como los LEDs, es importante estar al tanto de otras fuentes de consumo de corriente, ya que afectarán la duración de la batería.

Para el caso que nos ocupa, es necesario tomar una serie de consideraciones especiales:

1. **Tiempo de procesamiento:** el tiempo total de procesamiento para cada evento de conexión no siempre es exactamente el mismo. Esto significa que es necesario realizar varias pruebas. Sin embargo, no es completamente azaroso, sino que tiene distintos valores fijos. Para un análisis exhaustivo, **se debe determinar el porcentaje de tiempo en que se produce cada uno de estos casos o calcular el promedio con el tiempo más corto y el más largo.**
2. **Funciones no relacionadas con la comunicación:** dependiendo del dispositivo y la aplicación se tendrán una serie de componentes a los que será necesario aportar energía. Por tanto también habrá que tener en cuenta la corriente media que consumen.

Para realizar mediciones, se deben dividir las secciones de la forma de onda, con la corriente y el tiempo de cada estado:

Medir la corriente durante el intervalo de conexión: deben realizarse las medidas para el caso de mayor y menor duración.

Medir la corriente de los modos de reposo o de bajo consumo: esto es importante para la vida de la batería, porque en la mayoría de los casos de uso se pasa la mayor parte del tiempo en estos modos.

Otras medidas que sean significativas en nuestra aplicación en cuestión.

b. *Instrumentos de medida utilizados* [22]

Para obtener un valor aproximado del consumo medio de la aplicación, debemos de tener en cuenta de que pese a que en la mayor parte del tiempo se encontrará en un estado de baja energía, en los intervalos de comunicación se producirán picos de corriente que aumentarán de forma significativa la corriente media.

Durante el tiempo de sueño, se puede emplear un multímetro digital que proporcione medidas precisas en el rango de los microamperios y de los miliamperios.

Por otro lado, dado que los eventos de comunicación son de corta duración, para medir la corriente se puede emplear un osciloscopio con sonda de voltaje y una resistencia de 10Ω. Como dichos eventos no siempre duran lo mismo, se pueden realizar varias pruebas obteniendo el caso de tiempo máximo y mínimo. Midiendo la corriente de ambos casos y realizando el promedio obtendremos un valor bastante preciso.

Por último, habrá otros dispositivos que también demandarán corriente, que están fuera del estudio debido a la complejidad que supondría tener en cuenta la infinidad de casos posibles. Sin embargo debe tenerse presente de que el valor obtenido en el cálculo de consumo y duración siempre será mayor que el real.

c. *Fórmulas y cálculos* [16][16]

Para obtener el consumo medio y hacer una estimación de la duración de la batería en eventos de conexión se pueden seguir los siguientes pasos:

1. Calcular el consumo de corriente promedio durante el evento de conexión, una vez para el caso de tiempo más largo y otra para el más corto (*ecuación 5.*).

$$\bar{I}(\text{ev. conexión}) = \frac{\sum \text{Estado n(t)} * \text{Estado n(i)}}{\text{Tiempo total de vigilia}}$$

Ecuación 5. I promedio durante el evento de conexión

2. Calcular la corriente media para todo el intervalo de conexión, que tiene en cuenta el tiempo durante el cual el dispositivo está durmiendo. Igual que en el paso anterior, se realiza una vez para cada caso (*ecuación 6.*).

$$\bar{I}(\text{total}) = \frac{[\text{t(conexión)} - \text{t (tot. despierto)}] * \bar{I}\,(\text{sueño}) + \text{t(espera)} * \bar{I}(\text{ev. conexión})}{\text{t(conexión)}}$$

Ecuación 6. I total del tiempo de conexión

3. Hacer el promedio ponderado de los dos promedios que fueron calculados previamente.

$$\bar{I}\text{ de conexión} = I(t_{min}) + I(t_{max})$$

Ecuación 7. Cálculo de I de conexión promedio

4. Calcular cantidad de tiempo que puede esperar que dure la batería mientras se ejecuta continuamente en una conexión:

$$\text{Duración funcionando continuamente en un estado conectado} = \frac{\text{Capacidad}}{\bar{I}\text{ de conexión}}$$

Ecuación 8. Duración de la batería con el dispositivo transmitiendo continuamente

d. Otras consideraciones: picos de corriente [18] [23] [13]

Los picos de corriente son aquellos momentos en los que la corriente aumenta de forma significativa respecto a un estado normal del nodo. Éstos disminuyen la capacidad de la batería debido principalmente a un aumento de la IR, lo que supone una caída de tensión con la que se puede llegar incluso a un nivel inferior al **FEP** (*Functional End Point*)[5].

En este tipo de aplicaciones los momentos de mayor corriente demandada pueden coincidir con los períodos de comunicación, que pese a durar unos pocos milisegundos, pasan de una corriente de 4,1 mA de un modo inactivo a 18,6 mA (en el caso de Atmega256rfr2) [26]. Sin embargo, estos picos también pueden ser producidos por cualquier otro dispositivo del micro, como por ejemplo un **LED** (*Light-Emitting Diode*).

Para minimizar los efectos en la capacidad de la batería y evitar fallos debido a corrientes altas, se puede [13]*:*

- Maximizar el margen de FEP, que estará determinado por el dispositivo que demande más tensión.

[5] El FEP hace referencia al mínimo valor de voltaje con el cual puede funcionar la aplicación. Si, por ejemplo, tenemos un dispositivo con cuya demanda máxima de tensión se produce con la activación de determinados periféricos y es de 3.3V, ese será el FEP

- Minimizar la corriente media:
  - Corrientes de pico más bajas
  - Reducir su duración
  - Aumentar el período de las comunicaciones y la duración de los pulsos
- La adición de un condensador permite que un circuito maneje una alta resistencia interna y maximice la capacidad de la batería.

1.4.6. SENSORES Y APLICACIONES PARA LA MEDIDA DEL CONSUMO

Actualmente se encuentran en el mercado muchos productos para la medida del consumo. Cada uno adaptado a la cargar que se necesita medir, existe desde productos con sus propias aplicaciones, monitores y cuadros de mando para la visualización en tiempo real, hasta sensores encapsulados de dimensiones milimétricas para integrar en un microprocesador.

En cuanto a este proyecto se refiere, se buscaban principalmente sensores de corriente capaces de establecer comunicación I2C con la placa a analizar, de pequeñas dimensiones y bajo consumo. En el TFG de Óscar Escobar [19] se analizan sensores como el "Ina219" de **Texas Instruments** o el "Ltc2990" de **Linear Technology**.

Finalmente, se eligió el sensor de corriente PAC1710 de Microchip, del que se hablará más adelante, en el apartado

1.6. ESTRUCTURA DEL DOCUMENTO

A continuación se describe brevemente el contenido de cada capítulo:

CAPÍTULO 1: INTRODUCCIÓN

En este capítulo en primer lugar se hablará de la motivación, los objetivos y la metodología del proyecto el estado de la técnica en sus distintos. Seguidamente, se hablará sobre el estado de la técnica de estas tecnologías. Se presentará la información obtenida de distintos *papers* (publicaciones) que tratan sobre el uso eficiente de Redes de Sensores, medición y reducción del consumo y las baterías de uso típico en dispositivos inalámbricos. Finalmente, se describirá la estructura del documento.

CAPÍTULO 2: PROCESO DE DISEÑO DE UN PROTOTIPO PARA LA MEDIDA DE CONSUMO

Este capítulo es uno de los más importantes, pues en él se explicará paso a paso el proceso de diseño llevado a cabo para el desarrollo de la aplicación, describiendo cada elemento utilizado. Se hablará del microcontrolador del cual se pretende medir el consumo, la técnica de medida utilizada y el sensor de corriente utilizado. En cuanto a diseño se hablará de la PCB diseñada y los drivers de comunicación I2C. Finalmente, se describirá el procedimiento seguido para la obtención y el análisis de los datos.

CAPÍTULO 3: SISTEMA DE MEDIDA DEL CONSUMO DEL NODO PARA DISTINTOS ESTADOS

En esta sección se explicarán los montajes realizados para medir el consume de la placa y del micro, según de la forma en la que se adquieran los datos. Se hará un recorrido por las distintas pruebas de concepto con el polímetro, osciloscopio, placa de evaluación y sensor, hasta llegar a la configuración óptima de medida con los medios disponibles.

CAPÍTULO 4: RESULTADOS OBTENIDOS

Capítulo esencial donde se exponen los resultados conseguidos y a dónde se ha conseguido llegar tras todo el proceso de diseño, adquisición de datos y análisis de la información tras el procesado. Todo ello se resumirá en una comparativa del consumo en distintos modos de funcionamiento para medidas de 1/10/60 minuto(s) del nodo completo y del micro de la placa.

CAPÍTULO 5: CONCLUSIONES

En este último capítulo se exponen las conclusiones a las que se ha legado tras la realización del proyecto, así como posibles mejoras y líneas futuras de desarrollo.

ANEXO

En los anexos se incluirá todo el código comentado propio creado para el desarrollo de los drivers del sensor y la realización de las medidas. También se dedicarán aquí unos apartados para la **planificación**, el **marco regulador**, el **entorno socioeconómico** y los **presupuestos**.

2. PROCESO DE DISEÑO DE UN PROTOTIPO PARA LA MEDIDA DEL CONSUMO

En este capítulo se explicará paso a paso el proceso de diseño del prototipo para la medida del consumo.

Tras un apartado inicial en el que se indican los principales elementos que forman el prototipo, se hablará de cada uno de los pasos seguidos, que son:

1. Conocer la placa y el microcontrolador que forman la unidad del nodo inalámbrico. Es importante estudiar las características básicas del dispositivo a medir, ya que de ello depende principalmente la elección de un sistema de medida.

2. Elegir el método de medida que mejor se adecúe a las características del dispositivo a medir y la disponibilidad de herramientas.

3. Escoger un sensor apropiado con el cuál se pueda tomar la información necesaria según el método de medida escogido.

4. Diseño del hardware necesario para formar el circuito de medición y la comunicación entre el sensor y la placa.

5. Diseño de los drivers y las aplicaciones necesarias tanto para la comunicación nodo-sensor como para la adquisición y el procesamiento de datos.

2.1. RESUMEN DE LOS ELEMENTOS UTILIZADOS

En este apartado se explican los elementos utilizados en el prototipo utilizado para la medida del consumo. Consta principalmente de las siguientes partes.

- **El nodo del cual queremos conocer su consumo:** se empleará durante todo el proyecto la placa **ATMEGA256RFR2 XPLAINED PRO** (*Figura 2.1.*). Estas placas son las que forman el sistema de nodos, siendo algunos nodos finales y otros nodos de enrutamiento.
  En algunos ensayos han sido necesarios dos nodos. Por ejemplo, no es posible medir un estado de sueño del nodo si se pretende obtener los datos de medida de consumo de la placa con ese mismo nodo. Así, una de las placas se comunica con el sensor, mientras que el consumo del otro nodo, con un código de test cargado, estará siendo medido por el sensor.

Figura 2.1 Placas ATMEGA256RFR2 Xplained Pro

- **El dispositivo con el cual se medirá el consumo del nodo:** debe ser capaz de medir la corriente demandada por el nodo para distintos modos de funcionamiento, sin afectar significativamente al consumo total de corriente del sistema en conjunto. (*Figura 2.2.*)

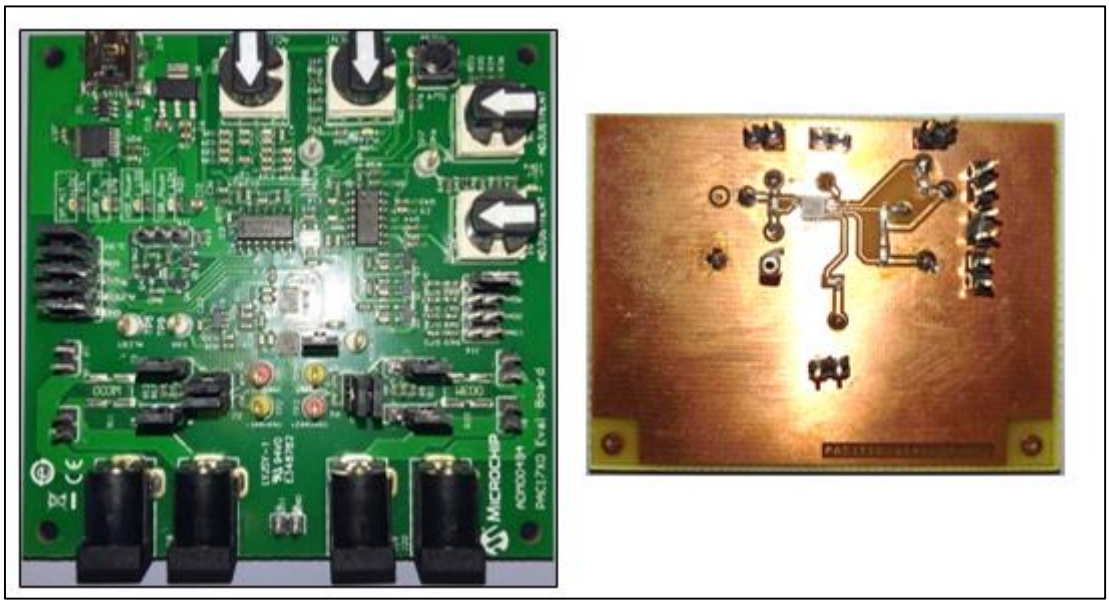

Figura 2.2. Placa de Evaluación PAC1710/20 y placa PAC1710 diseñada

- **Fuente de alimentación:** debe ser capaz de proporcionar suficiente energía como para que el sensor y el nodo funcionen correctamente durante los ensayos. Según el datasheet del nodo [24], la mínima tensión para el funcionamiento de todos los componentes de la placa es de 4,2 V. Por otro lado, en el datasheet del sensor de corriente [33], se indica que tanto la línea de alimentación (**Vcc**) como la de comunicación (**Vpp**) operan en el rango de 3 V a 5,5 V.
  Aunque en un primer momento se planteó la utilización de una pila de litio de botón (CR2450) y otra pila de litio recargable (LP-573-442-1S-3) para las pruebas, se terminó utilizando la fuente de alimentación del laboratorio de microelectrónica (modelo "*Gold Source df173158")* . Como alternativa, se alimentó tanto la placa de evaluación del sensor de corriente como el nodo a través de los puertos **USB** (Universal Serial Bus) del ordenador portátil.
  En la práctica, puede alimentarse el sensor y un nodo a través del ordenador y el otro nodo con la fuente de alimentación, siempre que se respete el rango de tensiones y corrientes entre los que operan y exista una referencia de masa común entre los elementos que se comuniquen por I2C (*Figura 2.3.*).

Figura 2.3. Fuente de alimentación empleada

- **Dispositivos y software para programar el nodo, obtener los datos y procesarlos:** para ello se han utilizados distintos ordenadores de uso personal. Como requisitos mínimos, el ordenador empleado debe tener instalados y mantener el funcionamiento durante la prueba de: **Termite v3.3**, **Atmel Studio v7.0**, **Microchip Chip Manager v4.16.8** (en el caso de querer medir con el software de Microchip) y **Packet Sniffer v2.18.**
- **Componentes intermedios:** resistencias de pull up, cables unipolares, protoboard, etc.
- **Instrumentos adicionales de medida:** polímetro, osciloscopio y sondas (*Figura 2.4*).

Figura 2.4. Polímetro y osciloscopio utilizados

En el siguiente apartado de este capítulo se describirá la placa de Atmega.

2.2. PLACA A MEDIR: ATMEGA256RFR2 XPLAINED PRO [24] [25]

ATmega256RFR2 Xplained Pro es una placa de evaluación de Atmel que integra el microcontrolador ATmega256RFR2. Dichas plataformas hardware son las que formaran los

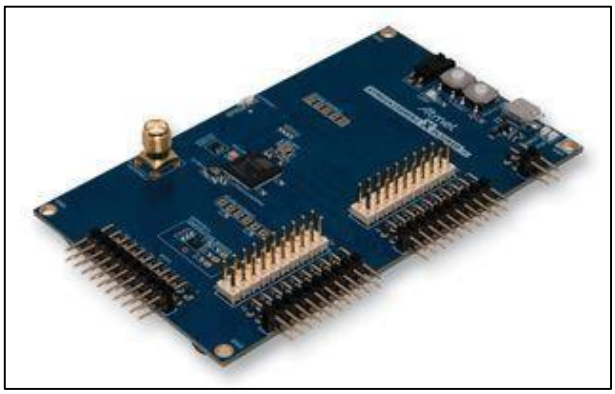

Figura 2.5 ATmega256RFR2 Xplained Pro

nodos que constituyen la red de la aplicación, y por tanto son a los que interesa realizar el estudio de consumo. (*Ver Figura 2.5.*)

Contiene los siguientes componentes:

- Microcontrolador ATmega256RFR2 de Atmel
- Interfaz USB
- Depurador integrado. Programación y depuración a través de la interfaz **JTAG** (*Joint Test Action Group*)[6]
- I/O Digitales. 5 conectores de extensión EXT 1-5, cada uno con 20 pines, entre los que destacan:
    - Pines de uso general
    - Pines de Vcc y de Gnd
    - Pines para comunicación I2C, **SPI** (*Serial Peripheral Interface*) y **UART** (*Universal Asynchronous Receiver-Transmitter*)
- Dos botones mecánicos (User y Reset)
- Un Led de usuario (User)
- Una antena de cerámica y un conector **SMA** (*SubMiniature version A*) para conectar una antena externa
- Sensor de temperatura
- Dos cristales: 16MHz y 32KHz
- Dos Fuentes de alimentación posibles:
    - Alimentación externa con pines "**PWR connector**"
    - Alimentación a través del PC por puerto USB

En la *Figura 2.6.* se pueden identificar los componentes que incorpora la plataforma.

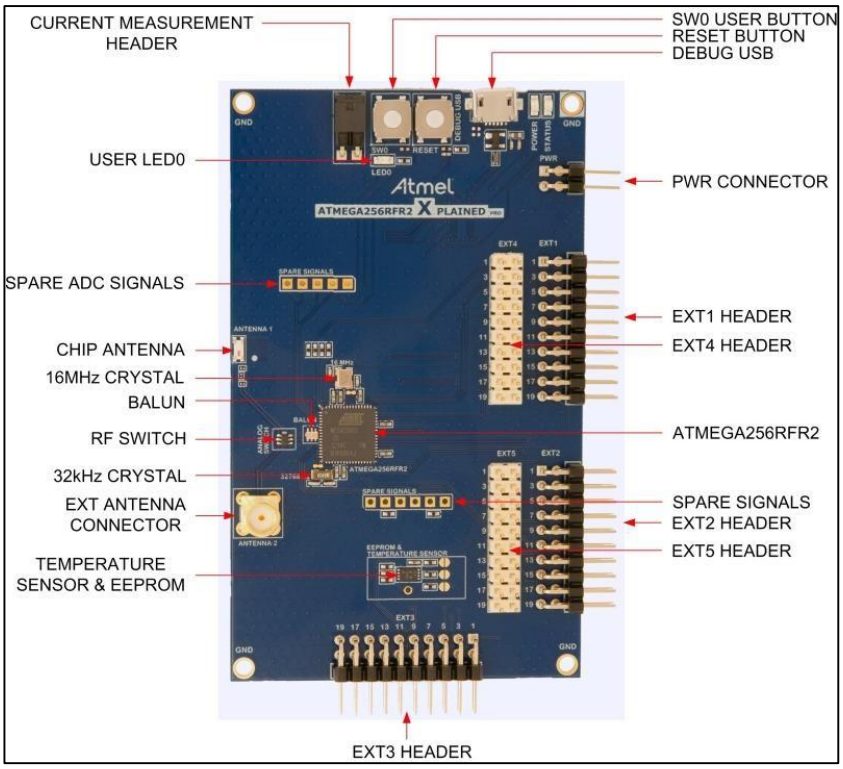

Figura 2.6. Componentes de ATmega256RFR2 Xplained Pro

[6] Consultar apartado del anexo A.5. Marco Regulador

2.2.1. *MICROCONTROLADOR ATMEGA 256RFR* [26]

El ATmega256RFR2 es un microcontrolador de 8-bits de alto rendimiento y bajo consumo, basado en la arquitectura AVR que incorpora un transceptor para la banda de 2.4GHz del protocolo **ZigBee**. (*Ver Figura 2.7*).

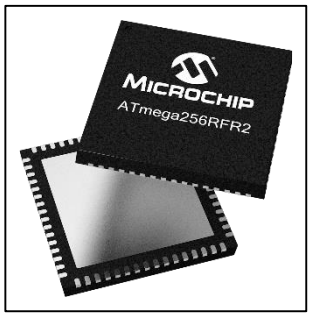

Figura 2.2.7. Atmega256RFR2

Las características del microcontrolador de Atmel son enumeradas a continuación:

- Arquitectura **RISC** (*Reduced Instruction Set Computer*)[7]:
  - 135 instrucciones
  - 32x8 registros de propósito general
  - Rendimiento de hasta 16 MIPS (Millones de Instrucciones Por Segundo) @16 MHz - 1.8V.
- Memoria de datos y programa:
  - 256 Kbytes de Flash, divida en:
    - Parte reservada para el programa de aplicación, dividida en 3 páginas, donde se almacena el código y las constantes
    - Parte reservada a proporcionar la capacidad de auto-programación
  - 8 KBytes de EEPROM de las cuales:
    - 32 registro de propósito general (Posiciones 0000h-001Fh)
    - Memoria de I/O (Posiciones 0020h-005Fh)
    - Memoria de I/O externas (Posiciones 0060h-01FFh)
  - 32 KBytes de memoria de datos interna **SRAM** (*Static Random Access Memory*) de lectura/ escritura, donde se almacenarán, por ejemplo, todas las variables de la aplicación (posiciones 0200h-81FFh)
- Interfaz JTAG
- Periféricos:
  - Múltiples temporizadores/contadores y canales **PWM** (*Pulse-Width Modulation*).
  - Contador en tiempo real con oscilador independiente.
  - Conversor analógico/digital 10-bit, 330 ks/s.
  - Comparador analógico.
  - Interfaz SPI Maestro/Esclavo.
  - Dos USART (*Universal Synchronous and Asynchronous Receiver-Transmitter*) programables.
  - Interfaz TWI.
- 38 pines de entrada y salida.
- Transceptor de baja potencia totalmente integrado para banda **ISM** (*Industrial, Scientific and Medical*) de 2,4 GHz.
  - Velocidades de datos soportadas: 250 kb/s y 500 kb/s, 1 Mb/s, 2 Mb/s.

[7] "La arquitectura RISC limita el número de instrucciones incorporadas en el microprocesador, pero optimiza cada una de ellas de forma que se ejecuten muy rápidamente (generalmente en un solo ciclo de reloj). Por lo tanto, los chips RISC ejecutan las instrucciones simples más rápidamente que los microprocesadores que cuentan con un conjunto más amplio de instrucciones" [28]

  - 100 dBm Sensibilidad de recepción.
  - Potencia de transmisión hasta 3,5 dBm.
- Seguridad del hardware (*AES, True Random Generator*).
- Osciladores de cristal integrados (32.768 kHz y 16 MHz, se necesita cristal externo).
- Consumo de energía muy bajo (1.8 a 3.6V) para AVR y Rx / Tx: 10.1mA / 18.6 Ma.
  - Consumo de energía reducido asistido por hardware avanzado.
  - Manejo avanzado de interrupciones y modos de ahorro de energía.
  - Modo de CPU Active (16MHz): 4,1 mA.
  - Transceptor de 2,4 GHz: RX_ON 6,0 mA / TX 14,5 mA (potencia máxima de salida TX).
  - Modo de sueño profundo: <700nA @ 25 ° C.
- Grado de velocidad: 0 - 16 MHz a 1.8 - 3.6V (modificable con los reguladores de voltaje integrados).

2.2.2. CONSUMO DE ENERGÍA Y MODOS DE FUNCIONAMIENTO [26]

Una de las características más interesantes del ATMEGA256RFR2 es la cantidad de modos de sueño configurables que posee. Éstos son modos de baja energía en los que se debe mantener el nodo en cualquier momento en el que no esté realizando alguna acción (tomando datos, comunicando, etc.) en el caso de que busquemos que nuestra aplicación tenga el menor consumo posible.

En la *Figura 2.4* obtenida de [26] se puede ver el consumo del microcontrolador a 16 MHz, en los estados más comunes de este tipo de aplicaciones:

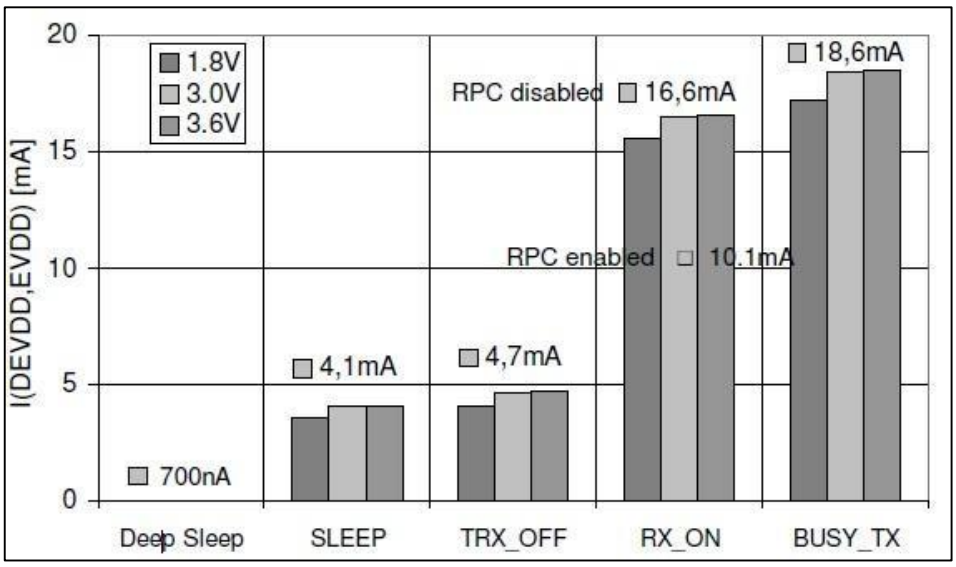

Figura 2.8. Consumo de Atmega256rfr2 para distintos estados del transceptor

Como se puede observar, el sistema incluye la posibilidad de regular el voltaje interno, siendo menor el consumo a menor voltaje. Ésta gráfica se ha tomado de referencia a la hora de analizar los datos obtenidos, teniendo en cuenta que la aplicación en la que se han realizado las medidas operaba a 8 MHz y 3,3 V.

a. Descripción de los modos de funcionamiento

Esta sección resume todos los estados para proporcionar la funcionalidad básica del transceptor de radio de 2,4 GHz, como la recepción y la transmisión de marcos, la secuencia de encendido y el transceptor de radio de sueño. El modo operativo básico está diseñado para aplicaciones IEEE 802.15.4 e ISM.

El diagrama de estados de estos modos se puede ver en la *Figura 2.9.*, correspondiente a la Figura 3.2 del datasheet [26].

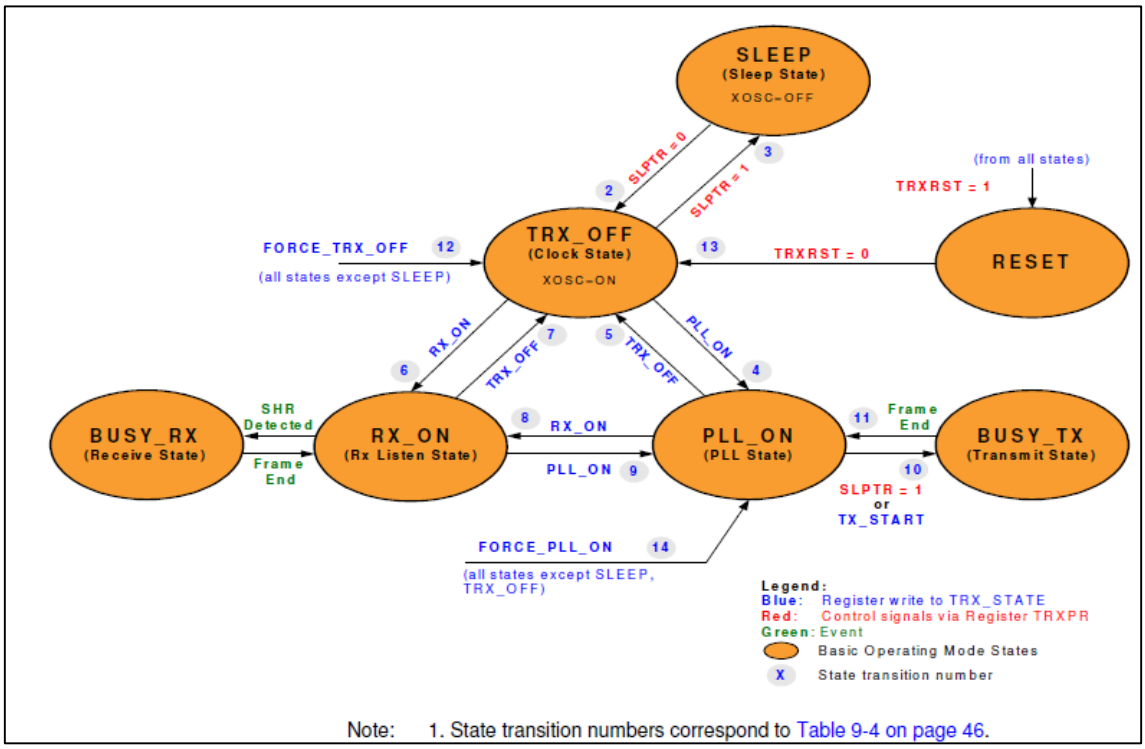

Figura 2.9. Diagrama de estados para los principales modos de Atmega256rfr

PRINCIPALES ESTADOS DEL MICROCONTROLADOR

Estado de sueño (SLEEP)

En los estados de SLEEP todo el transceptor de radio está deshabilitado. No hay circuitos en funcionamiento y el consumo de corriente se reduce a la corriente de fuga. Este estado sólo se puede introducir desde el estado TRX_OFF, ajustando el bit SLPTR a "1".

Ajustando el bit SLPTR a "0" o el bit TRXRST a "1" se pasa al estado TRX_OFF.

En el apartado siguiente (b. Descripción de los modos de sueño) se explica con más detalle cada modo de sueño.

Estado de Reloj (TRX_OFF)

Este estado se alcanza inmediatamente después de encender el micro o restablecerlo. En TRX_OFF el oscilador de cristal está en funcionamiento.

La entrada del estado TRX_OFF desde el estado SLEEP o el estado RESET se indica mediante la interrupción TRX24_AWAKE.

Estado PLL (PLL_ON)

El estado PLL_ON corresponde al estado TX_ON en IEEE 802.15.4. Al ingresar el estado PLL_ON desde el estado TRX_OFF se activa primero el regulador de voltaje analógico (**AVREG**). Tras esto, se habilita el sintetizador de frecuencia PLL.

Cuando el PLL se ha establecido en la frecuencia de recepción a un canal definido por los bits CHANNEL del registro PHY_CC_CCA, se produce la interrupción TRX24_PLL_LOCK.

Si se emite un comando RX_ON en estado PLL_ON, el receptor se activa inmediatamente. En otro caso, en primer lugar se espera a dicha interrupción.

Estado de Escucha y Recepción (RX_ON y BUSY_RX)

En estos estados el receptor "escucha" las tramas de datos, manteniéndose habilitados el receptor y el sintetizador de frecuencia PLL. Se puede acceder al estado RX_ON escribiendo un comando de cambio de estado en los bits TRX_CMD del registro TRX_STATE.

Después de detectar un encabezado de sincronización válido (**SHR**, *Synchronization Header*)[8], el receptor entra automáticamente en el estado BUSY_RX. La recepción de un encabezado PHY válido (PHR) genera una interrupción TRX24_RX_START y recibe y desmodula los datos **PSDU** (*PHY Service Data Unit*)[9].

Durante la recepción de PSDU los datos de trama se almacenan continuamente en el buffer de datos hasta que se recibe el último byte. La finalización de la recepción de la trama se indica mediante una interrupción TRX24_RX_END y el transceptor de radio vuelve a entrar en el estado RX_ON.

Estado de transmisión (busy_tx)

Una transmisión sólo puede iniciarse en el estado PLL_ON. Hay dos formas de iniciar una transmisión:

- Bit SLPTR del registro TRXPR a '1'.
- Comando TX_START en los bits TRX_CMD del registro TRX_STATE

Durante la transición al estado BUSY_TX, la transmisión real del primer paquete de datos comienza después de 16 µs para permitir el establecimiento de PLL y el incremento de **PA** (*Power Amplifier*)[10]. Después de la transmisión del SHR, se transmite el contenido del Frame Buffer (buffer donde se almacena la información a transmitir). En caso de que el **PHR** (*PHY header*)[11] indique una longitud de trama de cero, la transmisión es abortada.

Una vez finalizada la transmisión, el transceptor de radio apaga automáticamente el amplificador de potencia, genera una interrupción TRX24_TX_END y vuelve al estado PLL_ON.

Estado de RESET

El estado RESET se utiliza para restaurar la máquina de estados (devuelve al estado TRX_OFF) y restablecer todos los registros del transceptor de radio a sus valores predeterminados. Un reset devuelve al estado TRX_OFF.

Modo Suspensión Profunda (Deep Sleep Mode)

En el modo de suspensión profunda (Deep sleep mode) todos los bloques digitales principales sin requisitos de retención de datos están desconectados del suministro principal,

[8] "El SHR consiste en un campo de preámbulo de cuatro octetos (todo cero), seguido por un delimitador de comienzo de trama de un byte (SFD) que tiene el valor predefinido 0xA7. Durante la transmisión, el SHR es generado automáticamente por el transceptor de radio, por lo tanto, el buffer de marco debe contener PHR y PSDU solamente." [26]

[9] El PSDU es de tamaño variable y contiene los datos de capa de protocolo MAC [26]

[10] Amplificador de potencia externo necesario para la transmisión por RF

[11] "El encabezado PHY header es un octeto único que sigue al SHR. Los 7 bits menos significativos indican la longitud de trama de la siguiente PSDU, mientras que el bit más significativo de ese octeto está reservado y se pondrá a 0 para tramas compatibles con IEEE 802.15.4." [26]

proporcionando una corriente de fuga muy pequeña. El temporizador Watchdog[12], el contador de símbolos MAC[13] y el oscilador de 32.768kHz pueden configurarse para continuar ejecutándose.

CAMBIOS DE ESTADOS

Los cambios de estado se realizan o bien escribiendo comandos a los bits TRX_CMD del registro TRX_STATE, o modificando los dos bits de control SLPTR y TRXRST del registro TRXPR. Para comprobar que se ha cambiado de estado correctamente, se puede leer el estado del transceptor en el registro TRX_STATUS.

1. **SLPTR:** El bit SLPTR es un bit multifuncional. Si pasa de "0" a "1", causa las siguientes transiciones:
   - TRX_OFF→ESTADO DORMIDO
   - PLL_ON→ BUSY_TX

   Mientras que el paso a "0" produce:

   - ESTADO DORMIDO → TRX_OFF
2. **RESET:** El bit TRXRST causa un reinicio de todos los registros del transceptor de radio y obliga al transceptor de radio a pasar al estado TRX_OFF.
3. **FORCE_TRX_OFF:** Para todos los estados excepto SLEEP, los comandos de cambio de estado FORCE_TRX_OFF o TRX_OFF conducen a una transición al estado TRX_OFF.

Si el transceptor de radio está en estado activo de recepción o transmisión (BUSY_ TX o BUSY_RX), el comando FORCE_TRX_OFF interrumpe estos procesos activos y obliga a una transición inmediata a TRX_OFF. En cambio, un comando TRX_OFF (Paso de 1 a 0 del bit SLPTR) se almacena hasta que se ha terminado un estado activo (recepción o transmisión). Después se realiza la transición a TRX_OFF.

4. **FORCE_PLL_ON:** A diferencia del anterior, este comando no desactiva el PLL (*Phase Locked-Loop*)[14] y el regulador de voltaje analógico AVREG, siendo la transición a PLL_ON más rápida. No está disponible para los estados SLEEP y RESET.

b. Descripción de los modos de sueño

El ATmega256RFR2 tiene 6 modos de ahorro de energía seleccionables por software.

- El modo de inactividad (**Idle mode**) detiene la CPU y mantiene activos la SRAM, el temporizador, el puerto SPI y el sistema de interrupción.
- El modo de apagado (**Power-down mode**) guarda el contenido del registro pero detiene el oscilador, deshabilitando todas las demás funciones del chip hasta la siguiente interrupción o reinicio.
- En el modo de ahorro de energía (**Power-save mode**), tan sólo el temporizador asincrónico permanece activo

[12] El Watchdog es un mecanismo de seguridad que consiste en un contador que se va decrementando hasta 0. Si el contador no se reinicia en la subrutina del Watchdog y llega a 0, será significativo de algún fallo o permanencia en bucle, y se reiniciará el sistema. [29]

[13]El Contador de símbolos MAC proporciona información de temporización de símbolos para redes inalámbricas IEEE 802.15.4. Se usa por ejemplo como contador de tiempo o para despertar al micro de un modo de sueño [26]

[14] "El circuito PLL genera una señal de salida de amplitud designada y misma frecuencia que la frecuencia de entrada." [30]

- El modo Reducción de ruido o ADC (**Noise Reduction mode**) detiene la CPU y todos los módulos de E / S excepto el Temporizador asincrónico y ADC.
- En el modo de espera (**Standby Mode**), el oscilador RC (Resistencia Capacitor)[15] está funcionando mientras el resto del dispositivo está durmiendo., lo que combina un arranque muy rápido con un bajo consumo.
- Por último, en el modo espera extendida (**Extended Standby Mode**), el oscilador RC principal y el temporizador asíncrono continúan ejecutándose.

En la *tabla 2.1* (tabla 12-1 del datasheet [26]) se detallan las señales de reloj, osciladores y fuentes despertadoras activas en cada modo:

	Active Clock Domains					Oscillators		Wake-up Sources								
Sleep Mode	clkCPU	clkFLASH	clkIO	clkADC	clkASY	Main Clock-source Enabled	Timer Oscillator Enabled	INT7:0 and Pin Change	TWI Address Match	Timer/Counter2	SPM/EEPROM Ready	ADC	WDT Interrupt	Other I/O	Symbol Counter	Transceiver
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	X(4)	X(4)
ADCNRM				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		X(4)	X(4)
Power-down								X(3)	X				X		X(4)	X(4)
Power-save					X		X(2)	X(3)	X	X			X		X(4)	X(4)
Standby (1)						X		X(3)	X				X		X(4)	X(4)
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X(4)	X(4)

Tabla 2.1 señales de reloj, osciladores y fuentes despertadoras activas en cada modo

2.3. PROPUESTA DE MEDIDA Y MÉTODO DE ANÁLSIS EMPLEADO

El método inicialmente propuesto para conocer el consumo es la medida de corriente para distintos modos del nodo. Se medirá el consumo para los distintos modos de sueño y en estados de alta energía, como por ejemplo durante la transmisión de datos.

Una primera toma de datos podría realizarse conectando una resistencia de 10 Ω entre los pines "**CURRENT MEASUREMENT HEADDER**" de la placa (Ver *Figura 3.4*)[24]. Estos pines están puenteados habitualmente ya que conectan la alimentación de la placa desde el USB con la alimentación del microprocesador. [16] Así por ejemplo, si el multímetro midiese una tensión de 30mV en los bornes de la resistencia, aplicando la Ley de Ohm se puede conocer el valor de corriente demandado por el micro (I=V/R=30/10= 3mA).

No obstante, esta forma de medir no es posible en el caso de los intervalos en los que se comunica el nodo, con duración de ms. Para una primera visualización de la onda de tensión equivalente al consumo, se puede emplear un osciloscopio con sonda de voltaje en lugar del

[15] Los osciladores RC emplean una resistencia y un condensador en serie para generar una onda alterna. Suelen constar de un elemento amplificador y un circuito de realimentación.

[16] Sin embargo, en necesario tomar la precaución de comprobar que se está alimentando el microprocesador antes de iniciar la aplicación, dado que en caso contrario podría dañarse la placa de forma irreversible.

multímetro. Respecto a esta medida, se debe tener en cuenta que es necesario minimizar la longitud del cable entre el punto a medir y el osciloscopio, ya que podría falsearse la medida.[17]

Las anteriores propuestas de sistema de medida tan sólo sirven para una primera observación y análisis del sistema en el laboratorio, pero no para una aplicación funcional en una red de nodos. Para ello será necesario un sensor junto con una resistencia de Shunt y un circuito de adaptación capaz de obtener y almacenar el valor de corriente en un rango de mA.

Por último, a modo de análisis, con los resultados obtenidos, se podrá estimar por el método de Coulomb Counting el consumo de una hipotética batería y la duración de la misma dependiendo de la configuración. (Ver apartado 1.4.4.b.)

El sensor escogido finalmente para tal propósito es el PAC1710 de Microchip, del que se hablará en el apartado siguiente.

2.4. DISPOSITIVO ESCOGIDO PARA MEDIR: SENSOR PAC1710 DE MICROCHIP [33]

Los sensores PAC1710/20 de Microchip son sensores de corriente con cálculo de potencia integrado. Se escogió el primer modelo, el cual se diferencia del segundo en tener un canal más de medida. Sus características fundamentales en relación a la aplicación a desarrollar se mencionan a continuación. Una descripción más detallada de este sensor se puede encontrar en el Trabajo de Fin de Grado de Óscar Escobar [19].

- Medida de corriente a través de resistencia Shunt con tiempo de muestreo de 2.5ms a 2.6s y resolución de 8-11 bits. Tiempo de conversión, media de muestras, resolución y rango configurables
- Amplio rango de medida de tensión (0-40V). El rango de corrientes medible depende de la resistencia shunt y la caía del voltaje en ésta.
- Posibilidad de obtener el valor de potencia media
- 3 modos de funcionamiento (active, standby y one-shot)
- Comunicación SMBUS-I2C con dirección de esclavo seleccionable por resistencia externa
- Pin de alerta (sólo para SMBUS), límites mínimos y máximos de tensión y corriente configurables.[18]

2.4.1. MODOS DE OPERACIÓN

El PAC1710 / 20 tiene tres estados de operación:

- **Activo** – en este estado se realiza la medida de Vsouce y/o de Vsense, en función del valor de los bits 1 y 0 del registro 00h (Configuration Register). Donde:
  - **Vsource:** se almacenan en los registros de Vsource Hight y Vsource Low un valor proporcional a la tensión de la fuente.
  - **Vsense:** se almacenan en los registros Vsense Hight y Vsense Low un valor proporcional a la caída de tensión por la resistencia shunt.
- **Standby**: en este estado el sensor no realiza mediciones. La comunicación SMBUS/ I2C permanece activa. Es el estado de más baja energía. Para llegar a este estado en necesario configurar los bits 1 y 0 del registro 00h a 1.

[17] Los cables presentan una inductancia equivalente, cuyo valor L guarda relación con la longitud del cable. A mayor longitud se puede producir un mayor pico en el régimen transitorio, que puede falsear la medida al manejar rangos de muy baja tensión y corriente.

[18] El pin de alerta no se utilizará ya que sólo está disponible para SMBUS y sólo es posible comunicarse con el nodo por SPI o I2C.

- **One-Shot**: durante el modo Standby, escribiendo en el registro 02h (One-Shot Register), el sensor realiza una única medida de Vsense y Vsource. Al finalizar, vuelve al modo Standby.

A modo de resumen, el diagrama de estados de los modos del sensor PAC1710 se muestra a continuación (Ver *Figura.2.9.*)**:**

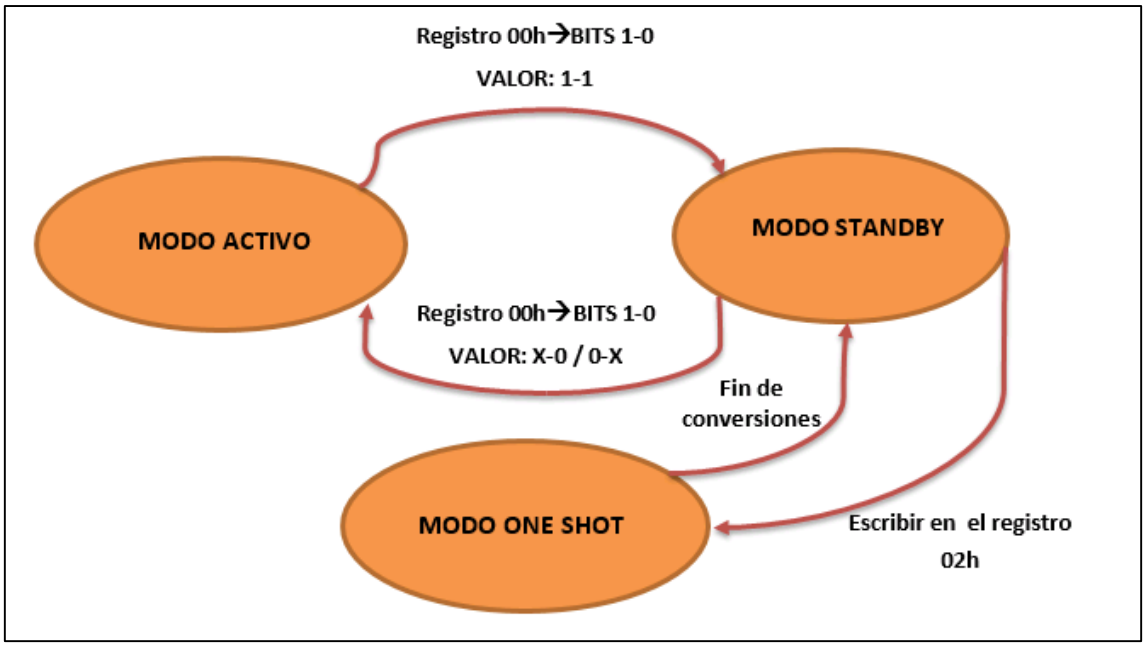

Figura 2.9 Diagrama de estados de los modos del sensor PAC1710

2.4.2. TASA DE CONVERSIÓN

La frecuencia de conversión controla la frecuencia con la que se realizan medidas y cuándo se actualizan los registros de los datos medidos en VSENSE, VSOURCE y PRATIO. Es posible configurar el sensor para que realice conversiones continuas o bien 1,2 o 4 veces por segundo. Cuando se termina un ciclo de conversión, es decir, cuando uno de los registros se actualiza, el bit 7 (CVDN) del registro 04h (Hight-Limit Status) se pone a 1.

La tasa de conversión sólo se puede cambiar cuando el sensor se encuentra en modo STANDBY.

2.4.3. MEDIDA DE CORRIENTE

El PAC1710, a través del canal 1 (SENSE+ y SENSE-) mide el voltaje (VSENSE) inducido a través de una resistencia de detección de corriente externa fija (RSENSE) y almacenan el voltaje como un número de 11 bits (por defecto) en los registros 0Dh y 0Eh.

La detección de corriente funciona según uno de los cuatro rangos configurables: (FSR): ± 10 mV, ± 20 mV, ± 40 mV o ± 80 mV (por defecto). Junto con el valor de la resistencia de shunt utilizada (Rsense), se puede calcular el fondo de escala de corriente.

$$FSC = \frac{FSR}{R_{SENSE}}$$

Ecuación 8. Cálculo de FSC

Con el valor de fondo de escala FSC, el dato en binario de la caída de tensión equivalente por la resistencia (V_{SENSE}) (Registros 0Dh y 0Eh) y teniendo en cuenta el tiempo de muestreo

configurado (determina el DEN_c, ver *Tabla 2.8.*), se puede calcular finalmente la corriente a través de R_{SENSE} (*Ecuación 9.* Cálculo de Ibus*):*

$$I_{BUS} = FSC * \frac{V_{SENSE}}{DEN_c}$$

Ecuación 9. Cálculo de Ibus

2.4.4. MEDICIÓN DEL VOLTAJE

La tensión que suministra la fuente de energía del nodo se mide a través del pin Vsense+ (ver Figura 2.10) y se almacena en los registros 11h y 12h (Ver apartado 2.3.2.-h). Se corresponde con la variable V_{SOURCE} de la Ecuación 11.

La tensión a fondo de escala (FSV) (ecuación 10), así como la tensión final (V_{SOURCE_PIN}) (ecuación 11) se cala con las siguientes fórmulas:

$$FSV = 40 - \frac{40}{DEN_A}$$

Ecuación 10. Cálculo de FSV

$$V_{SOURCE\_PIN} = FSV * \frac{V_{SOURCE}}{DEN_B}$$

Ecuación 11. Cálculo de la tensión de la fuente

El valor de DEN_A y DEN_B depende del número de bits utilizados en los registros 11h y 12h. Ello determina la resolución, que depende del tiempo de muestreo, como se muestra más adelante en la Tabla 2.10.

2.4.5. PROTOCOLO DE COMUNICACIÓN I2C [33] [34][35]

a. Concepto

El protocolo de comunicación **I2C** (*Interfaz Inter Circuitos*), desarrollado por **Philips** [35] permite la comunicación serie entre dos dispositivos en modo Half-Duplex. Sus características más destacables son:

- Lo forman dos hilos, el de transferencia de datos o SDA (*Serial Data*) y el de la señal de reloj o SCL (*Serial Clock*), más un hilo de referencia a masa.
- SDA y SCL disponen cada uno de una resistencia de PULL-UP (*Rpp*) conectadas a un nivel de tensión determinado (entre 2,2-10kΩ dependiendo de la tensión de alimentación (*Vpp*), el número de dispositivos o la capacidad del bus) que las mantiene a nivel alto en reposo.
- Por ejemplo la placa del sensor se ha diseñado con Vpp=3,3 V coincidente con la tensión de alimentación, y Rpp=ADDR_SEL=10kΩ.
- El número máximo de dispositivos conectados al bus viene limitada por su capacidad, de 400 pF. Estos pueden ser maestros (proporcionan la señal de reloj) o esclavos, siendo posible la existencia de varios maestros en una misma línea.
- Las velocidades van entre los 100 Kbits/s de máximo del modo estándar hasta los 3,4 Mb/s del “Fast Mode Plus”. En el caso del sensor, opera en el modo “Fast Mode” de 400Kbits/s.

b. Slave Address y ADDR_SEL

La resistencia ADDR_SEL estará conectada entre masa y el pin ADDR_SEL. En la siguiente tabla (*Tabla 2.2*), relacionada con la tabla 5-1 del datasheet se encuentran los valores de dirección según la resistencia.

VALOR DE LA DIRECCIÓN DE ESCLAVO SEGÚN EL VALOR DE ADDR_SEL (en Ω)					
ADDR_SEL	SLAVE_ADDR	ADDR_SEL	SLAVE_ADDR	ADDR_SEL	SLAVE_ADDR
0	98h	750	94h	5600	58h
100	9Ah	1270	96h	9100	5Ah
180	9Ch	1600	50h	20000	5Ch
300	9Fh	2000	52h	Abierto	30h
430	90h	2700	54h		
560	92h	3600	56h		

Tabla 2.2 Valor de la dirección de esclavo en función de la resistencia ADDR_SEL

c. Escritura

El bloque de escritura se utiliza para escribir varios bytes de datos a un grupo de registros contiguos, como se muestra en la *tabla 2.3.* (tabla 5-8 del datasheet [33]).

START	Slave Address	WR	ACK	Register Address	ACK	Register Data	ACK
1 → 0	YYYY_YYY	0	0	XXh	0	XXh	0
Register Data	**ACK**	**Register Data**	**ACK**	**Register Address**	**Register Data**	**ACK**	**STOP**
XXh	0	XXh	0		XXh	0	0 → 1

Tabla 2.3 Escritura I2C

Los pasos son los siguientes:

1. La condición de START se produce cuando SCL está a nivel alto durante la transición de 1 a 0 de SDA. Así comienza la comunicación y el maestro ocupa el bus.
2. El maestro envía por la línea de datos los 7 bits correspondiente a la dirección de esclavo + 1 bit que determina si va a leer o escribir en el registro, en este caso de valor 0 lógico. Es decir, con qué dispositivo quiere comunicarse y el tipo de comunicación.
3. Durante el noveno pulso de reloj el receptor envía el bit de reconocimiento **ACK** (*acknowledge*), manteniendo la línea SDA a nivel bajo. Por cada paquete de 1 byte hay un bit de ACK
4. Tras ello el maestro envía el byte o bytes relativos a la dirección del registro que se va a escribir
5. Por último envía el byte a escribir en el registro. Estos pasos se repiten hasta que se produzca la condición de STOP
6. La condición de STOP se produce cuando SCL está a nivel alto durante la transición de 0 a 1 de SDA. Con esta condición el dispositivo maestro deja libre el bus.

d. Lectura

El bloque de lectura se utiliza para leer varios bytes de datos de un grupo de registros contiguos, como se muestra en la *tabla 2.4* (tabla 5-9 del datasheet [33]).

START	Slave Address	WR	ACK	Register Address	ACK	START	Slave Address	RD	ACK	Register Data
1→0	YYYY_YYY	0	0	XXh	0	1→0	YYYY_YYY	1	0	XXh
ACK	Register Data	ACK	Register Data	ACK	Register Data	ACK	...	Register Data	NACK	STOP
0	XXh	0	XXh	0	XXh	0	...	XXh	1	0→1

Tabla 2.4 Lectura I2C

Para la lectura, los pasos 1 a 4 del caso de escritura se repiten. Tras ello:

5. Se vuelve producir la condición de START
6. El maestro envía la dirección de esclavo de nuevo. El octavo bit será esta vez un 1 lógico, que indica lectura.
7. Se envía el byte o bytes en los que se va a almacenar el byte del registro o los registros
8. La comunicación continua hasta que se produzca la condición de STOP

2.4.6. REGISTROS

En esta sección se explican los registros más importantes para la aplicación. Se detallarán la función de los bis utilizados de cada registro. Se clasifican en:

- **Registros de configuración:** establecen la configuración de modo del sensor, activación del pin de Alert, modo de conversión y tiempo de muestreo, rango y número de medias para Vsense y Vsource de cada canal.
- **Registros de límites:** guardan relación con los límites alto y bajo de Vsource y Vsense para cada canal.
- Registros de estado: su lectura permite conocer si se han superado los límites alto o bajo determinados en los registros de límites
- **Registro de Monitorización del bus:** permite visualizar los valores de Vsource, Vsense y pratio para cada canal
- **Registros de información del dispositivo:** contienen la información sobre la identidad del producto, la manufacturación y la revisión. Se han empleado principalmente para comprobar el correcto funcionamiento de la lectura a través de I2C, ya que son registros de sólo lectura invariables.

En la *Tabla 2.5.* se muestra un resumen de los registros. Se han resaltado aquellos que son utilizados en el código de los drivers.

1. REGISTROS DE CONFIGURACIÓN		
NOMBRE	**DIRECCIÓN**	**LECTURA/ ESCRITURA**
CONFIGURATION	00	LECTURA Y ESCRITURA
CONVERSION RATE REGISTER	01	
ONE SHOT REGISTER	02	
CHANNEL MASK REGISTER	03	
VSOURCE SAMPLING CONFIGURATION REGISTER	0A	
CHANNEL 1 VSENSE SAMPLING CONFIGURATION REGISTER	0B	
CHANNEL 2 VSENSE SAMPLING CONFIGURATION REGISTER	0C	
2. REGISTROS DE LÍMITES		
CHANNEL 1 VSENSE HIGHT LIMIT	19	SÓLO LECTURA
CHANNEL 1 VSENSE LOW LIMIT	18	
CHANNEL 1 SOURCE LOW LIMIT	1D	
CHANNEL 1 SOURCE LOW LIMIT	1F	
CHANNEL 2 VSENSE HIGHT LIMIT	1A	
CHANNEL 2 VSENSE LOW LIMIT	1C	
CHANNEL 2 SOURCE LOW LIMIT	1E	
CHANNEL 2 SOURCE LOW LIMIT	20	
3. REGISTROS DE ESTADO		
HIGHT LIMIT STATUS REGISTER	04	SÓLO LECTURA
LOW LIMIT STATUS REGISTER	05	
4. REGISTROS DE MONITORIZACIÓN DEL BUS		
CHANNEL 1 SENSE VOLTAGE	0D, 0E	SÓLO LECTURA
CHANNEL 1 BUS VOLTAGE	11, 12	
CHANNEL 1 POWER RATIO	15, 16	
CHANNEL 2 SENSE VOLTAGE	0F, 10	
CHANNEL 2 BUS VOLTAGE	13, 14	
CHANNEL 2 POWER RATIO	17, 18	
5. REGISTROS DE INFORMACIÓN DEL DISPOSITIVO		
PRODUCT ID	FD	SÓLO LECTURA
MANUFACTURER ID	FE	
SILICON REVISION ID	FF	

Tabla 2.5. Principales registros del sensor PAC1710

A. Registro 00h: Configuration (Ver *tabla 2.6*)

- **Los bits 7, 4 y 3** no están implementados en el PAC1710
- **Los Bits 6, 5 y 2** manejan funcionalidades del **PIN ALERT y SMBUS.** No se utiliza.

- **Bit 1 C1IDS:** Desactiva la medición VSENSE para el canal 1 si tiene valor 1 lógico. Desactiva la medición si vale 0.
- **Bit 0 C1VDS:** Desactiva la medición VSOURCE para el canal 1 si tiene valor 1 lógico. Desactiva la medición si vale 0.

DIRECCIÓN 00H (0000-0000)							
BITS							
-	CDEN	MSKAL	C2IDS	C2VDS	TOUT	C1IDS	C1VDS
7	6	5	4	3	2	1	0
ACTIVO (VSENSE)							
0-X	0-X	0-X	0-X	0-X	0-X	0	1
ACTIVO (VSOURCE)							
0-X	0-X	0-X	0-X	0-X	0-X	1	0
ACTIVO (VSENSE+VSOURCE)							
0-X	0-X	0-X	0-X	0-X	0-X	0	0
STANDBY							
0-X	0-X	0-X	0-X	0-X	0-X	0	0

Tabla 2.6. Registro de configuración general

B. Registro 01h: Conversion Rate Register

- **Los bits 7 a 2** no están implementados
- **Los bits 1 a 0** determinan la tasa de conversión
  - **a.** 00: 1 muestra por segundo
  - **b.** 10: 2 muestra por segundo
  - **c.** 01: 4 muestras por segundo
  - **d.** 11: continua (por defecto)

C. Registro 02h: One Shot Register

Cuando el dispositivo está en estado de espera, escribir en el registro **02H (0000-0010)** iniciará un ciclo de conversión y actualizará todas las mediciones.

D. Registro 04h: Hight Limt Status

De este registro de sólo lectura, se utiliza el bit 7 (CVDN), el cual toma valor de 1 lógico cuando se termina una conversión y toma valor 0 durante la conversión o justamente después de haber leído este registro.

Conocer el estado de este bit resulta muy útil, como se verá más adelante, por dos motivos:

1. Los bits de configuración de modo del sensor no se pueden cambiar durante una conversión
2. Se puede programar la lectura de los registros de datos para que realice sólo cuando este bit está a nivel alto para asegurar que el registro se ha actualizado entre lecturas.

El resto de bits del registro o no están implementados o guardan relación con flags de valores límite de tensión y corriente y el pin de ALERT. Al no ser soportado por el protocolo I2C, no se utiliza.

E. Registro 0Ah: Vsource Sampling Configuration Register

- **Los bits 7 a 4** guardan relación con el canal 2 de medida y no se utilizan en el sensor PAC1710.

- **Los bits 3 a 2 (C1RS)** determinan el tiempo en la medición de VSOURCE del canal 1. **A menor tiempo, menor resolución (determinada por el número de bits), como se puede ver en la tabla xx.**
- **Los bits 1-0** (**C1RA)** controlan el promedio digital que se aplica a la medición VSOURCE del canal 1.

En la *Tabla 2.7.* se resumen las tablas 4-2 y 4-3 del datasheet [33]. Se muestran los tiempos de muestreo junto la resolución y los denominadores de las ecuaciones Ecuación 10)Ecuación 11) (DEN. A y DEN. B) asociados. También se muestran el número de muestras promediadas dependido del valor de los bits 1-0.

El signo (d) señala el valor por defecto de los bits del registro.

DIRECCIÓN 0AH (0000-1010)						
BITS						
C1RS: CHANNEL1 VOLTAGE SOURCE SAMPLING TIME SETTINGS					C1RA: CHANNEL 1 VOLTAGE SOURCE AVERAGING SETTINGS	
3		2			1	0
VALORES	TIEMPO (ms)	BITS	DEN. A	DEN. B	VALORES	MUESTRAS A PROMEDIAR
00	2.5	8	256	255	00 (d)	Desactivado
01	5	9	512	511	01	2
10 (d)	10	10	1024	1023	10	4
11	20	11	2048	2047	11	8

Tabla 2.7. Registro de configuración de Vsource. Bits 3-0

F. Registro 0Bh: Vsense Sampling Configuration Register

- **El bit 7** no está implementado
- **Los bits 6 a 4** (**C1CSS)** determinan el tiempo de muestreo de VSENSE del canal 1.
- **Los bits 3 y 2** (**C1SA)** controlan el promedio digital que se aplica a la medición VSENSE del canal 1.
- **Los bits 1 y 0 (C1SR)** determinan el rango a fondo de escala de VSENSE para el canal 1.

En la tabla 2.8 se resume la tabla 4-5 del datasheet [33]. Se muestran los tiempos de muestreo junto la resolución y el denominador de la *Ecuación 9* (DEN. X) asociados.

DIRECCIÓN 0BH (0000-1011)			
BITS			
C1SS			
6	5	4	
VALORES	**TIEMPO DE MUESTREO (ms)**	**BITS**	**DENOMINADOR C**
000	2.5	6 + signo	63
001	5	7 + signo	127
010	10	8 + signo	255
011	20	9 + signo	511
100	40	10 + signo	1023
101	80	11 + signo	2047
110	160	11 + signo	2047
111	320	11 + signo	2047

Tabla 2.8. Registro de configuración de Vsense. Bits 6-4

Por otro lado en la *Tabla 2.9* se resumen las tablas 4-4 y 4-6 del datasheet [33]. Se muestran el número de muestras promediadas dependido del valor de los bits 2-3 y el rango de voltajes según los bits 1-0.

El signo (d) señala el valor por defecto de los bits del registro.

DIRECCIÓN 0BH (0000-1011)			
BITS			
C1SA: CURRENT-SENSING SAMPLING TIME SETTINGS		C1SR: CHANNEL 1 VOLTAGE SOURCE AVERAGING SETTINGS	
3	2	1	0
VALORES	**MUESTRAS A PRMEDIAR**	**VALORES**	**RANGO DE FONDO DE ESCALA**
00 (d)	Desactivado	00	-10 mV a +10 mV
01	2	01	-20 mV a +20 mV
10	4	10	-40 mV a +40 mV
11	8	11 (d)	-80 mV a +80 mV

Tabla 2.9. Registro de configuración de Vsense. Bits 3-0

En este caso, la resolución es inversamente proporcional al tiempo de muestreo y al rango de fondo de escala, como se muestra en la siguiente imagen (*Tabla 2.10*) obtenida de la tabla 4-8 del datasheet [33].

Sampling Time	Resolution (±)			
	±10 mV	±20 mV	±40 mV	±80 mV
2.5 ms	156.3 µV	312.5 µV	625.0 µV	1.250 mV
5 ms	78.13 µV	156.3 µV	312.5 µV	625.0 µV
10 ms	39.06 µV	78.13 µV	156.3 µV	312.5 µV
20 ms	19.53 µV	39.06 µV	78.13 µV	156.3 µV
40 ms	9.76 µV	19.53 µV	39.06 µV	78.13 µV
≥ 80 ms	4.88 µV	9.76 µV	19.53 µV	39.06 µV

Tabla 2.10. Resolución dependiendo del rango y el tiempo de muestreo

G. Registros 0Dh y 0Eh: CH1 Sense Voltage Hight/ Low Byte

Estos registros guardan el valor en binario equivalente a la caída de tensión por Rsense. Los bits 3 a 0 del registro 0Eh no se utilizan. Dependiendo de la resolución configurada, se emplean entre 7 y 12 bits, según la *Tabla 2.8*.

H. Registros 11h y 12h: CH1 Vsource Voltage Hight/Low Byte

Estos registros guardan el valor en binario equivalente a la tensión de la fuente. Los bits 4 a 0 del registro 0Eh no se utilizan. Dependiendo de la resolución configurada, se emplean entre 8 y 11 bits, según la *Tabla 2.7*.

El dato se almacena en CA2 (complemento a 2) estándar. El valor de escala completa positivo es 7F_Fh y el negativo 80_0h, que se corresponderá al valor de FSR configurado.

El bit más significativo indica el signo, la dirección de la corriente. Si el bit de signo es '0', la corriente fluye a través de RSENSE desde el pin SENSE + a SENSE- (corriente positiva). Si es '1', la corriente fluye en sentido contrario y se considera negativa.

2.5. COMUNICACIÓN NODO SENSOR. HARDWARE: DISEÑO DE LA PCB PARA EL PAC1710

En este apartado se comentará brevemente el diseño de la placa para la medición con el sensor de PAC1710 y la comunicación I2C nodo-sensor. Este apartado se puede ver con más detalle en el **TFG de Óscar Escobar** [19].

2.5.1. PROGRAMA ORCAD PSPICE Y ORCAD LAYOUT

Para la elaboración de la PCB del sensor se utilizaron las herramientas:

- OrCAD PSPICE v15.7, para los esquemáticos y conexionados. En la *Figura 2.11*. se puede ver el esquema de conexiones de la placa:

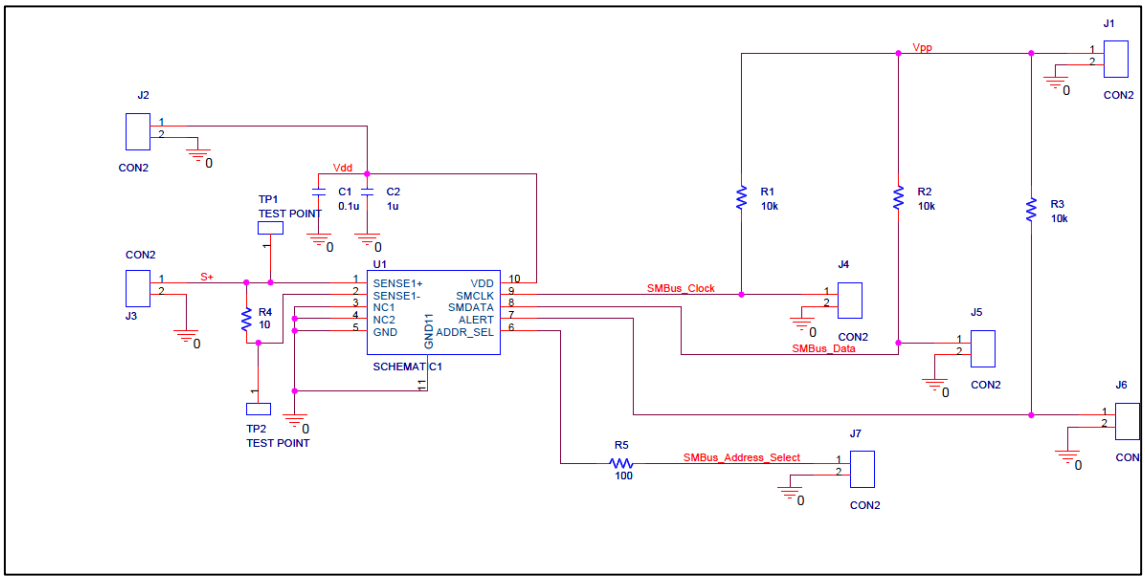

Figura 2.10 Esquemático de la PCB en OrCAD Pspice

- OrCAD Layout v15.7, para el diseño de la PCB y la huella del sensor. En la *Figura 2.12.* puede verse la capa Top de la PCB:

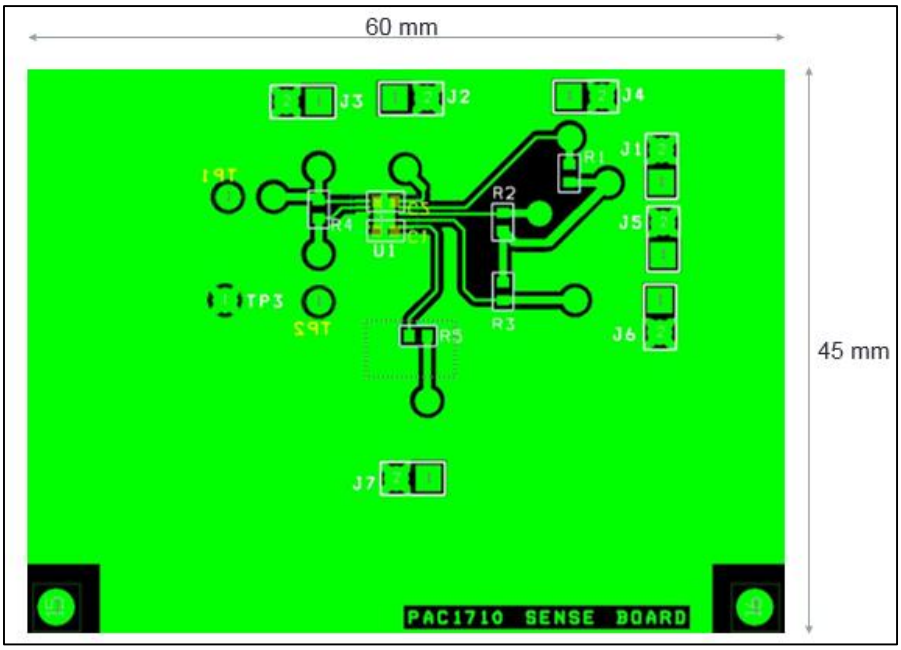

Figura 2.11 Capa Top de la PCB en OrCAD Layout

La licencia que se utilizó para estas herramientas fue proporcionada por la universidad.

2.5.2. BREVE DESCRIPCIÓN HW DEL SISTEMA PROPUESTO

La placa consta de:

- Vías desde los pines de alimentación del sensor a los jumpers (J2 y J1) para facilitar la conexión a la alimentación de 3V a 5.5V. Vdd alimenta al sensor, mientras que Vpp es la línea de referencia para las resistencias de Pull-Up de $10\ \mathrm{K\Omega}$ para la comunicación I2C (ver sección 2.3.5.a). Se incluyen dos condensadores de desacoplo (de valor total de 1,1uF) a la entrada del pin Vdd.
- Resistencia de SMBUS_ADDR de $100\ \Omega$ para determinar la dirección de esclavo para la comunicación I2C, conectada entre el pin ADDR_SEL y un jumper (J7) a masa.

Cortocircuitando los dos pines de J7, la dirección será 9Ah. En caso contrario, la dirección será 30h, la correspondiente a "circuito abierto". (Ver apartado 2.3.5.b.)

- Conectores hasta los pines SENSE1+ y SENSE1- para la alimentación del nodo. Entre estos dos pines se encuentra la resistencia de shunt Rsense. El valor de Rsense dependerá del rango de corrientes a medir. Con una resistencia de 2.8 Ω es posible medir gran parte de los consumos que se espera ver en el nodo. (Ver apartado 2.3)
- La referencia de tensiones consta de dos planos de masa, TOP y BOTTOM, unidos entre sí, a cada pin 2 de los jumpers y al pin 5 (GND) del sensor.

El diseño final de la placa es el siguiente (*Figura 2.13.*):

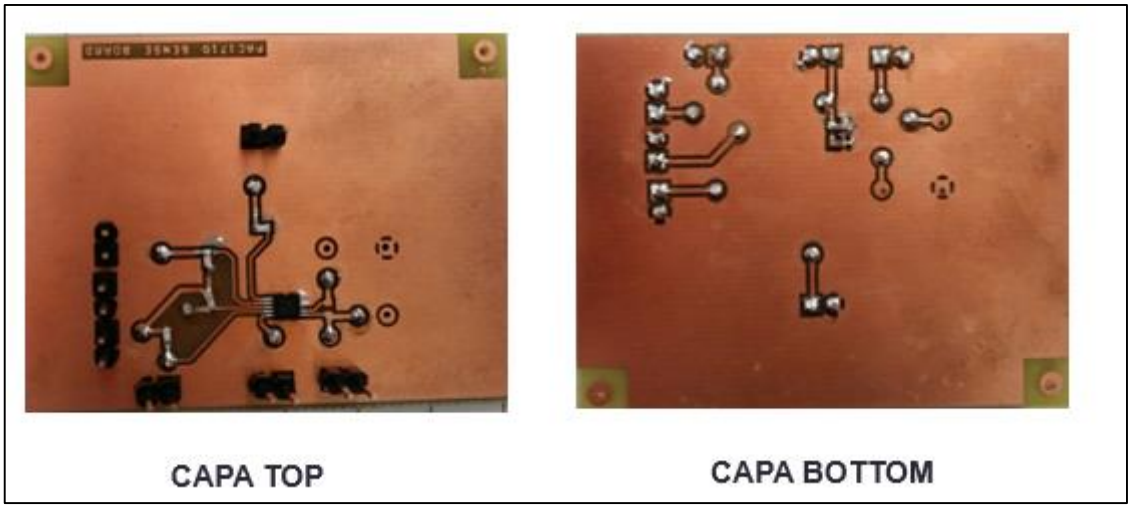

Figura 2.12 Diseño final de la PCB

2.6. PLACA DE EVALUACIÓN PAC1710 Y PAC1720 [36]

En este apartado se explicará el funcionamiento y características de la placa de evaluación PAC1710 y PAC1720 (**PAC1710 and PAC1720 High-Side Current Sensors Evaluation Board**) junto con la aplicación software gratuita Chip Manager Application.

Esta placa se añadió al proyecto tras experimentar ciertas complicaciones a la hora de implementar el sensor PAC1710 con la PCB diseñada. Ello ha proporcionado:

- Conocimiento más profundo del funcionamiento de la escritura y lectura de los registros del sensor.
- Obtener unos primeros resultados de consumo y establecer los rangos de consumo esperados y las resistencias "Rsunt" y configuraciones adecuadas para cada modo.
- Posibles mejoras de la PCB, tales como la inclusión de sistemas de protección **TVS** (*Transient-Voltage-Suppression*), presentes en la placa de evaluación.

2.6.1. DESCRIPCIÓN DEL FUNCIONAMIENTO Y CARACTERÍSTICAS DE LA PLACA

A continuación se pasará a explicar de forma resumida el funcionamiento de la placa de evaluación. De igual modo que en capítulos anteriores, la explicación del hardware se describe con más detalle en el **TFG de Óscar Escobar Muñoz** [19].

El "**PAC17X0 High-Side Current Sensors Evaluation Board (ADM00494)**" de Microchip consta de un sensor PAC1710 y un sensor PAC1720. En ambos casos es posible cambiar su configuración cambiando los registros con la aplicación Microchip Chip Manager (Ver apartado 2.6.2) o bien mediante una comunicación I2C/ SMBUS externa y una aplicación propia. También dispone de

jumpers (JX) y conmutadores (SWX) para el cambio manual de los distintos modos de medida, los cuales se pueden ver en la *Figura 2.14.* (ver Figura 3-2 del datasheet [36]):

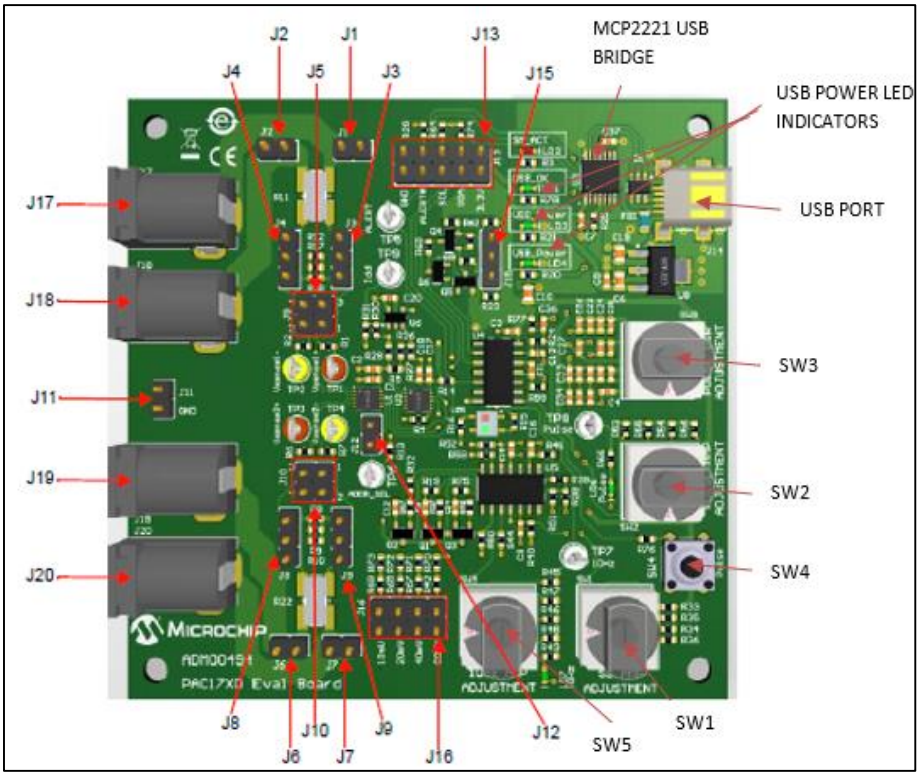

Figura 2.13 Componentes de la placa de evaluación del PAC17X0

El canal Vsense del PAC1710 integrado se usa para demostraciones de pulsos de corrientes y corrientes variables, a través de una fuente de corriente interna y amplificadores. Los parámetros de configuración, así como los pulsos de corriente, pueden modificarse por medio de los conmutadores (SWX) y cortocircuitando los jumpers diseñados para tal función.

Por otro lado, también dispone de un sensor PAC1720, cuyos dos canales de Vsense tienen la posibilidad de establecerse en "demo mode", en el cual se usa una fuente interna para realizar distintas demostraciones de medida, o el "sys mode", en el cual es posible medir con una fuente externa conectada a los jumpers J2-J1 y/o J6/J7[19].

Por último, la placa se alimenta mediante la conexión a un ordenador por USB. El MCP1703/3.3V transforma la tensión de 5V del ordenador a los 3.3V requeridos para la alimentación del sensor y la línea de SMBUS/I2C, mientras que el MCP2221 permite la comunicación SMBUS, I2C y UART a través del puerto USB (Ver *Figura 2.10.*). Para terminar, desconectando los pines de J13 del MCP2221 y conectándolos a un circuito externo es posible alimentar el sensor con 3.3V con una fuente secundaria, además de comunicar el sensor por un circuito alternativo a la transmisión de datos por USB.[20]

[19] La fuente se conectará en los jumpers J10/ J5 junto con una resistencia Rsense externa. Esto es debido a que las resistencias disponibles para este modo en ambos canales es de 3mΩ, un valor demasiado bajo para el rango de corrientes de la aplicación a medir.

[20] Como se verá en el apartado 3.3, para este modo de uso será necesario incluir al menos las resistencias de Pull-Up de las líneas SCL y SDA de SMBUS/ I2C.

2.6.2. MICROCHIP CHIP MANAGER

a. Descripción general

La aplicación "Chip Manager" permite:

- Mostrar en tiempo real el valor de cada registro y una descripción de las configuraciones según su valor
- Escritura y lectura de los registros por medio de la herramienta "Debug I/O"
- Test de escritura y lectura SMBUS/I2C completo y automático
- Visualización gráfica de los valores de Vsense y Vsource con rango de ejes seleccionable
- Grabación de los datos de simulación con opciones de parada y puesta en marcha y exportación a fichero de texto.

b. Primeros pasos

En primer lugar es necesario instalar los drivers para el control de la tarjeta y del MCP2221 ("MCP2200/MCP2221 Windows Driver & Installer") los cuales se pueden obtener en la página del fabricante:

http://www.microchip.com/wwwproducts/Devices.aspx?product=MCP2221

Una vez hecho esto, se procede a la descarga de "ChipMan v4.16.7" desde www.microchip.com y posteriormente a su instalación con el ejecutable ChipMan-v4.16.7-windows-installer.exe de la carpeta zip descargada. Tanto los drivers como el programa son completamente gratuitos.

El siguiente paso es conectar la tarjeta al ordenador con el cable USB-Mini USB incluido en el kit. Tras comprobar que los leds de comprobación de Power y USB se iluminan (ver *Figura 2.13*), se procede a ejecutar el programa.

Como último paso, al iniciar el programa por primera vez, es posible que sea necesario seleccionar el dispositivo que se está utilizando. Para ello, desde el menú principal, seleccionar **Options → Select Device**. En la lista de dispositivos, seleccionar el PAC17x0. Finalmente, en Master Controller seleccionar USB SMBus Bridge y en Slave Addresses escribir la dirección 98h. (Ver *Figura 2.15.*)[21]

[21] La dirección de esclavo depende del jumper J12 (ver esquemático A.2. Board del Datasheet [33]). Si está cerrado (por defecto), la resistencia equivalente de la conexión del pin addr_select a masa da una dirección de esclavo de 98h. Si está abierto, se conectará a la resistencia R13 (100Ω), correspondiente a una dirección de esclavo de 9Ah (ver *Tabla 2.2*.)

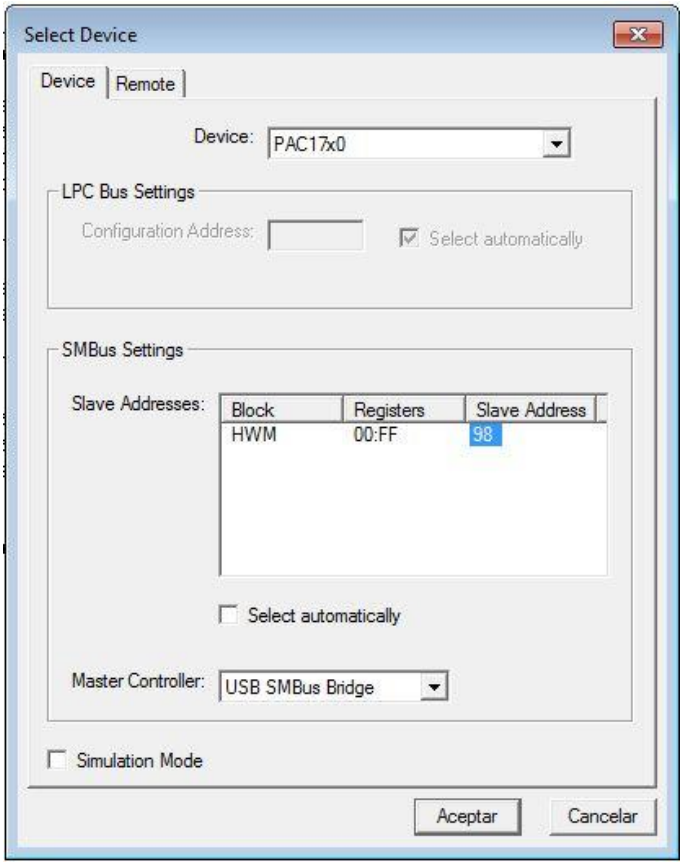

Figura 2.14 Microchip Chip Manager. Selección del dispositivo

c. Uso de la herramienta

En el siguiente apartado se explicará cómo hacer uso de cada una de las utilidades de la herramienta que se han empleado en el proyecto.

- Visualización de registros en tiempo real
- Escritura y lectura de registros en la ventana "Debug I/O Window"
- Visualización gráfica de los registros y exportación a un fichero de texto

i. Visualización de registros en tiempo real

En el lado izquierdo aparecerá una lista de los distintos registros organizados según su función (*Figura 2.16.*).[22] Seleccionando alguna de las ramas del esquema, aparecerá en la ventana derecha el conjunto de registros, indicando su dirección, el tipo (lectura/ escritura), su último valor actualizado, las unidades en las que se muestra, su abreviatura y el tipo de bus (*Figura 2.17.*).

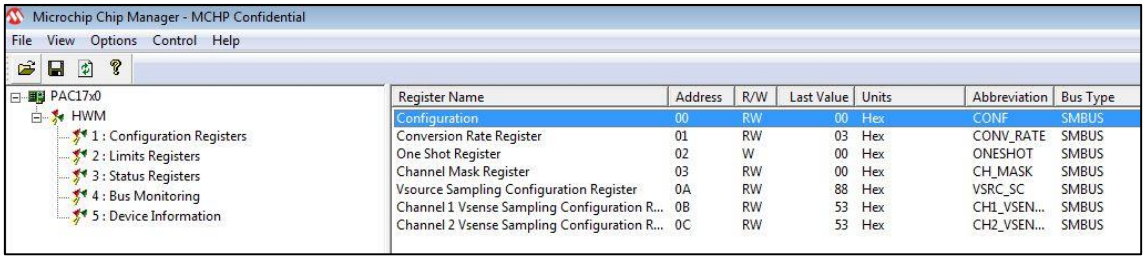

Figura 2.15 Microchip Chip Manager. Visualización de registros

[22] Para más información sobre los registros, consultar el apartado "2.3.6. Registros" de la memoria.

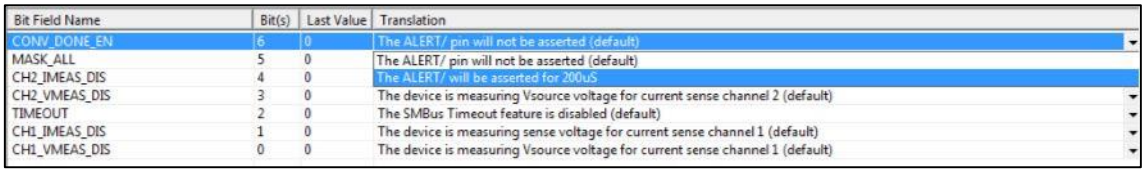

Figura 2.16 Microchip Chip Manager. Descripción de la configuración según el valor de los bits

ii. Escritura y lectura de registros en la ventana Debug I/O Window

Resulta muy útil para conocer el comportamiento de sensor en este protocolo de comunicación y tenerlo en cuenta a la hora de realizar los drivers de comunicación por I2C con el nodo.

Figura 2.17 Microchip Chip Manager. Debug I/O window

A esta función se accede en View→Debug I/O Window. Una vez en la ventana:

- Escribiendo la dirección de esclavo adecuada (ver apartado "a. Primeros Pasos", nota al pie) en "**SMBus Slave Address**" es posible escribir y leer los registros por SMBUS.
- En "**Offset Address**" escribir la dirección que se desee leer o escribir
- En "Data", escribir el valor hexadecimal del registro al que se desea cambiar
- Por último, seleccionar "**Read**" o "**Write**" y visualizar en el lado derecho de la ventana si la acción de lectura (-i) o escritura (-o), respectivamente, se ha realizado con éxito. (Ver *Figura 2.18.*)

iii. Visualización gráfica de los registros y exportación a un fichero de texto

Aunque está disponible para cualquier registro, esta opción es especialmente útil para visualizar los datos de la tensión Vsense, Vsource y la potencia en una gráfica, para su posterior importación a haciendo click derecho en el nombre de cualquiera de los registros, aparece la opción "**Add Registers(s) to Plot**". Se abrirá una pestaña adicional con una gráfica bidimensional, en el que se puede modificar el rango de los ejes (x: tiempo, y: valor del registro) y el tiempo por división (de 500 segundos por división a 100 ms), a fin de optimizar la visualización de la información.

En **Options → configure** se pueden encontrar todas las opciones de personalización, tales como el color de la línea graficada, del fondo, el tamaño del buffer de datos o los rangos.

Para comenzar la captura de datos, hay que seleccionar **Control → Start**. Así mismo, se pausar en **Control → Pause** y volver a retomar con **Control → Resume**. Cuando se acabe de tomar datos, se debe parar con **Control → Stop**.

Una vez parada la captura, es posible exportar los datos a un fichero de texto con **File → export**.

Se creará un fichero de texto con todos los datos adquiridos desde que se pulsó "**Start**" hasta "**Stop**" con información sobre el/ los registros seleccionados y el tiempo. Un ejemplo de ello se puede ver en la *Figura 2.25*.

2.7. COMUNICACIÓN NODO SENSOR. SOFTWARE: DISEÑO DE LOS DRIVERS DE COMUNICACIÓN

En este apartado se explicará paso a paso el desarrollo de los drivers para la comunicación I2C del PAC1710 con el nodo ATMEGA256rfr2 Xplained Pro.

En primer lugar se hablará del entorno de desarrollo utilizado para ello, Atmel Studio.

2.7.1. ATMEL STUDIO [27]

Para el desarrollo del código en C, su depuración, testeado y cargado en la placa de ATMEGA256rfr2 Xplained Pro se ha utilizado Atmel Studio 7.0.

Atmel Studio es un entorno de desarrollo integrado (IDE) para escribir y depurar aplicaciones para microcontroladores basados AVR y ARM en Windows. Proporciona un editor de código y simulador para C/ C++/ ensamblador, herramientas de depuración y programación en el chip.

Por otro lado tiene la capacidad de detectar el microcontrolador una vez está conectado por puerto USB al ordenador, permitiendo configurarlo de forma sencilla, así como la visualización del cambio de los registros durante la ejecución del programa.

A continuación se resumen las herramientas más utilizadas del entorno durante el proyecto. Para conocer con más detalle el uso de esta herramienta, consultar el manual de usuario [27] o a través de la pestaña "help" del programa.

- **ASF:** en esta pestaña se puede configurar una vez se haya detectado el microcontrolador utilizado, la aplicaciones existentes dentro de las librerías de Atmel que resulten útiles para el proyecto.
- **Build:** contiene todo lo relativo a la compilación del proyecto y su configuración. Una vez compilado, proporciona información sobre los errores de código (motivo del error, localización y sugerencia de resolución), así como advertencias o "warnings". Si bien es posible cargar el código en el nodo con "warnings", no es recomendable, ya que su funcionamiento puede ser errático e inesperado.
- **Debug:** para la depuración del código, programación del nodo, control de puntos de parada. Permite ejecutar paso a paso el programa para comprobar correcto funcionamiento de la aplicación. Con QuickWatch y AddWatch, además, es posible ver el estado de las variables, funciones y estructuras seleccionadas durante la pausa, siempre que continúen definidas en el momento de parada. Por último, con "I/O View" es posible visualizar los registros durante el modo debug, mostrándose de color rojo aquéllos bits que han cambiado de valor.
- **Device Programming:** se encuentra dentro de la pestaña "**tools**". Se utiliza para programar las distintas memorias del dispositivo, "fuses", bloqueos, borrar las memorias o escribir firmas de usuario. También permite seleccionar los generadores de voltaje y el reloj utilizado.
- Entre otras utilidades disponibles se encuentra el "**Solution Explorer**", para poder ver de forma rápida la estructura del proyecto organizado en capetas, los archivos de cabecera y

fuente que lo componen y las funciones, estructuras y definiciones existentes dentro de éstos.

Una vez descrito el entorno, se describirá el proceso de diseño de los drivers, hablando en primer lugar del código de referencia del cual se partió, en el siguiente apartado.

2.7.2. PROGRAMA/ CÓDIGO BASE

Para la elaboración de este trabajo se partió de un proyecto ya proporcionado y estructurado por José Ángel Miranda Calero [38].

En cuanto al diseño de los drivers se ha seguido como modelo los drivers para el sensor bme280 empleado en el trabajo de Ismael Granados Llorente "Desarrollo e implementación de drivers para aplicación IoT. Diseño y desarrollo de red de sensores para seguridad basada en IEEE802.15.4" [39]. Dichos drivers disponían de funciones básicas de comunicación I2C que aunque estaban adaptadas para dicho sensor han servido de guía útil para crear las de este proyecto.

2.7.3. DISEÑO DE LOS DRIVERS

Los drivers para el PAC1710 se han diseñado de acuerdo a los siguientes criterios, algunos de los cuales ya se tuvieron en cuenta en [39]:

- **Optimizado**: realizar las funciones útiles para la aplicación evitando repeticiones innecesarias y manteniendo el sensor lo máximo posible en el estado de baja energía.
- **Fácil comprensión**: elaborar el código de forma que sea lo más sencillo posible a la vez que cumple con todas las especificaciones propuestas. Comentarlo de forma adecuada para que otros usuarios puedan utilizarlo de forma rápida e intuitiva.
- **Flexible**: en consonancia con el punto anterior, tratar que el código de los drivers pueda ser reutilizado en un futuro en aplicaciones similares o en otros sistemas.
- **Control y detección de errores**: diseñar el código para que sea posible encontrar posibles errores funcionales y anticiparse a posibles problemas. Por ejemplo, usar funciones que devuelvan un determinado valor de estado dependiendo del éxito de la operación, error debido a una incorrecta inicialización, error de comunicación, etc.
- **En inglés**: ya que es universalmente usado

Por otro lado, se han tratado de seguir las normas para la escritura en C (ver Anexo: reglas de programación en C).

Los drivers están formados por dos archivos.

El archivo de cabecera PAC1710.h contiene la estructura principal (struct), las directivas (#define) de los valores de uso común y las declaraciones de las funciones. Por otro lado, el archivo fuente PAC1710.c contiene la definición de todas las funciones.

a. Estructura

La estructura contiene variables que guardan tanto la configuración de los registros como los valores finales de tensión y corriente tras haber sido calculados. También tiene dos punteros a las funciones de lectura y escritura I2C.

Al ser todos los registros de 8 bits, las variables asociados son enteros sin signo (uint8_t), mientras que las variables relacionadas con los niveles de tensión y corriente son de 16 bits. Esto es así debido a que como se mencionó en el apartado 2.4.6. , esta información se almacena en

dos registros de 8 bits, pudiendo tomar la corriente un valor negativo. Se puede ver a continuación (*Figura 2.20.*).

```
struct PAC1710_T {
    //Values of configuration
    uint8_t mode;           //Mode data
    uint8_t conv_rate;      //Conversion rate data
    uint8_t conv_done;      //Conversion done bit, 0 while conv_rate is running
    uint8_t vsource_avr;    //Vsource average data
    uint8_t vsource_stime;  //Vsource sample time data
    uint8_t vsense_stime;   //Vsense sample time data
    uint8_t vsense_avr;     //Vsense average data
    uint8_t vsense_rang;    //Vsense range data
    uint8_t chip_id;        //chip id of the sensor
    uint8_t dev_addr;       //device address of the sensor
    uint8_t config_reg;     //status of configuration register
    uint8_t flag;           //Flag status sensor

/*Voltage and current data (of registers)*/
    uint16_t voltage_vsource;   /* voltage over sampling*/
    int16_t current_vsense;     /* current over sampling*/
    uint16_t vsource_stime_10ms;/* vsource sample time in ms*10*/
    uint16_t vsense_stime_10ms;/* vsense sample time in ms*10*/
```

Figura 2.18. Principales variables de la estructura PAC1710_T

a. *Directivas*

#define GET Y SET REG_BITS VALUE. Sirven respectivamente para obtener o modificar los bits determinados del registro seleccionado (*Figura.2.20.***)**

```
//PAC1710_GET_REG_BITS_VALUE
//Get only the bits determined by bitname##__POS and bitname##__MSK
//of the register determined by regvar
#define PAC1710_GET_REG_BITS_VALUE(regvar, bitname)
\((regvar & bitname##__MSK) >> bitname##__POS)

//PAC1710_SET_REG_BITS_VALUE
//Set only the bits determined by bitname##__POS and bitname##__MSK
//to val of the register determined by regvar
#define PAC1710_SET_REG_BITS_VALUE(regvar, bitname, val)
\((regvar & ~bitname##__MSK) | ((val<<bitname##__POS)&bitname##__MSK))
```

Figura.2.19. Macros de manipulación de bits

Dichas funciones se encontraban ya implementadas en el proyecto base y se han reutilizado para la escritura y lectura de registros.

Directivas #define para la configuración de los distintos modos

Son el valor en decimal que corresponde al equivalente en binario de la parte del registro que modifica el modo en concreto. Se muestra como ejemplo las definiciones para el cambio de modo y los distintos valores de conversión.

Directivas #define para tamaños de los registros de resultados, valores nulos y de error y valores de desplazamiento de bits (*Figura 2.22.***)**

```
/* Constants */
#define PAC1710_NULL (0)
#define PAC1710_V_I_DATA_SIZE (1)
#define PAC1710_V_I_DATA_N_BITS (16)
#define PAC1710_RESULTS_DATA_LENGHT (2)
#define PAC1710_INIT_VALUE (0)

/* shift definitions*/
#define PAC1710_SHIFT_BIT_POSITION_BY_02_BITS   (2)
#define PAC1710_SHIFT_BIT_POSITION_BY_04_BITS   (4)
#define PAC1710_SHIFT_BIT_POSITION_BY_06_BITS   (6)
#define PAC1710_WR_DATA_LENGTH  (1)

/*************************************************/
/**\name    ERROR CODE DEFINITIONS  */
/*************************************************/
#define C_SUCCESS   ((uint8_t)1)
#define E_PAC1710_NULL_PTR  ((int8_t)-127)
#define E_PAC1710_COMM_RES  ((int8_t)-1)
#define E_PAC1710_OUT_OF_RANGE   ((int8_t)-2)
#define ERROR_LIMIT 100
#define ERROR   ((int8_t)-1)
```

Figura 2.20. Directivas para tamaños de registros, desplazamientos de bits, valores nulos y de error

Directivas #define relativas a la comunicación I2C (*Figura 2.23.***)**

Como la dirección de esclavo típica (DEV_ADDR) o el tamaño en bits del registro (I2C_BUFFER_LEN). Según el datasheet del sensor PAC1710, la dirección de esclavo, determinada por SMBUS_ADDR (Ver Tabla 2.2), difiere en el bit menos significativo a la hora de realizar la lectura (nivel 1 lógico) o la escritura (nivel 0 lógico).

```
/*************************************************/
/**\name    I2C ADDRESS DEFINITIONS  */
/*************************************************/
#define I2C_BUFFER_LEN  8
#define PAC1710_DEV_ADDR (0x98)
#define PAC1710_SMBUS_I2C_ADDR  (0x4B) //7 BITS
#define PAC1710_ADDR_SEL_W  (PAC1710_DEV_ADDR)  // LSB 0->W, LSB 1->R
#define PAC1710_ADDR_SEL_R  (PAC1710_DEV_ADDR+1)  // LSB 0->W, LSB 1->R
```

Figura 2.21. Directivas *para la comunicación I2C*

Directivas #define para cada registro del PAC1710 utilizado (*Figura 2.24.***)**

```
/*************************************************/
/**\name    REGISTER ADDRESS DEFINITIONS  */
/*************************************************/
#define PAC1710_PRODUCT_ID_REG  (0xFD)
    #define PAC1710_PRODUCT_ID_VAL  (0x57)  //(0x58)
#define MANUFACTURER_ID_REG     (0xFE)
    #define MANUFACTURER_ID_VAL     (0x5D)
#define REVISION_REG            (0xFF)
    #define REVISION_REG_VAL        (0x81)
//-------------------------------------------------
#define PAC1710_MODE_ADDR       (0x00)
#define PAC1710_CONV_RATE_ADDR  (0x01)
#define PAC1710_ONE_SHOT_ADDR   (0x02)
#define PAC1710_MASK_ADDR       (0x03)
#define PAC1710_HLS_ADDR        (0x04)
#define PAC1710_LLS_ADDR        (0x05)
#define PAC1710_VSOURCE_ADDR    (0x0A)
#define PAC1710_VSENSE_ADDR     (0x0B)
#define PAC1710_VSENSE_HB_ADDR  (0x0D)
#define PAC1710_VSENSE_LB_ADDR  (0x0E)
#define PAC1710_VSOURCE_HB_ADDR (0x11)
#define PAC1710_VSOURCE_LB_ADDR (0x12)
#define PAC1710_POWER_HB_ADDR   (0x15)
#define PAC1710_POWER_LB_ADDR   (0x16)
```

Figura 2.22. Directivas *para cada registro del PAC1710 utilizado*

Directivas #define para la manipulación de bits y registros

```
/*************************************************************/
/**\name    BIT MASK, LENGTH AND POSITION DEFINITIONS  */
/*************************************************************/
/* MODE */
//MASK BIT
#define PAC1710_MASKBIT__LEN (8)
#define PAC1710_MASKBIT__POS (5)
#define PAC1710_MASKBIT__MSK (0x20)
#define PAC1710_MASKBIT__REG (PAC1710_MODE_ADDR)

//STANDBY
#define PAC1710_MODE__LEN (8)
#define PAC1710_MODE__POS (0)
#define PAC1710_MODE__MSK (0x03)
#define PAC1710_MODE__REG (PAC1710_MODE_ADDR)

//V MODE
#define PAC1710_V_PIN__LEN (8)
#define PAC1710_V_PIN__POS (0)
#define PAC1710_V_PIN__MSK (0x01)
#define PAC1710_V_PIN__REG (PAC1710_MODE_ADDR)
```

Figura 2.23. Directivas #define para la manipulación de bits y registros

c. Funciones

El conjunto de funciones que constituyen el driver se pueden ver a continuación:

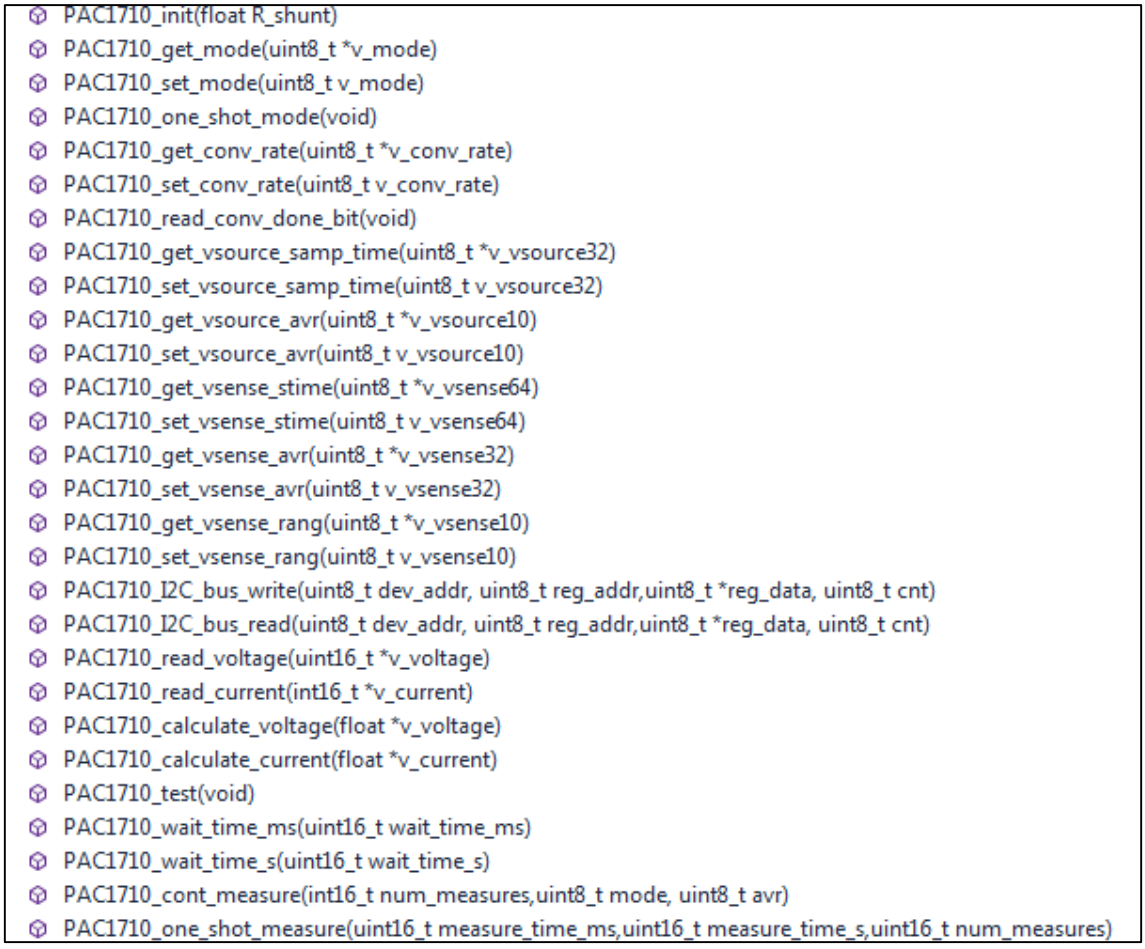

Figura 2.24. Principales funciones del driver PAC1710

Dichas funciones se pueden dividir del siguiente modo:

- Funciones de escritura y lectura I2C
- Funciones de establecimiento de registro y lectura de los registros de configuración
- Funciones de lectura de los registros de Vsource, Vsense y Pratio
- Función de inicialización
- Funciones de cálculo de Corriente, Tensión y Potencia
- Funciones de toma de datos
- Función de test
- Funciones de tiempo

A continuación se explicará cada una de manera más detallada.

El código en C de todas las funciones puede verse detallado y comentado en el anexo (A1.1.)

- **Funciones de lectura y escritura I2C:**

Ambas toman como parámetros la dirección de esclavo, la dirección del registro a escribir/leer, un puntero a la dirección del array que almacenará el dato a escribir/leído y el número de bytes del registro.

El valor que devuelven dependerá si se ha realizado la comunicación con éxito (devolviendo 1) o se ha producido algún error (devolviendo 0)

int8_t PAC1710_I2C_bus_write(*uint8_t* dev_addr, *uint8_t* reg_addr,*uint8_t* *reg_data, *uint8_t* cnt):

Internamente realiza lo siguiente:

- Crea una estructura con toda la información necesaria para la comunicación: bytes del registro, dirección de esclavo y del registro, número de registros a escribir y un array con el valor a escribir en el registro.
- Llama a la función HAL_TwiMaster_Write del archivo halTwi[23]. Dicha función toma como parámetro la estructura anterior y retorna un valor según los distintos resultados de la comunicación, distinguiendo entre distintos errores, lo que facilita la optimización y la depuración del código.
- Si la comunicación ha tenido éxito, devuelve 1.

Int8_t PAC1710_I2C_bus_read(*uint8_t* dev_addr, *uint8_t* reg_addr,*uint8_t* *reg_data, *uint8_t* cnt):

Según el protocolo de comunicación I2C, para leer en un registro, primero es necesario realizer la acción de escritura. (Ver apartado 392.4.5). Por ello los pasos seguidos en esta función son:

- Primero se crea una estructura de igual forma que con la función de escritura, con la diferencia en que no hay un valor a escribir. Tras ello se llama a la función HAL_TwiMaster_Write y espera el valor 1 de retorno, que se identifica como una comunicación exitosa.
- Después, se crea otra estructura con la información necesaria para la lectura, diferenciándose principalmente en la anterior en que no hay valor a escribir y en su lugar hay un puntero a la dirección del array en la que se almacenará el dato del registro.
- Seguidamente, se llama a la función HAL_TwiMaster_Read del archivo halTwi. Dentro de esta función, se modifica directamente la dirección de esclavo sumándole 1, ya que el último bit de esta dirección debe ser 1 si se desea leer y 0 en el caso de escribir (ver apartado 2.4.5.c. y 2.4.5.d).
- Si la comunicación ha tenido éxito, se leerá el registro, se almacenará en la variable asignada y la función retornará el valor de 1.

➢ **Funciones de establecimiento de registros y lectura de los registros de configuración**

Las funciones con el nombre "get" almacenan el valor de configuración determinado tanto en la variable de la estructura definida en PAC1710.h reservada para ello como en la variable que toma como parámetro.

Las funciones con el nombre "set" sobrescriben los bits necesarios del registro de configuración de cada caso y almacenan el nuevo valor de configuración en la variable de la estructura reservada para ello. Son las siguientes:

int8_t PAC1710_get_mode(*uint8_t* *v_mode);
int8_t PAC1710_set_mode(*uint8_t* v_mode);
int8_t PAC1710_one_shot_mode(void);
int8_t PAC1710_get_conv_rate(*uint8_t* *v_conv_rate);
int8_t PAC1710_set_conv_rate(*uint8_t* v_conv_rate);
int8_t PAC1710_get_vsource_samp_time(*uint8_t* *v_vsource32);
int8_t PAC1710_set_vsource_samp_time(*uint8_t* v_vsource32);
int8_t PAC1710_get_vsource_avr(*uint8_t* *v_vsource10);

[23] Ésta y otras funciones pertenecen al paquete añadido con ASF wizard, de las cuales se partió para el desarrollo de la aplicación.

```
int8_t PAC1710_set_vsource_avr(uint8_t v_vsource10);
int8_t PAC1710_get_vsense_stime(uint8_t *v_vsense64);
int8_t PAC1710_set_vsense_stime(uint8_t v_vsense64);
int8_t PAC1710_get_vsense_avr(uint8_t *v_vsense32);
int8_t PAC1710_set_vsense_avr(uint8_t v_vsense32);
```

Excepto las tres primeras funciones, que guardan relación con los modos del sensor, el resto son de configuración. Como ejemplo de estas últimas, se explican a continuación dos de ellas. El resto de las funciones actúa de forma similar:

int8_t PAC1710_get_vsense_rang(*uint8_t* *v_vsense10):

- Se hace una llamada a la función PAC1710_I2C_bus_read para leer el registro y almacenar su valor en la variable v_vsense10.
- Conociendo el valor anterior del registro, se llama a la función PAC1710_GET_REG_BITS_VALUE, que toma sólo el valor de aquellos bits relacionados con la configuración a modificar y los guarda en las variables designadas.

int8_t PAC1710_set_vsense_rang(*uint8_t* v_vsense10):

- Se hace una llamada a la función PAC1710_I2C_bus_read para leer el registro y almacenar su valor en la variable v_vsense10.
- Conociendo el valor anterior del registro, se llama a la función PAC1710_SET_REG_BITS_VALUE, que devuelve el valor que debe tomar el registro teniendo en cuenta su valor anterior y los bits que deben cambiar para la configuración deseada.
- Con este nuevo dato, se llama a la función PAC1710_I2C_bus_write para sobrescribir el registro

Por último, en la función PAC1710_one_shot_mode se realiza la acción de escritura del registro 02h del sensor para realizar una toma de medidas, siempre y cuando el sensor se encuentre en modo STANDBY (Ver apartado 2.3.6.Registros). Su valor siempre es 00h, ya que se sobrescribe. Escribir en este registro tan sólo activa el modo ONE SHOT del sensor.

- **Funciones de lectura de los registros de Vsource y Vsense**

Son aquellas con las que se leen los respectivos registros de Vsense y Vosurce para obtener los valores relativos de tensión de la fuente y caída de tensión en la resistencia de shunt.

```
int8_t PAC1710_read_voltage(uint16_t *v_voltage);
int8_t PAC1710_read_current(int16_t *v_current);
```

Estas funciones llaman a la función PAC1710_I2C_bus_read, con la diferencia de que deben leerse dos registros consecutivos. Por ello, el array en el que se almacena el dato, es de 16 bits.

- **Funciones de cálculo de Corriente, Tensión y Potencia**

Las operaciones que se realizan son las mismas que se indican en el datasheet del sensor PAC1710 [33] (Ver apartado 2.3.3. y 2.3.4.). También se ha tenido en cuenta que hay cierto número de bits de los registros “Low_byte” de Vsense y Vsource que no se utiliza, debiendo desplazar un número determinado de bits para obtener el valor real en decimal del registro.

La función para el cálculo del valor real de corriente, tensión y potencia es la siguiente:

```
int8_t PAC1710_calculate_current(uint16_t *v_current,uint16_t rsense);
int8_t PAC1710_calculate_voltage(uint16_t *v_voltage);
```

Donde los dos primeros parámetros son un puntero a la dirección del array donde se almacenará el dato y rsense es el valor de impedancia de shunt utilizado.

- **Función de comprobación de fin de conversión**

```
int8_t PAC1710_read_conv_done_bit(void);
```

Lee el bit 7 del registro "*Hight Limit Status*" para comprobar el estado del ciclo de conversión. Si tiene valor lógico 1, se ha realizado el ciclo, mientras que si vale 0, se está realizando la conversión. Una vez leído el bit, éste vuelve a tener valor 0.

- **Función de inicialización**

```
int8_t PAC1710_init(float R_shunt);
```

En esta función se inicializa el sensor para una configuración determinada, dependiendo de la resistencia de shunt y el rango de corrientes a medir. Se ha definido las configuraciones para los valores de resistencia utilizados, 2,8 y 10 ohmios, además de una configuración por defecto para cualquier otro valor.

Además, la función devuelve un valor según se haya realizado todo el procedimiento con éxito o se haya producido algún error, existiendo un valor de error según el registro que no se haya escrito correctamente. Ello se comprueba almacenando en una variable el valor que se desea escribir y comparándolo con el valor que retorna la función de lectura tras haber sido sobrescrito el registro.

El procedimiento paso a paso es el siguiente:

- Se define la dirección de esclavo
- Se asigna el puntero a la estructura
- Se leen los registros PRODUCT_ID_REG y PRODUCT_ID_VAL para comprobar el correcto funcionamiento de la comunicación I2c
- Una vez comprobado que no hay problemas de comunicación, se procede a configurar los distintos registros:
    - Primero se establece el modo del sensor en STANDBY, de forma que sea posible cambiar posteriormente en ciclo de conversión y consuma lo mínimo posible hasta que se comience a medir
    - El ciclo de conversión se mantiene en continuo por defecto. Se comprobó con la herramienta Chip Manager que si se establece otro tipo de conversión, es producen errores al intentar sobrescribir el registro de modo 00h (Ver 2.6.2)
    - Se establece el tiempo de muestre de Vsource en 10 ms, medida sin realizar medias tanto para Vsource como para Vsense.
    - Por último, se configura la medida de la caída de tensión por la resistencia dependiendo de su valor. Para ello, se toma el dato de resistencia y se multiplica por el mayor valor de corriente que se prevee medir (en torno a 14 mV), estableciendo según este valor el mínimo rango posible para mejorar la resolución.

- **Funciones de toma de datos**

```
int8_t PAC1710_cont_measure(int16_t num_measures,uint8_t mode, uint8_t avr);
```

```
int8_t PAC1710_one_shot_measure(uint16_t measure_time_ms,uint16_t
measure_time_s,uint16_t num_measures);
```

Son las funciones que se utilizan con para la adquisición de datos. La primera realiza una medida continua según un número de medidas determinado para el modo seleccionado. Es decir, medirá corriente, tensión o ambas a la vez.

La segunda, toma medidas de corriente y voltaje estableciendo el tiempo entre ellas con las entradas de tiempo en ms y en segundos.

El siguiente paso es calcular los valores reales según los bits leídos en los registros Vsense y Vsource. Se llaman a las funciones cálculo de tensión y corriente.

Una vez hecho esto, se separa el valor tipo *float* en dos variables, una para la parte entera y otra para la parte decimal, que permita ser enviado por puerto serie.

En ambos casos, tras ello se procede al envío por puerto serie con el siguiente formato:

Tiempo1(entero),Tiempo1(decimal); Medida1(entero),Medida1(decimal)

Tiempo2(entero),Tiempo2(decimal); Medida2(entero),Medida2(decimal)

Esto permite guardar la información en un fichero de texto que pueda ser leído como matriz de datos en Matlab y finalmente sean representados.

- **Función de test**

Tiene la finalidad de comprobar de forma rápida y sencilla que la lectura y escritura de todos los registro funciona según lo esperado. No ha sido diseñada para el usuario, sino que tiene una finalidad más práctica para la comprensión y el testeo del código y del sensor.

void PAC1710_test(void):

- Primero se comprueba que la lectura de los registro de ID del sensor (FDh, FEh y FFh) da como resultado el valor indicado en el datasheet, comprobando así que la comunicación I2C es correcta.
- Para cada pareja de funciones set y get, se hace un bucle for (for(i=0;i<='MAX'24;i++)) en el cual, en cada iteración:
    - Se llama a cada función set creada, que sobrescribe con el valor i decimal los bits relacionados de registro determinado.
    - Se llama a la función get homónima, leyendo el registro y almacenando el valor de los bits en la posición del array i.
    - Si todo ha ido correctamente, el valor almacenado en cada posición del array debe coincidir con el valor del índice del array[25] y se enviará por puerto serie un mensaje de éxito. En caso contrario se enviará un mensaje de error.

- **Funciones de tiempo**

```
void PAC1710_wait_time_ms(uint16_t wait_time_ms);
void PAC1710_wait_time_s(uint16_t wait_time_s);
```

[24] MAX es el valor máximo en binario equivalente al nivel alto de todos los bits relacionados con la configuración. Por ejemplo, los bits de modo, (bits 0 y 1 del registro 00h) valdrán 00 (medida Vsense y Vsource), 01, (medir Vsense) 10 (medir Vsource) ó 11 (STANDBY). El máximo valor corresponde a 11 binario (3 en decimal), por lo que MAX en ese caso será 3.

[25] Si i=0, se estará escribiendo el valor 0 (00 binario en el caso de los bits de modo), que al hacer el get se guardará en la posición i=0 del array.

Se utiliza para asignar el tiempo entre medidas respectivamente en milisegundos y en segundos.

Éstas eran las principales funciones utilizadas en los drivers. No obstante, en el anexo se encuentra el código completo comentado. Una vez vistos los drivers, pasamos a explicar el procedimiento de adquisición de datos.

2.8. DISEÑO DEL PROCEDIMIENTO PARA LA ADQUISICIÓN Y EL ANÁLISIS DE DATOS

2.8.1. ADQUISICIÓN DE DATOS CON EL PROGRAMA CHIP MANAGER DE MICROCHIP

En un primer momento los datos se guardaban en un fichero de texto tras la medición con la placa de evaluación, con el procedimiento explicado en el apartado 2.5.2.c.iii.

El archivo de texto generado por la aplicación se muestra a continuación, en la *Figura 2.25*.

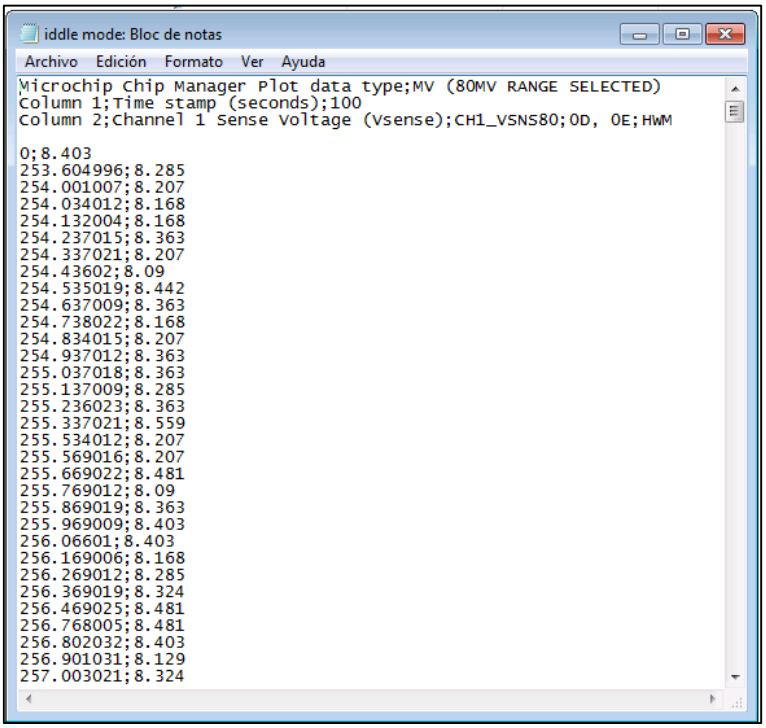

Figura 2.25. Archivo de texto generado con Chip Manager

Como se puede observar en la imagen, el fichero de texto consta de tres líneas iniciales con toda la información sobre el tipo de medida y la configuración, seguido de los datos representados en columnas divididas por punto y coma.

Los valores temporales presentan un offset a partir del segundo dato tomado debido al momento en el que se realiza la muestra.

Inicialmente para la representación se siguieron los siguientes pasos:

- Se guardaron los datos en un archivo de Excel, eliminando las líneas de cabecera del fichero de texto.
- Se diseñó un script de Matlab para la representación de los datos del registro Vsense obtenidos y la realización de los cálculos relativos al consumo. Dicho código se explica a continuación. (*Figura 2.28.*)

	A	B
1	0	8,403
2	253,604996	8,285
3	254,001007	8,207
4	254,034012	8,168
5	254,132004	8,168
6	254,237015	8,363
7	254,337021	8,207
8	254,43602	8,09

Figura 2.28. Excel con datos de tiempo y Vsense

Diseño del Script de Matlab para representación gráfica del consumo

En primer lugar se almacenan los datos en una matriz con la instrucción:

```
tabla_sense=xlsread('nombre_archivo.xlsx');
```

A continuación se resuelve el error de la información de los tiempos restando el primer dato temporal distinto de cero (es decir, el segundo) y despreciando los datos anteriores:

```
tabla_sense(:,1)=tabla_sense(:,1)-tabla_sense(2,1);
tabla_sense(1,:)=[];

Se guardan los datos de cada columna para su representación:
time=tabla_sense(:,1);
vsense=tabla_sense(:,2);

Se representa en una gráfica los valores tomados del registro:
grid on
plot(time,vsense,'r')
xlabel('Time(s)')
ylabel('Vsense(mV)')
```

Dando como resultado (*Figura 2.29.*):

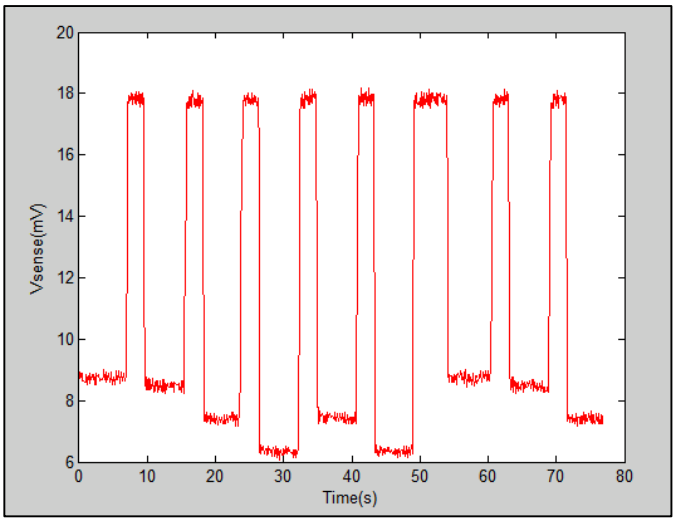

Figura 2.29. Gráfica Matlab de Tiempo (s) vs Vsense (mV)

Tras ello se calcula la corriente circulante por la resistencia de shunt aplicando la Ley de Ohm para el valor de resistencia utilizado:

```
Rshunt=2;
IBUS=vsense/Rshunt;
```

Se representa la corriente IBUS en función del tiempo:

```
graphic=plot(time,IBUS_mA);
xlabel('Time(s)');
ylabel('IBUS(mA)');
```

Dando como resultado (*Figura 2.26.*):

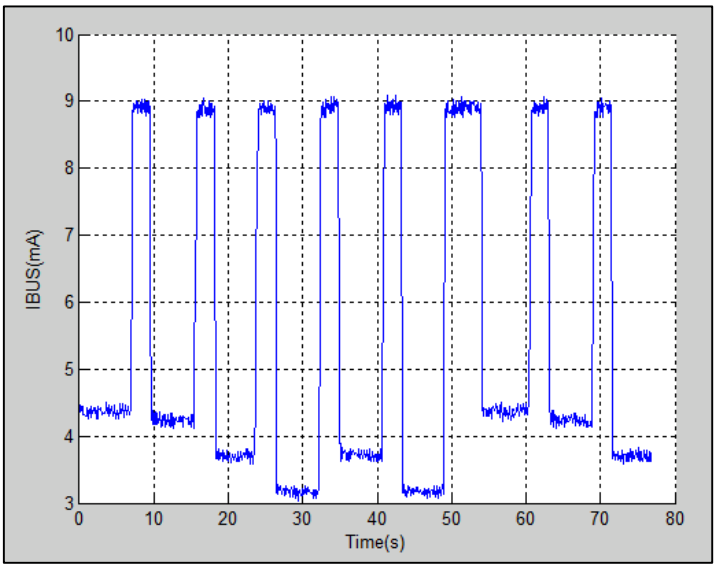

Figura 2.26. Gráfica Matlab tiempo (s) vs IBUS (mA)

Como primer análisis se calcula la capacidad por el método de **Coulomb counting** (Ver apartado 1.4.4.b.), es decir, integrando la corriente en función del tiempo.

Para ejemplificar este método, se toma como dato de capacidad inicial la capacidad nominal de la pila de botón **cr2450** [40].

```
[rows, colums] = size(IBUS_mA);
C_boton=650*3600; %ma*s
```

La integración se realiza en este caso por aproximación de cálculo del área por su mayor sencillez. Dado que se disponen de una gran cantidad de puntos para la gráfica, dicha aproximación resulta bastante precisa:

```
integration=zeros(1,2);
for i=1:(rows-1)
altura=abs(IBUS_mA(i+1)-IBUS_mA(i));
base=abs(time(i+1)-time(i));
integration(1)=base*altura;
integration(2)=integration(2)+integration(1);
end
```

La capacidad consumida será el resultado de la integración de la corriente en función del tiempo de la muestra. La restante, la diferencia en entre la capacidad de la batería al inicio (en este caso la capacidad nominal) del análisis menos la capacidad consumida:

```
Capacidad_consumida=integration(2)
Capacidad_restante=C_boton-Capacidad_consumida
```

Por último, el estado de carga será la relación entre la capacidad restante y la capacidad nominal:

```
SoC=Capacidad_restante/C_boton
```

Los resultados obtenidos para el ejemplo mostrado son los siguientes (*Figura 2.27.*):

```
Capacidad_consumida =

   13.7805


Capacidad_restante =

  2.3400e+006


SoC =

    1.0000
```

Figura 2.27. Ejemplo de resultados de estimación del SoC de una pila CR2450 con Matlab

Teniendo en cuenta el corto periodo de tiempo de la muestra y el bajo consumo durante la misma, los resultados son los esperados.

2.8.2. ADQUISICIÓN DE DATOS CON LOS DRIVERS DISEÑADOS

El procedimiento seguido es el siguiente:

1. Se realiza el montaje de adquisición A2 (ver apartado 3.1)

2. Mediante comandos por puerto serie (explicados en el apartado 2.10.), se inicializa el sensor indicando la resistencia de shunt.

```
int8_t PAC1710_init(float R_shunt);
```

3. De igual modo, se configura el sensor: modo, rango, tiempo de conversión, promediado, etc.

PACconf Mode/Rate/Vsource/Vsense…

4. Una vez hecho esto, se elige entre hacer una medida continua o una medida con pasos de tiempo. Por ejemplo:

```
status PAC1710_cont_measure(10,PAC1710_V_MODE,AVERAGE_OFF);
status PAC1710_one_shot_measure(5,10,20);
```

5. Se indica según el caso, el número de medidas y/o el tiempo entre medidas.

6. Los datos se enviarán por puerto serie en formato de tabla con los datos separados por ";".

7. Se guardan los datos en un archivo de texto. El formato está preparado para que con el script de Matlab anterior citado sea posible visualizar el valor de corriente medido.

2.9. DISEÑO DE APLICACIONES PARA EL CAMBIO CONTROLADO DE LOS MODOS DEL MICROCONTROLADOR DE ATMEGA

Para poder obtener información sobre los distintos estados en los que se puede encontrar la placa de Atmel, ha sido necesario diseñar procesos en los que se configura el micro en los modos a estudiar de manera controlada. A continuación se pasará a explicar en qué consisten éstos programas y la funcionalidad de los mismos.

2.9.1. APLICACIÓN EN ATMEL STUDIO PARA EL ANÁLISIS DE LOS MODOS DE SUEÑO

Para el estudio del consumo en los distintos modos de sueño, se cargó en la placa un código de ejemplo de los disponibles en AVR Atmel Software Framework: **Common API for Sleep Management / Example for ATmega256RFR2_XPLAINED_PRO kit** [41].

Tras el inicio del dispositivo o su reseteo, permanece en el modo activo durante tres segundos. Mientras se encuentra en el modo activo, el led amarillo de la placa permanece encendido.

El led se apaga y pasará a un primer modo de sueño. Tras pulsar el botón se volverá al modo activo y al soltarlo pasará al siguiente modo de la lista, correspondientes a los modos explicados en el apartado 2.1.1.b.ii.:

- **SLEEPMGR_ACTIVE:** modo activo.
- **SLEEPMGR_IDLE:** modo de inicio.
- **SLEEPMGR_ADC_NOISE_REDUCTION:** modo reducción de ruido de ADC.
- **SLEEPMGR_ESTDBY:** modo espera extendida.
- **SLEEPMGR_PSAV:** modo ahorro de energía.
- **SLEEPMGR_STDBY**: modo espera.
- **SLEEPMGR_PDOWN:** modo apagado.

Una vez recorridos todos los modos, se vuelve al modo inicial.

El funcionamiento paso a paso del código es el siguiente:

1. Se selecciona el primer modo de sueño en el que se iniciará el dispositivo

2. Se inicialan los pines de entrada y salida y se configura la interrupción del botón USER

3. Se inicializa el reloj y se deshabilitan los módulos no utilizados.

4. Se inicializan los puertos de entrada y salida.

5. Se enciende el led amarillo del nodo y se inicia en el modo seleccionado, manteniéndose en ese estado durante 3 segundos.

6. Paso al primer modo de sueño. Se mantiene hasta que se pulsa el botón USER.

7. Mientras se mantiene presionado el botón, el nodo se encuentra en modo activo y permanece.

8. Se pasa al siguiente modo de sueño, permaneciendo en él hasta que se vuelva a pulsar el botón.

En el siguiente diagrama de bloque se puede ver de forma más sencilla el comportamiento de esta aplicación (*Figura 2.30.*):

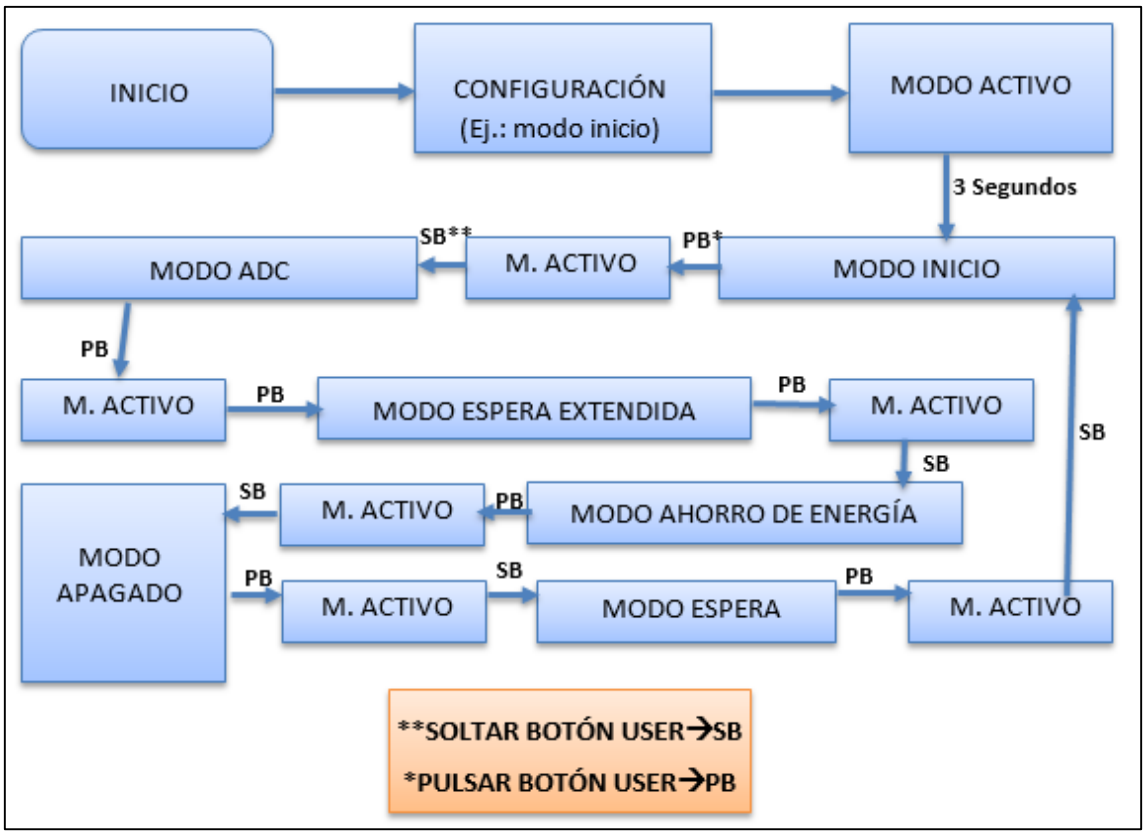

Figura 2.28.Diagrama de bloques de aplicación para la medición de los modos de sueño

2.9.2. APLICACIÓN EN ATMEL STUDIO PARA LA MEDICIÓN DE LOS MODOS ACTIVO, SUEÑO Y DE TRANSMISIÓN

Para acercarse a la medida de consumo de una rutina más real de un nodo teniendo en cuenta que forma parte de una red inalámbrica, se diseñó este código. Se trata de un bucle continuo en el que la placa va pasado de forma automática por tres estados básicos: sueño o baja energía, activo y comunicación.

Como se desconocía la existencia de alguna norma que regula la validación de este tipo de ensayos, se trató de establecer tiempos aleatorios para cada modo, teniendo en cuenta que en un buen diseño, el nodo debería permanecer más tiempo en modos de bajo consumo.

Así, se establecieron tres tiempos distintos para cada modo, asignándose uno u otro mediante la comparación de 4 variables aleatorias definidas.

El código se encuentra comentado en la sección A.1. No obstante, se puede ver el diagrama en la *Figura 2.29*.

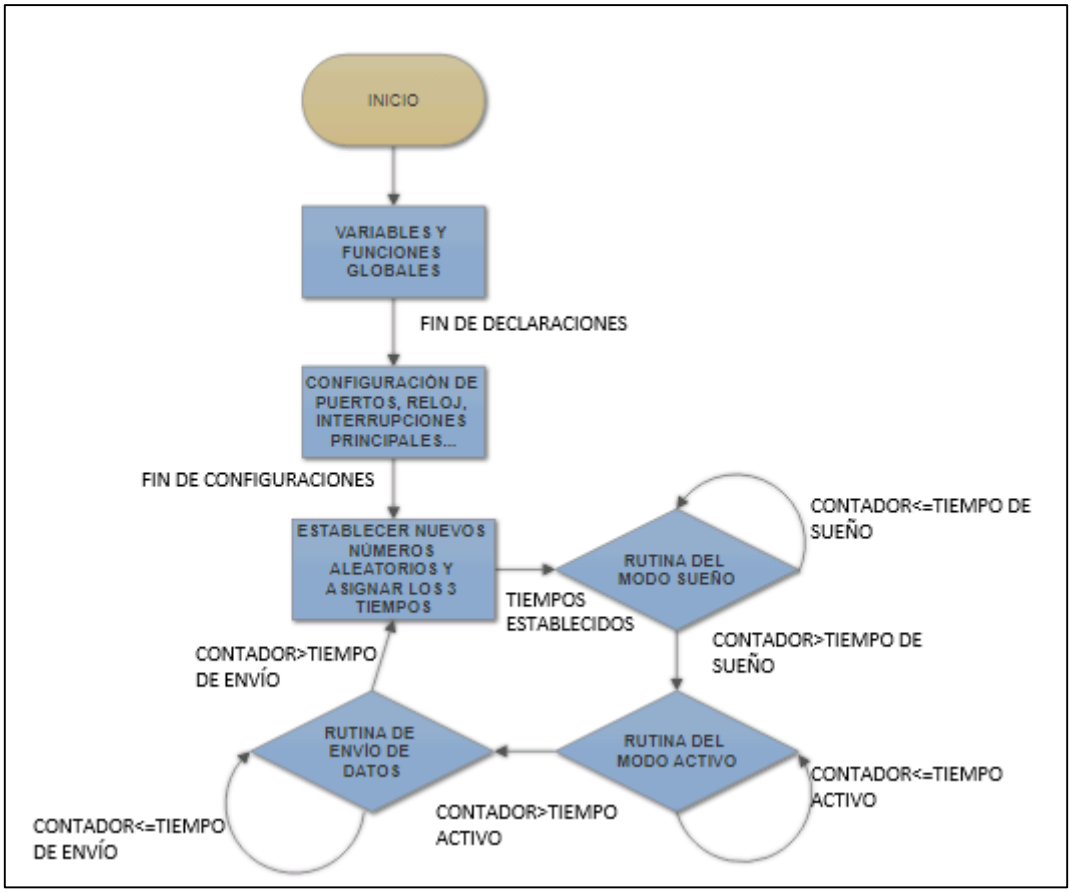

Figura 2.29. Diagrama de rutina de modos de sueño, activo y de comunicación

Una vez explicados todos los programas diseñados para la medida de consumo en diferentes condiciones, se pasa a comentar el sistema de comandos implementado para controlar las funcionalidades del sensor de forma sencilla por puerto serie.

2.10. DISEÑO DE SISTEMA DE COMANDOS POR PUERTO SERIE PARA LA CONFIGURACIÓN DEL SENSOR Y LA ADQUISICIÓN DE DATOS

Dentro de la stack proporcionada, existe un módulo diseñado por José Ángel Miranda Calero [38] para la gestión de todas las aplicaciones del micro a través de comandos por puerto serie.

Básicamente se trata del acceso a las distintas funciones de cada periférico del micro (en este caso el sensor de corriente) por medio de la escritura por puerto serie de comandos jerarquizados a través de un terminal específico (en este proyecto se ha utilizado Termite).

En la *Tabla 2.11.* y la *Tabla 2.12.* se indican los comandos programados para la aplicación del sensor de corriente. Se ha divido en comandos de inicialización (**PACinit**), de configuración (**PACconf**), medida (**PACmeas**), de información del sensor (**PACinfo**) y de testeado de I2C (**PACtest**), donde el prefijo PAC hace referencia al nombre del sensor, PAC1710, para distinguir estos comandos de forma sencilla respecto de los de otros dispositivos conectado al micro.

1º COMMAND	2º COMMAND	3º COMMAND	4º COMMAND
PACinit	-10R		
	-2R8		
	-def		
PACconf	Mode	I	
		V	
		IV	
		Stby	
	Rate	1	
		2	
		3	
		cont	
	Vsource	-time	-2.5
			-5
			-10
			-20
		-avr	-off
			-2
			-4
			-8
	Vsense	-time	-2.5
			-5
			-10
			-20
			-40
			-80
			-160
			-320
		-avr	-off
			-2
			-4
			-8
		-rang	-10
			-20
			-40
			-80

Tabla 2.11. Listado de commandos 1

1º COMMAND	2º COMMAND	3º COMMAND	4º COMMAND
PACinfo	Mode		
	Rate		
	Vsource	-time	
		-avr	
		-all	
	Vsense	-time	
		-avr	
		-rang	
		-all	
	All		
PACmeas	-cont		
	-shot		
PACtest			

Tabla 2.12. Listado de comandos 2

En la *Figura 2.30.* se muestra un ejemplo de la salida de datos por puerto serie utilizando Termite.

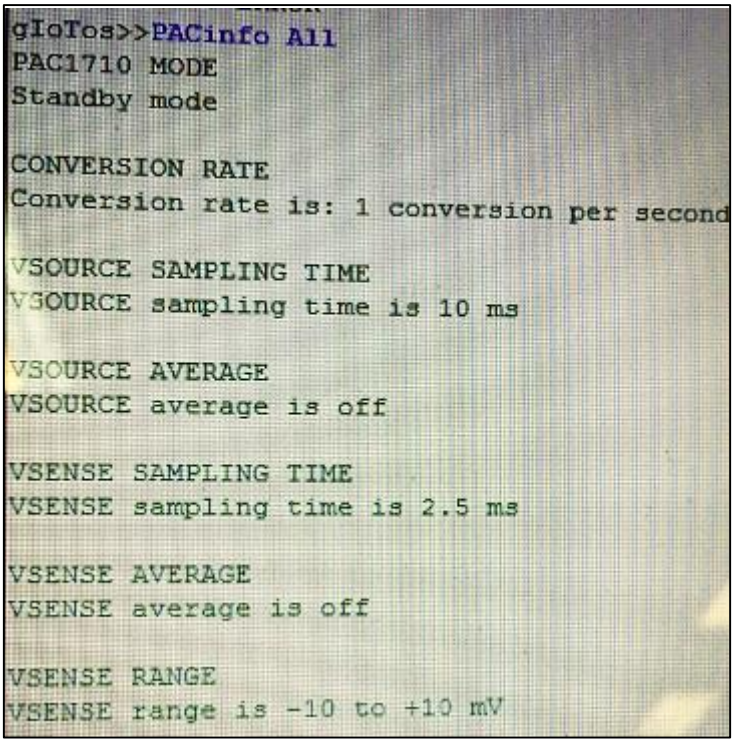

Figura 2.30. Ejemplo de visualización de comandos por puerto serie

3. SISTEMA DE MEDIDA DEL CONSUMO DEL NODO PARA DISTINTOS ESTADOS

En este apartado se describirá todo el proceso de toma de dantos y análisis, desde la configuración del sensor hasta su representación y estudio. Se dividirá en cuatro apartados:

- Descripción del montaje
- Medidas iniciales
- Medidas con la placa de evaluación PAC1710
- Medidas con la PCB y los drivers diseñados

3.1. DESCRIPCIÓN DEL MONTAJE

Tanto para las medidas con la PCB, como con el polímetro, el osciloscopio y la placa de evaluación relativas al consumo del nodo, se siguieron dos configuraciones de montaje principalmente.

1. Medición del consumo de la placa

El montaje consiste en colocar la resistencia de medida entre la salida positiva de la fuente y la entrada positiva de la placa de Atmega, cerrando el circuito al unir la masa de la fuente con el pin de GND del micro (Ver *Figura 3.1. y Figura 3.2.*).

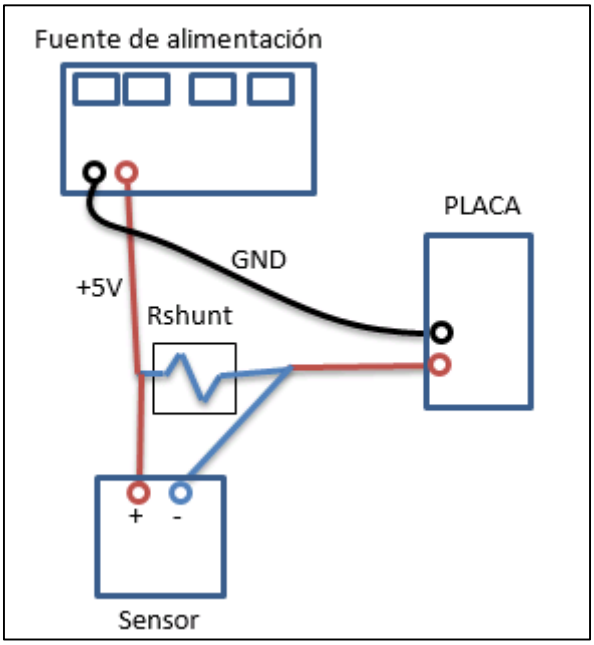

Figura 3.1. Set-up para la medición del consumo de la placa (M1)

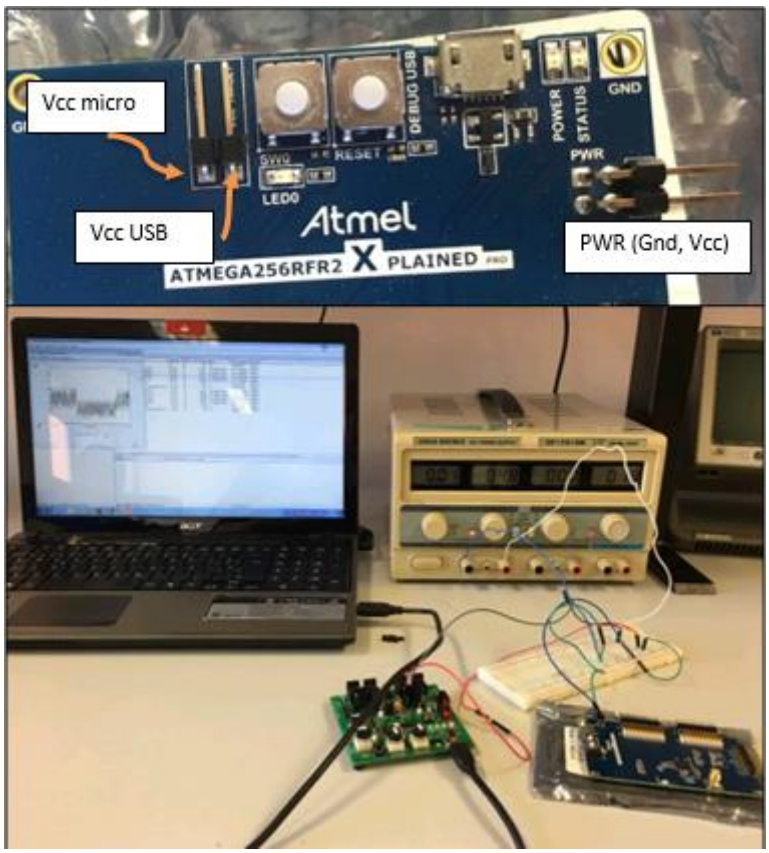

Figura 3.2. Imagen del montaje M1. Pines PWR y Current Header

Ésta fue la configuración inicial, sin embargo se trató de medir el consumo durante el envío de datos por RF del sensor, con el código de "Comunicación RF" explicado en el anexo y se comprobó que éste método no era adecuado ya que se observaban los picos de corriente durante la comunicación. Los datos obtenidos eran constantes picos de 3 a 0.5 mA, y dicha forma de onda no parecía cambiar a pesar de cambiar por comando un el tiempo entre envíos. Además se esperaban picos de en torno a 16 mA según la información del datasheet.

2. Medición del consumo del microcontrolador

Consiste en conectar la resistencia de medida entre los pines que comunican la alimentación de la placa por USB y el pin de alimentación del microcontrolador. Más información sobre ello se puede encontrar en el datasheet [25] en el apartado "**Current Measurement Header".**

Hay que tener especial cuidado en este montaje, ya que si se alimenta la placa mientras no le llega tensión al microcontrolador, pueden producirse daños irreparables (Ver *Figura 3.3. y 3.4.*).

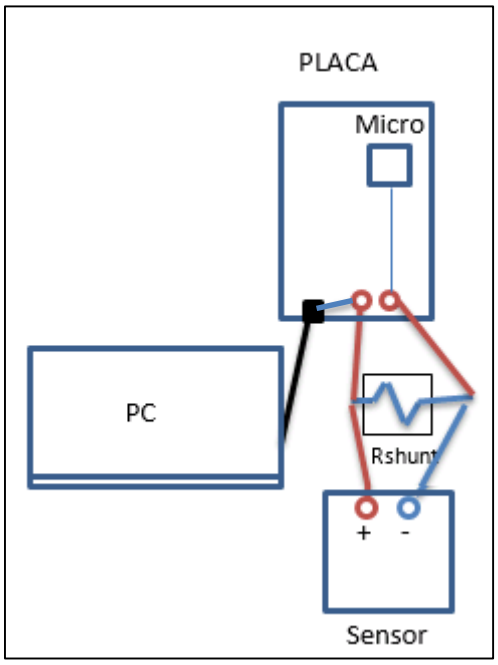

Figura 3.3. Set-up para la medición del consumo del microcontrolador (M2)

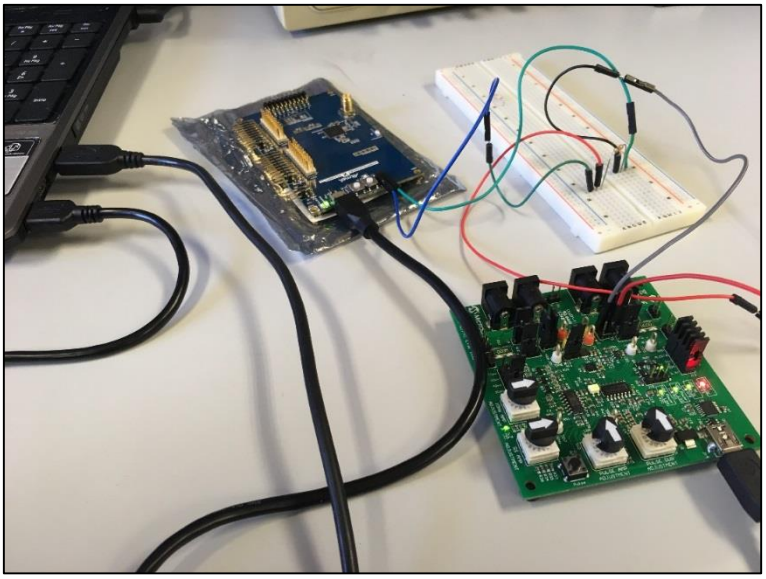

Figura 3.4. Imagen del montaje M2

En cada uno de las Figuras anteriores el "Sensor" representa el dispositivo de media, pudiendo ser el polímetro, el osciloscopio, la placa de evaluación o la PCB. El montaje no se diferencia entre cada una más que la utilización de sondas o cables y en el cambio de valor de la resistencia de medida.

Dentro de los montajes explicados anteriormente, aún no se ha hecho mención de cómo se adquieren los datos. En todos los casos lo que se obtiene principalmente es la caída de tensión por la resistencia de medida y la tensión de la fuente.

- En el caso del polímetro la medida se apuntó tras observar el display, una vez seleccionada la medida de voltaje en un rango adecuado.
- Con el osciloscopio, se seleccionó una escala de tiempo y voltaje adecuados y se capturó y almacenó en un USB los instantes en los que la forma de onda de la tensión por la resistencia ofreciese información relevante.
- Con la placa de evaluación del sensor de corriente, hay dos configuraciones posibles de adquisición de datos:
    - **Utilizando la aplicación Chip Manager**

      La placa de evaluación está adaptada para comunicarse por i2C al PC a través del USB, lo que permite configurarla con la aplicación ChipMan sin necesidad de un circuito adicional (Ver *Figura 3.5.*).

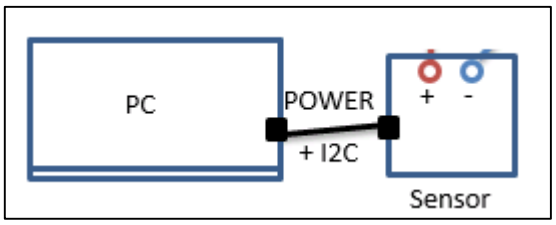

Figura 3.5. Montaje de adquisición de datos con Chip Manager (A1)

Como se explicó en la sección 2.5.2.c. y 2.7.1., este software tiene la opción de graficar el valor de los registros de medida Vsense, Vsource y Pratio y exportarlos a un fichero de texto. Dicho fichero de texto puede ser interpretado en un formato de tabla de tiempo/ valor medido en Excel, o como una matriz en Matlab, pudiendo representarse de forma sencilla la forma de onda con estas herramientas.

- **Utilizando los drivers diseñados en Atmel Studio**

  Para usar los drivers implementados, se utilizan dos nodos: el nodo a medir y aquél que se comunique con el sensor. En el caso de que se emplee la placa de evaluación, es necesario realizar una transición con las resistencias de Pull-Up de las línea SDA y SCL y la unión de las masas nodo-sensor (*Figuras 3.6. y 3.7.*). (Ver apartado 2.3.5. Comunicación I2C).

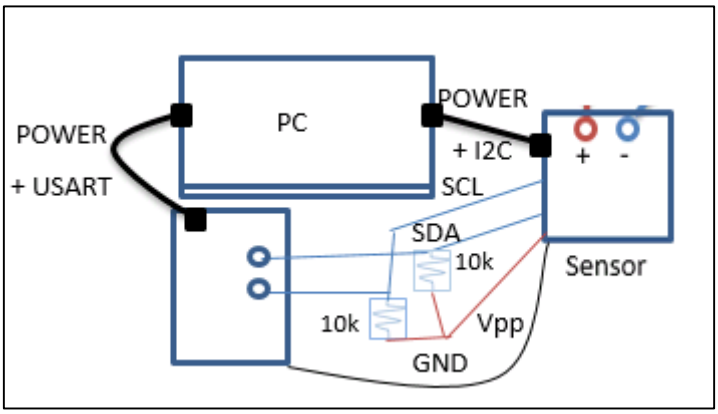

Figura 3.6. Montaje de adquisición de datos con los drivers (A2)

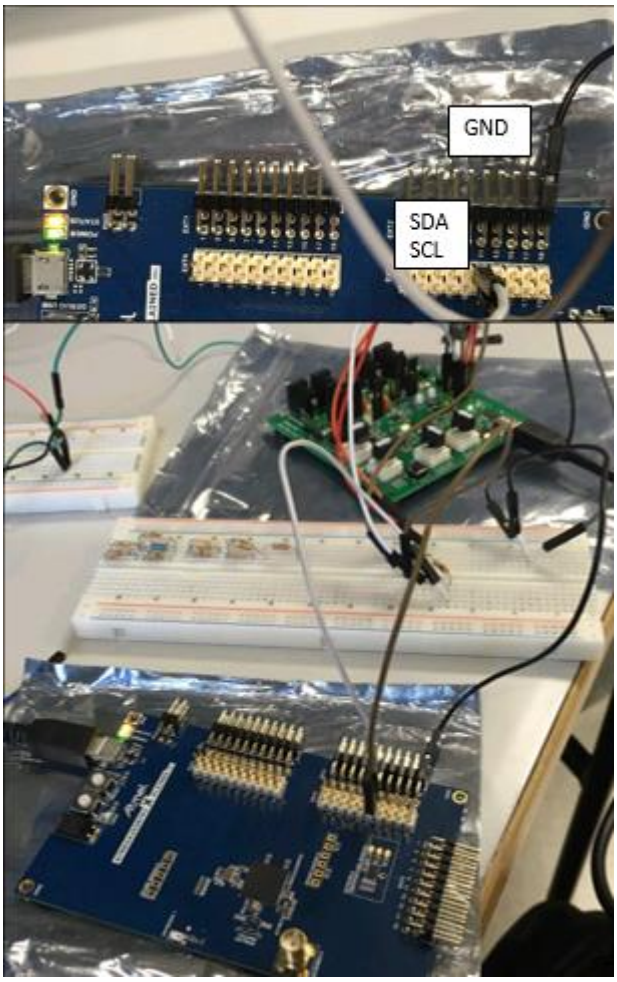

Figura 3.7. Imagen de montaje A2. Pines I2C.

3.2. MEDIDAS INICIALES Y USO DEL OSCILOSCOPIO Y EL POLÍMETRO

En primer lugar se comprobó el funcionamiento del sensor utilizando los modos de demostración disponibles en la placa de evaluación. Con ellos era posible aplicar una tensión y corriente conocidas y compararlas con las leídas en los registros. Sin embargo no fue posible realizar ensayos de este tipo para corrientes de mA, ya que las resistencias de medida por defecto de la placa de evaluación eran de 3 mΩ y se sobrepasaba el rango de tensiones máximo de 80 mV.

Por otro lado, se trató de utilizar el polímetro para tomar valores de corriente de distintos estados para obtener las primeras referencias de corriente demandada. Es un método útil para modos en los que el consumo no cambie bruscamente en un tiempo corto (de ms), como en el caso en el que se mide la corriente del nodo cuando está en modo sueño o activo sin transmisión. En el resto de medidas, el polímetro no es suficiente, ya que la velocidad de adquisición de datos mediante la observación del display es insuficiente.

A continuación, se comenzó a monitorizar la corriente en tiempo real del nodo con la aplicación de Chip Manager. Se pudo comprobar que para modos que presentan pocos picos de corriente la gráfica se asemejaba al resultado esperado. No obstante, resultaba más sencillo distinguir entre modos o estados cuando se medía directamente el consumo del micro.

El siguiente paso fue medir el consumo durante la transmisión de datos. Por medio de la aplicación Packeet Sniffer era posible ver en tiempo real en los paquetes enviados y conocer así con precisión los momentos en los que se estuviese transmitiendo información, como se puede ver en la *Figura 3.8*.

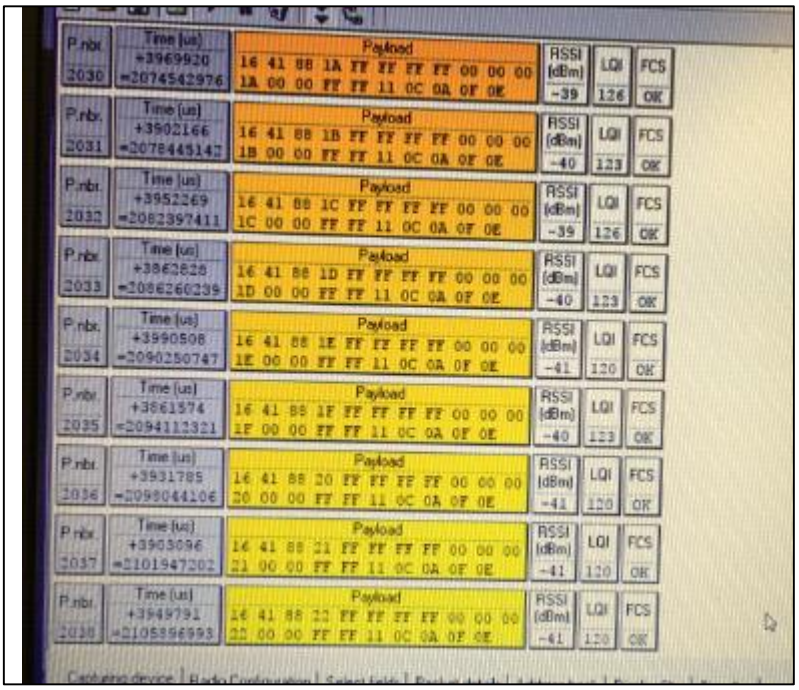

Figura 3.8. Visualización de paquetes enviados a través de PAcket Sniffer

En este primer test, empezó a observarse ciertas limitaciones del sensor. No se apreciaban picos de corriente característicos de la comunicación ni variaciones en la gráfica al cambiar el tiempo entre envíos de 1ms a 1s de 4 a 16bits.

Debido a ello, se procedió a intentar visualizar con el osciloscopio la onda característica de la caída de tensión por la resistencia de shunt, para lo cual fue necesario utilizar 33 ohmios. Con esto último, como se puede ver en las Figuras 3.9 y 3.10., fue posible determinar lo siguiente:

- El pico corriente que debe observarse es de en torno a los 12 mA.
- La transmisión del paquete de datos se efectúa en aproximadamente 1.5 ms.

Con ello, se encuentran las siguientes limitaciones del sensor:

- El tiempo de toma de datos mínimo configurable es de 2.5 ms. Además, a menor tiempo, menor resolución
- El rango de tensión máximo es de 80 mV. De igual modo, habrá menor resolución cuanto mayor sea este valor.

Con la configuración descrita y teniendo en cuenta las limitaciones, se observó lo siguiente *Figura 3.9. y 3.10.)*:

Figura 3.9. Señal del osciloscopio de envío de trama de 4bits cada 1ms. A 200mV/div y (500us/100ms)/div

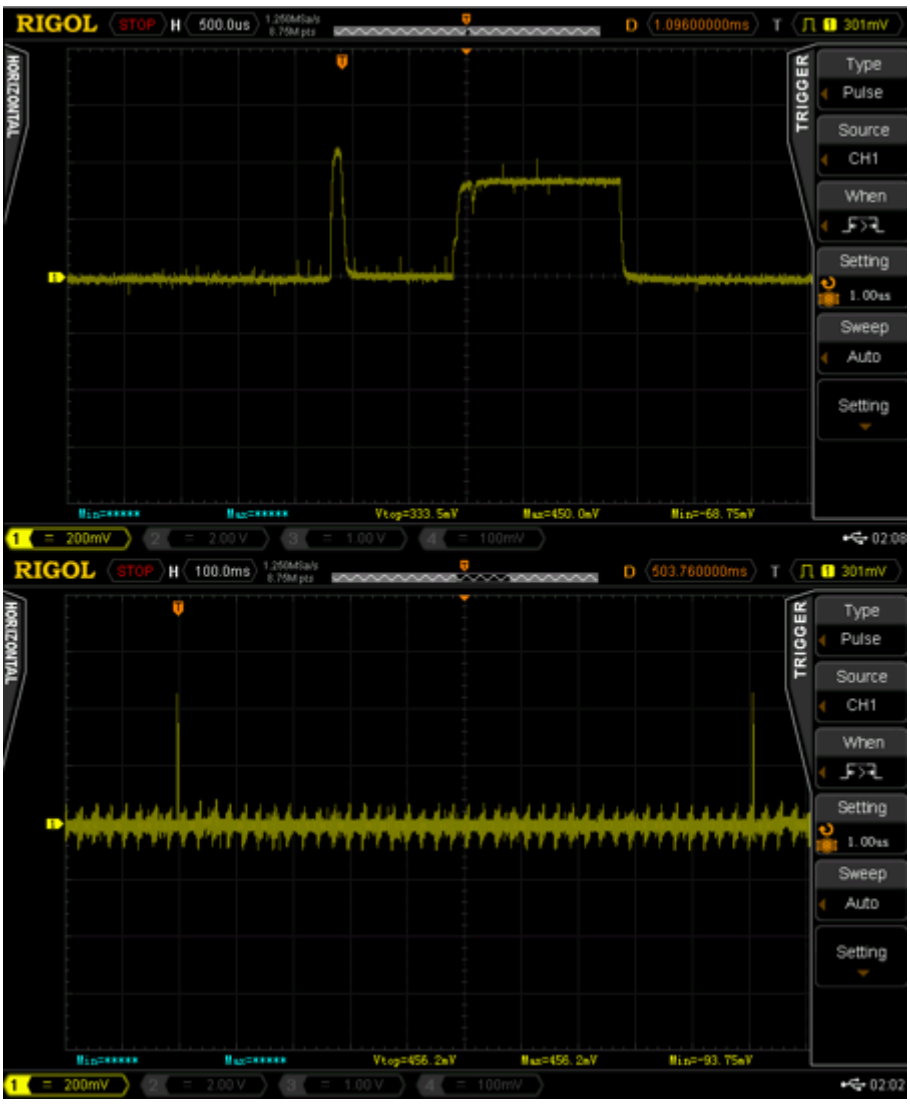

Figura 3.10. Señal del osciloscopio de envío de trama de 16bits cada 1s. A 200mV/div y (500us/100ms)/div

- Se ven dos picos. El primero, de 200us de duración, es debido a la comunicación y encendido de la antena de RF). El segundo guarda relación con el envío, de duración variable dependiendo del tamaño del paquete transmitido. El pico corriente que debe observarse es de en torno a los 420mV/33Ω=12,72 mA (pico de comunicación micro-antena RF). Concretamente durante el envío de la trama se están consumiendo 360mV/33Ω=10,9 mA. Para medir esos niveles de corriente la Rshunt máxima es de 80mV/16mA=5 ohmios, ya que un valor mayor sobrepasaría el rango de tensión.
- La transmisión del paquete de datos se efectúa en aproximadamente en 1.5 ms en el caso de 4 bits, y en 1.2ms.

Con ello, se pueden encontrar las siguientes limitaciones del sensor:

- El tiempo de toma de datos mínimo configurable es de 2.5 ms. Además, a menor tiempo, menor resolución
- El rango de tensión máximo es de 80 mV. De igual modo, habrá menor resolución cuanto mayor sea este valor.
- Para medir esos niveles de corriente la Rshunt máxima es de 80mV/16mA=5 ohmios, ya que un valor mayor sobrepasaría el rango de tensión.

3.3. MEDIDAS CON LA PLACA DE EVALUACIÓN PAC1710

Como la resistencia por defecto de la placa de evaluación era de 3 mΩ, se sacaron dos cables de los jumer de Vsense+ y Vsense- y se conectó la resistencia de shunt que más se ajustase a las necesidades.

Para todas las pruebas realizadas, se ha tratado de medir en consumo bajo las mismas condiciones del micro y de la placa. A continuación se pueden ver las pruebas realizadas para la rutina de cambio de modo con tiempos pseudo-aleatorio.

En primer lugar se hizo una prueba de concepto de la medida de consumo de la placa en su totalidad.

MEDIDA DE LA PLACA (Montaje M1-A1)

En la *Figura 3.11.* queda manifiesto que con un tiempo de muestreo de 80 ms apenas hay diferencia entre la señal obtenida durante la comunicación y el consumo del modo activo. Aunque la resolución es mayor, es un tiempo demasiado grande como para visualizar el aumento de corriente durante el envío de datos.

Ensayo 1

- **Rango:** +/- 20 mV.
- **Resistencia de shunt:** 1 Ω.
- **Tiempo de muestreo:** 80 ms.
- **Bits:** 12 bits.
- **Resolución:** 9,76 µV.

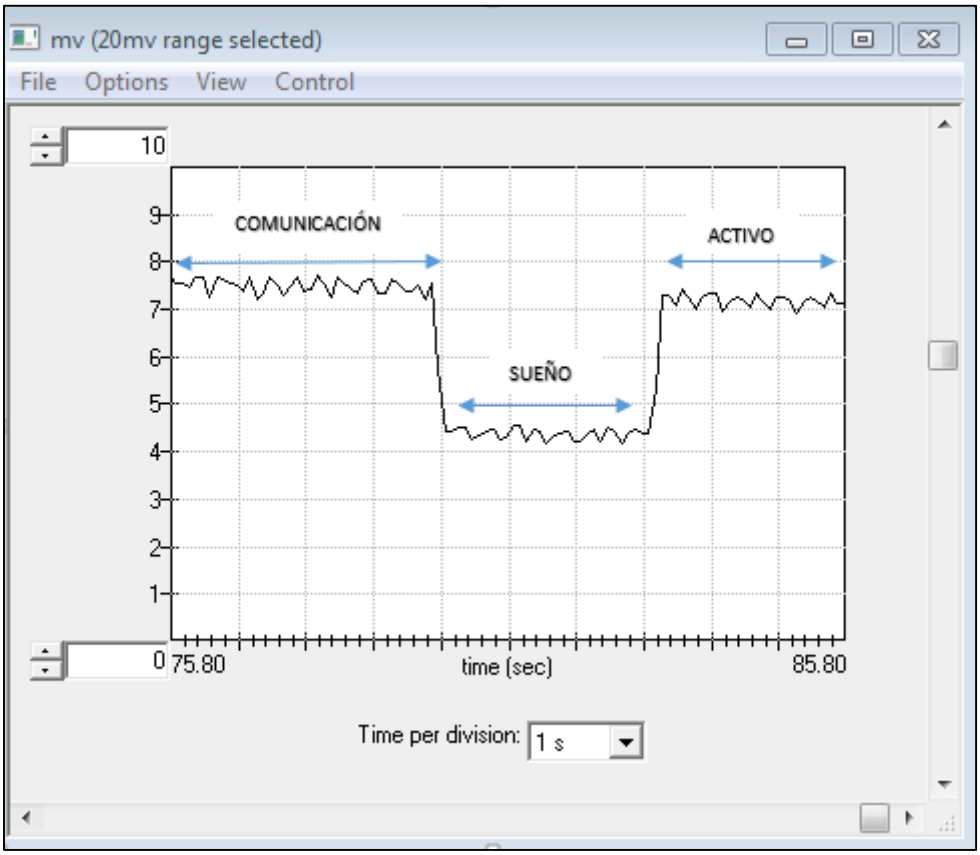

Figura 3.11. Test de medida de placa 12 bits.

Por otro lado, en la *Figura 3.12.*, en la que se ha bajado el tiempo de muestre al mínimo (2,5 ms), se obtienen datos erróneos de Vsense, iguales o incluso menores que cero. Al contrario que en el ensayo anterior, el tiempo de muestreo es el mejor posible, a costa de la resolución, demasiado baja para poder adquirir datos con suficiente precisión.

Ensayo 2

- **Rango:** +/- 20 mV.
- **Resistencia de shunt:** 1 Ω.
- **Tiempo de muestreo:** 5 ms.
- **Bits:** 7 bits.
- **Resolución:** 312,5 µV.

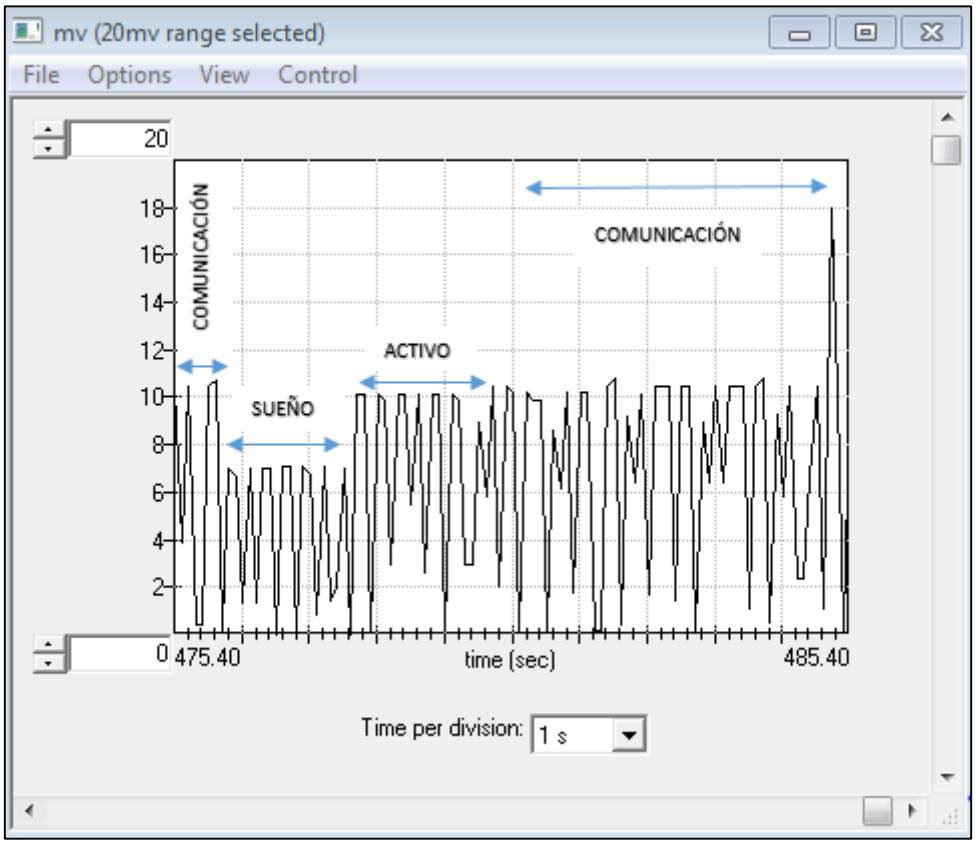

Figura 3.12. Test de medida de placa 7 bits.

Tras estas pruebas, se procedió a medir la corriente demandada directamente por el microcontrolador, sin tener en cuenta el resto de la placa.

MEDIDA DEL MICRO (Montaje M2-A1)

En la *Figura 3.13.* se muestra la medida del consumo del micro bajo las mismas condiciones que las anteriores.

Ensayo 3

- **Rango:** +/- 20 mV.
- **Resistencia de shunt:** 1 Ω.
- **Tiempo de muestreo:** 10 ms.
- **Bits:** 8 bits.
- **Resolución:** 156,3 µV.

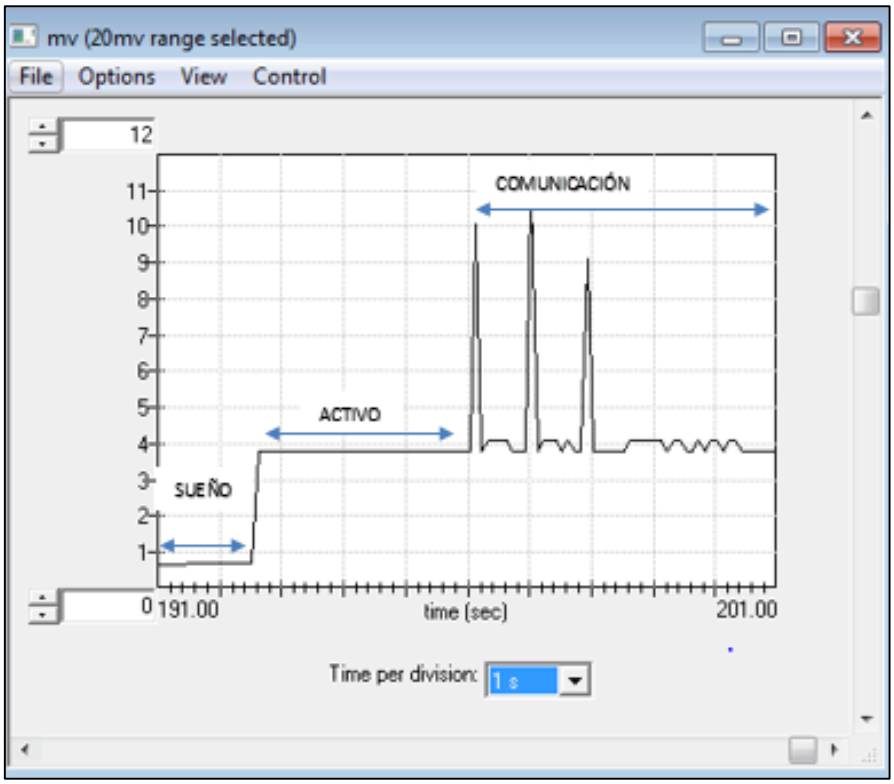

Figura 3.13. Test de medida del micro 8 bits.

Se puede observar que el sensor consigue capturar alguno de los picos de corriente durante la comunicación. Sin embargo, se programó para que se produjera un envío cada milisegundo, con lo que se ve fácilmente que aún se pierde gran parte de la información del consumo real en esos momentos.

Con toda la información conseguido durante este proceso de testeo, se establecieron las pruebas a realizar para poder analizar el consumo:

- Parámetros:
  - Resistencia de shunt: 1 Ω.
  - Rango: mínimo posible para aumentar la resolución (+/- 20 mV).
  - Tiempo de muestreo: entre 10 y 20 ms.
- Medidas a realizar:
  - Modos de sueño, para la placa y para el micro
  - Medida simulada de un proceso típico del nodo, con tiempos de modo activo, sueño y de comunicación pseudo-aleatorios.
    - Para la placa y para el micro
    - Tiempos de 1 minuto, 10 minutos y 1 hora

A continuación se pasará a explicar el procedimiento de medida con los drivers diseñados.

3.4. MEDIDAS CON LA PCB Y LOS DRIVERS DISEÑADOS

En este caso, el procedimiento de medida es el mismo, salvo que ya no se utiliza la aplicación Chip Manager sino que se emplean los drivers diseñados. En este caso, se puede utilizar tanto la placa de evaluación como la PCB diseñada. No se apreció una gran diferencia

entre ambos casos, por lo que para evitar dañar la PCB, la mayor parte de las medidas se han realizado con la placa de evaluación.

Existen tres funciones implementadas para ello: una de ellas mide de forma continua la caída de tensión en la resistencia, la tensión en la fuente, o ambas. La segunda, realiza una medida simple. La restante, realiza una cantidad de medidas dada con un tiempo entre medidas determinado.

Una vez leído el dato, se envía por puerto serie, como se muestra en la *Figura 3.14*.

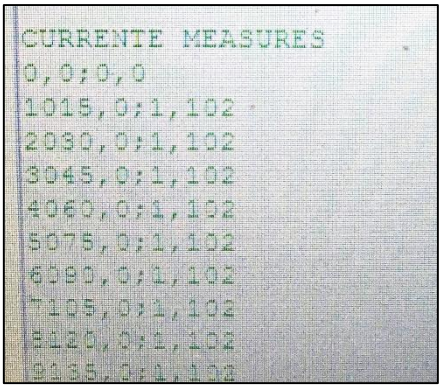

Figura 3.14. Salida de datos de corriente por puerto serie

Tras explicar cómo se han realizado las medidas, pasamos a ver los resultados obtenidos de las mismas.

4. RESULTADOS

En este apartado se muestran los resultados obtenidos de los ensayos realizados, así como un breve análisis del consumo.

Se divide en la medida de consumo para los distintos modos de sueño, y la medida durante una situación simulada de distintos modos con tiempos aleatorios para distintos tiempos, diferenciando entre la demanda de corriente del microprocesador y del microcontrolador.

4.1. MEDIDA DEL CONSUMO EN LOS DISTINTOS MODOS DE SUEÑO DEL NODO

En la Figura 4.1. Medida de consumo de los distintos modos de sueño*Figura 4.1.* se muestra la gráfica de corriente respecto al tiempo medida en el microcontrolador. Las características de la adquisición de datos son las siguientes:

- **Montaje**: M1
- **Rango:** +/- 80 mV
- **Resistencia de shunt:** 2,1Ω
- **Tiempo de muestreo:** 80 ms
- **Bits:** 12 bits

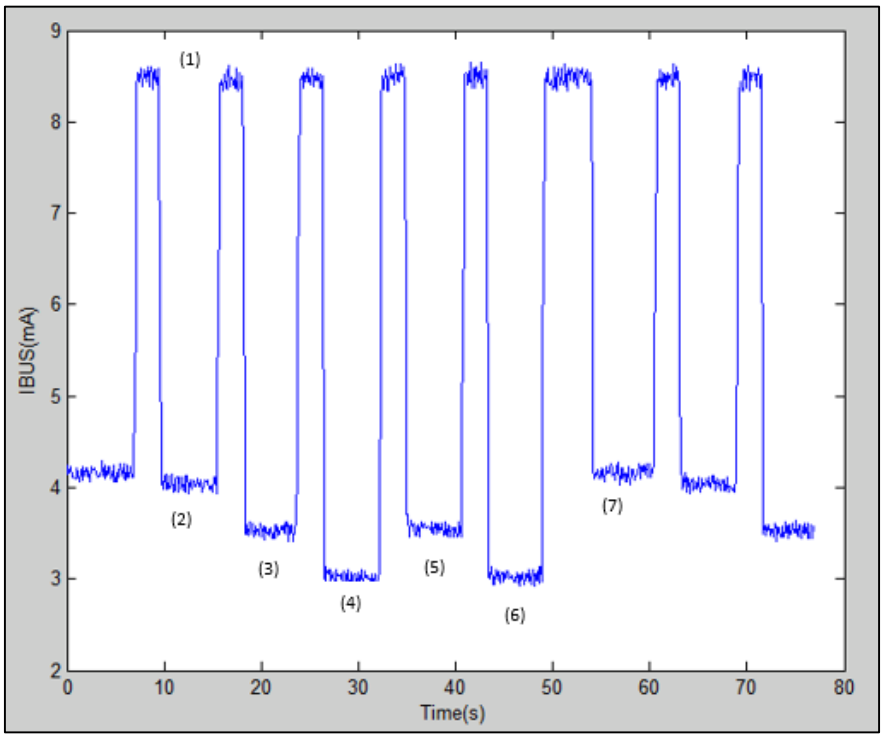

Figura 4.1. Medida de consumo de los distintos modos de sueño

En la *Tabla 4.1* se muestra la corriente demanda en el modo activo en cada modo de sueño (Ver la referencia numérica de la imagen anterior):

MODO	CORRIENTE MEDIDA
ACTIVE (1)	8,393 mA
IDLE (2)	3,98 mA
ADC NOISE REDUCTION (3)	3,55 mA
EXTENDED STANDBY (4)	3,015 mA
POWER SAVE (5)	3,592 mA
STANDBY (6)	3,033 mA
POWER DOWN (7)	4,206 mA

Tabla 4.1. Valores de consumo de la placa medidos para cada modo de sueño y modo activo

Tras analizar el consumo de los distintos modos, se pueden ordenar de menor a mayor corriente demandada:

EXTENDED STANDBY<STANDBY<ADC NOISE REDUCTION< POWER SAVE <IDLE<POW. DOWN

Así, suponiendo por ejemplo que en una determinada aplicación el nodo se encuentra el 80% del tiempo en uno de estos modos, la diferencia entre mantenerlo en POWER DOWN o STANDBY, por ejemplo, es de 1,173 mAh. En un proceso de 24 horas, estará 24*0,8=19,2 horas en modo sueño, lo que supondría una diferencia de 22,52 mAh diarios.

Por ello, es importante determinar qué modo de sueño es posible usar en cada momento y optimizar los procesos para mantener el nodo el mayor tiempo posible en el estado de menor energía permitido.

4.2. MEDIDA DEL CONSUMO EN UNA SITUACIÓN REAL SIMULADA CON TIEMPOS ALEATORIOS

MEDIDA DE LA PLACA

Medida de 1 minuto (*Figura 4.2*)

- **Montaje**: M1
- **Tiempo:** 1 minuto
- **Rango:** +/- 20 mV
- **Resistencia de shunt:** 1 Ω
- **Tiempo de muestreo:** 10ms
- **Bits:** 9 bits
- **Resolución:** 78.13 µV

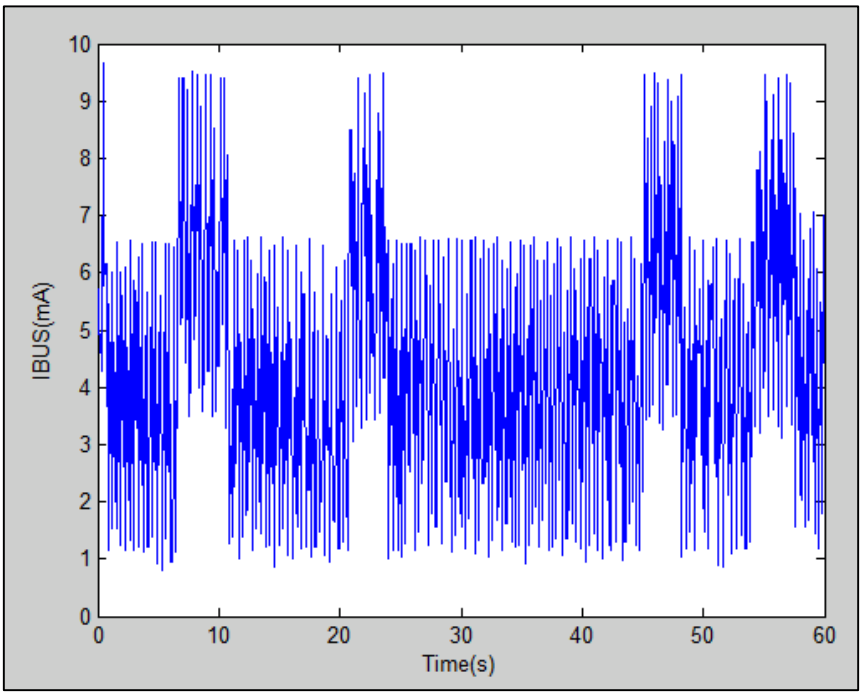

Figura 4.2.Medida de consumo 1 minuto placa

Medida de 10 minutos (*Figura 4.3*)

- **Montaje**: M1
- **Tiempo:** 10 minutos
- **Rango:** +/- 20 mV
- **Resistencia de shunt:** 1 Ω
- **Tiempo de muestreo:** 10ms
- **Bits:** 9 bits
- **Resolución:** 78.13 µV

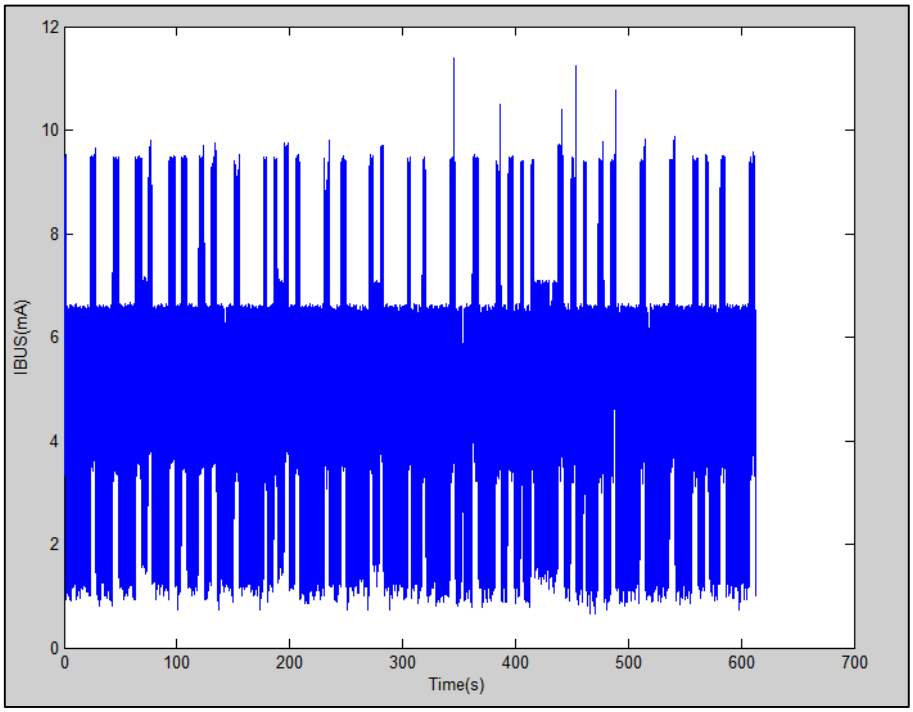

Figura 4.3. Medida de consumo 10 minutos placa

Medida de 60 minutos (*Figura 4.4*)

- **Montaje**: M1

- **Tiempo:** 60 minutos
- **Rango:** +/- 20 mV
- **Resistencia de shunt:** 1 Ω
- **Tiempo de muestreo:** 10ms
- **Bits:** 9 bits
- **Resolución:** 78.13 µV

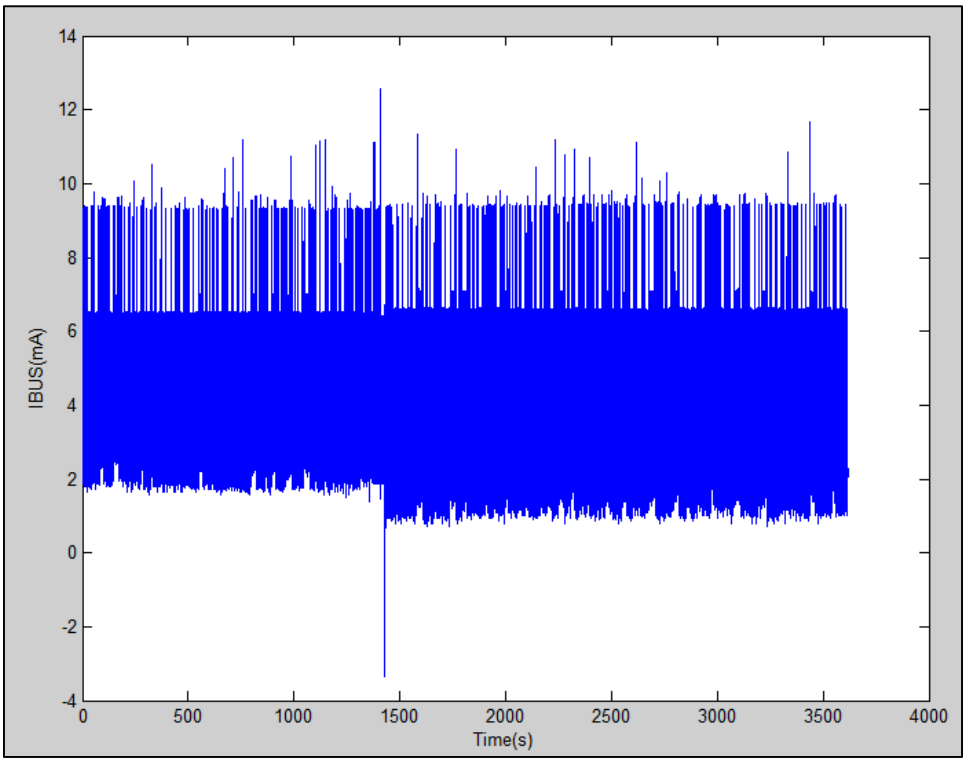

Figura 4.4.Medida de consumo 60 minutos placa

El pico que se observa en la gráfica fue debido a la desconexión de uno de los cables entre la resistencia y el sensor. Esto muestra la importancia de un montaje robusto, pues cualquier desconexión momentánea altera el resto de la medida.

MEDIDA DEL MICRO

Medida de 1minuto (*Figura 4.5*)

- **Montaje**: M2.
- **Tiempo:** 1 minuto
- **Rango:** +/- 20 mV
- **Resistencia de shunt:** 1 Ω
- **Tiempo de muestreo:** 5ms
- **Bits:** 8 bits
- **Resolución:** 156.3 µV

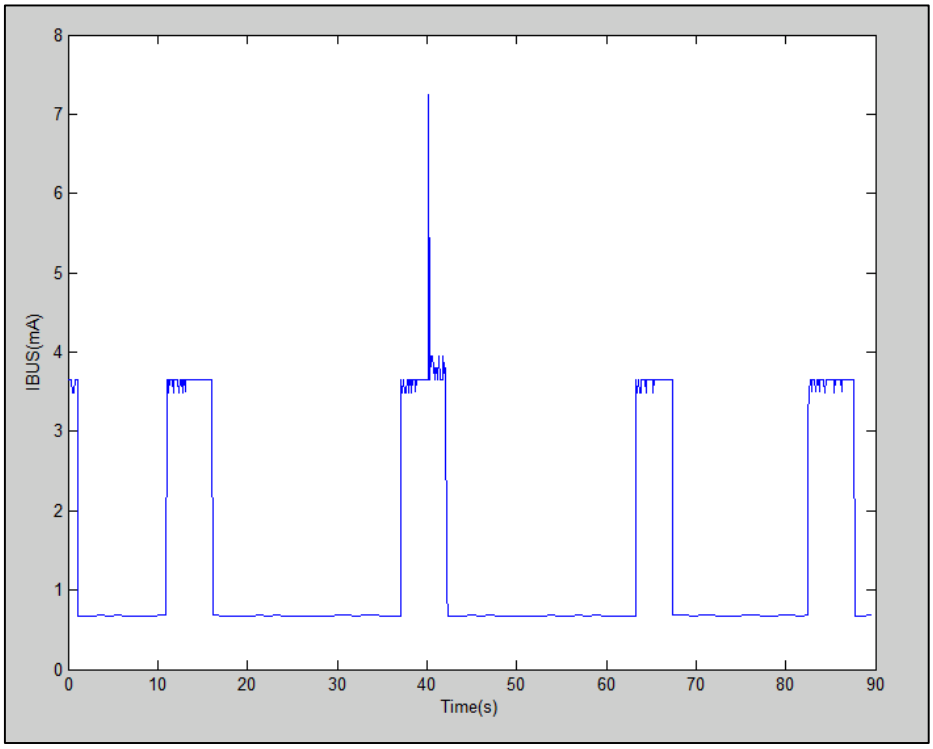

Figura 4.5.Medida de consumo 1minuto micro

Medida de 10 minutos (*Figura 4.6*)

- **Montaje**: M2.
- **Tiempo:** 10 minutos.
- **Rango:** +/- 20 mV.
- **Resistencia de shunt:** 1 Ω.
- **Tiempo de muestreo:** 5ms.
- **Bits:** 8 bits.
- **Resolución:** 156.3 µV.

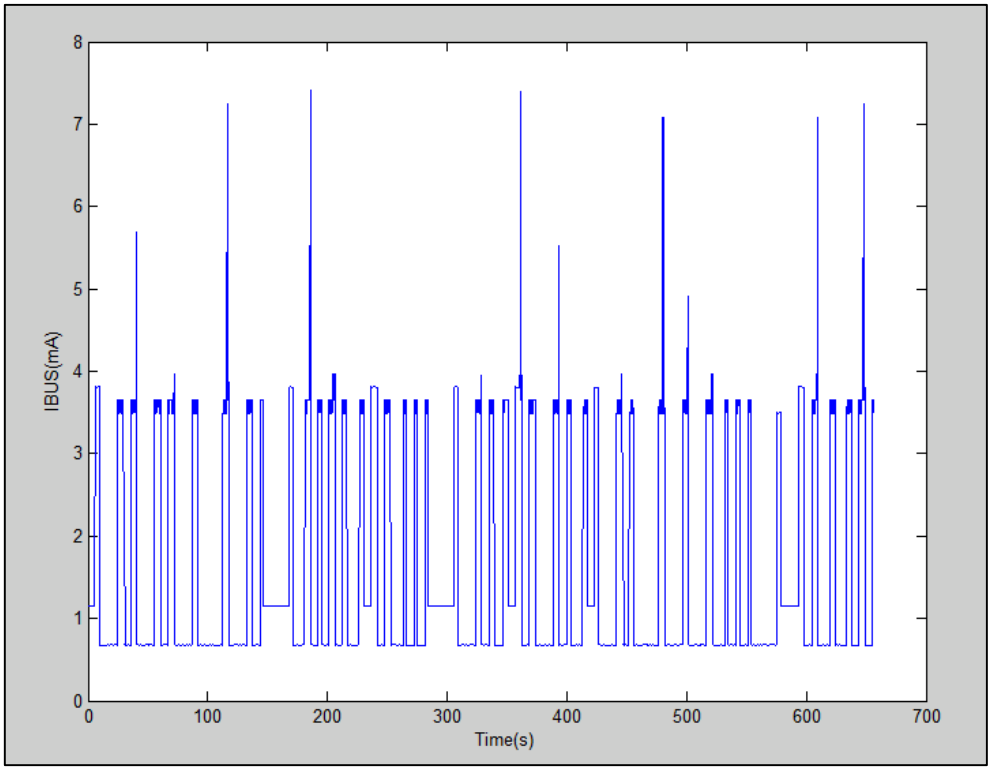

Figura 4.6. Medida de consumo 10 minutos micro

Medida de 60 minutos (Figura 4.7)

- **Montaje**: M2.
- **Tiempo:** 60 minutos.
- **Rango:** +/- 20 mV.
- **Resistencia de shunt:** 1 Ω.
- **Tiempo de muestreo:** 5ms.
- **Bits:** 8 bits.
- **Resolución:** 156.3 µV.

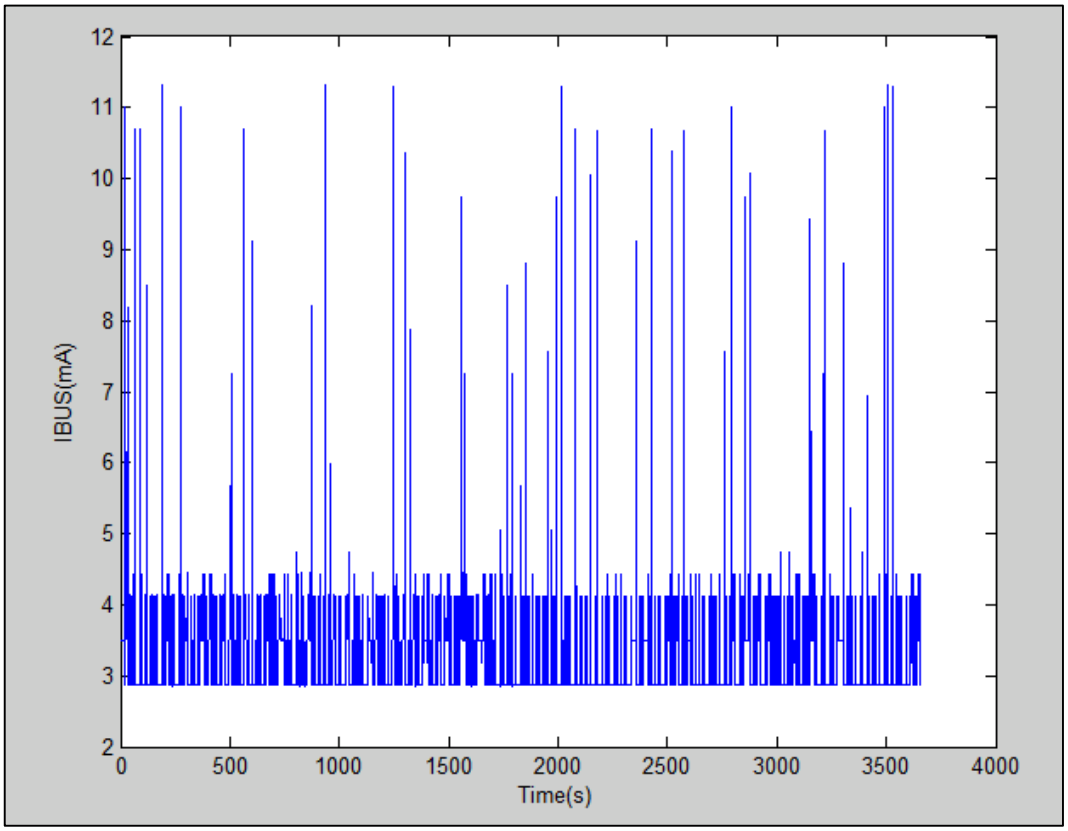

Figura 4.7. Medida de consumo 60 minutos micro

Los resultados de consumo, en comparativa a la capacidad de una pila de botón CR2450 se resumen en la Tabla 4.2.

CONSUMO DE LA PLACA			
TIEMPO MEDIDO	**CAPACIDAD CONSUMIDA (mAh)**	**CAPACIDAD RESTANTE (mAh) (C. nominal: 650 mAh)**	**SoC (%) (C. nominal: 650 mAh)**
1 MINUTO	0,052	649,948	99,992
10 MINUTOS	0,535	649,465	99,918
60 MINUTOS	2,920	647,080	99,551
CONSUMO DEL MICRO			
TIEMPO MEDIDO	**CAPACIDAD CONSUMIDA (mAh)**	**CAPACIDAD RESTANTE (mAh) (C. nominal: 650 mAh)**	**SoC (%) (C. nominal: 650 mAh)**
1 MINUTO	0,001	649,999	100,000
10 MINUTOS	0,012	649,988	99,998
60 MINUTOS	0,038	649,962	99,994

Tabla 4.2. Resumen de resultados delas medidas de consumo a 1, 10 y 60 minutos

Si tenemos en cuenta estos datos, en 1 hora de funcionamiento la pila habría gastado un 0,45% de su capacidad. En el caso de esta pila de litio, el voltaje que proporciona cae por debajo de la tensión mínima de funcionamiento de la aplicación cuando se encuentra al 20% del SoC. Por tanto teniendo en cuenta ese primer análisis, la batería duraría en torno a 80 horas. Además, hay que tener en cuenta que no se llega a medir todo el consumo real durante la comunicación, por lo que la duración real de la vida de la batería sería incluso menor.

Bajo este último punto cabe mencionar que la medida de consumo y cálculo de capacidad será más precisa cuanto más larga sea la prueba, ya que el tiempo de comunicación en relación al tiempo total será menor.

Una vez vistos los resultados, pasamos a comentar las conclusiones de este trabajo.

5. CONCLUSIONES

En este proyecto se buscaba el desarrollo de una aplicación que permita medir el consumo de energía en nodos de sensores inalámbricos, en el caso de que lo constituyan placas Atmega256rfr2 Xplained Pro.

Se han conseguido desarrollar con éxito los drivers para poder comunicar el sensor de corriente PAC1710 con el nodo, implementando las funcionalidades básicas que permiten medir la corriente con distintas configuraciones de medida. También se han programado en paralelo rutinas que permitiesen visualizar y medir los distintos modos del sueño del micro, y se ha simulado un proceso pseudo-aleatorio de cambios de estado activo, sueño y de envío de datos, intentando emular un entorno de medida más real.

El sensor PAC1710 permite conocer la tensión de la fuente, dato muy representativo en algunas baterías en las que el voltaje disminuye significativamente a medida que se reduce su capacidad, llegando incluso al FEP del sistema alimentado. El otro parámetro fundamental es la corriente administrada por la batería en cada instante de tiempo, lo que nos da su capacidad.

Sin embargo, se han encontrado complicaciones a la hora de medir el consumo durante el envío de información, debido a las limitaciones del sensor. Pese a ello, con dicho dispositivo se ha podido calcular de manera precisa la demanda de corriente durante los modos sueño y activo, en los que se prevee que el nodo pase la mayor parte del tiempo posible.

Aunque ha estado enfocado en el uso del sensor PAC1710 en la aplicación de redes de sensores inalámbricos con placas Atmega256rfr2 Xplained Pro, puede exportarse fácilmente a otros proyectos donde exista un dispositivo con comunicación I2C y entorno de desarrollo en lenguaje C, rediseñando aquellas funciones y configuraciones específicas del micro.

Por ello, aunque se ha podido comprobar que el sensor PAC1710 no es el más indicado para medir consumos muy bajos y en tiempos muy cortos por la baja frecuencia de muestreo, menor que el tiempo mínimo de cambio de la corriente durante la comunicación), este sensor podría utilizarse en dispositivos cuyo consumo no varíe tan rápido en el tiempo.

Además, analizando el consumo de la aplicación sería posible establecer qué rutinas y periféricos son los que demandan más corrientes y actuar en consecuencia para optimizar energéticamente el producto.

A continuación se hablará con más detalle de las limitaciones encontradas en el sensor:

LIMITACIONES DEL SENSOR

Se ha observado mediante todos los ensayos realizados que el sensor escogido finalmente no cumple con las características necesarias para medir el consumo durante el envío y la recepción de datos por el transceptor de radio y, en extensión, cualquier estado en el que la corriente cambie bruscamente en el orden de 5 ms o menos, pues el tiempo de muestreo es como mínimo de 2,5 ms. Además, al ser la resolución directamente proporcional al tiempo de muestreo, configurar el sensor para la mayor frecuencia de muestreo posible no es una solución práctica.

A pesar de ello, se ha conseguido aumentar la resolución utilizando una resistencia de medida menor, pues ello permite disminuir el rango.

Finalmente, en el siguiente apartado se exponen todas las posibles mejoras propuestas de este trabajo.

LÍNEAS FUTURAS DE DESARROLLO Y POSIBLES MEJORAS

La primera mejora clara de este sistema de medida de consumo es sustituir el sensor por uno cuya frecuencia de muestreo (al menos dos veces mayor que 1/t, donde t es el tiempo mínimo de cambio de la corriente) sea lo suficientemente alta, con unos rangos de tensiones y corrientes adecuados para este caso (0 a 100 mA, por ejemplo). También es importante tener una resolución lo suficientemente alta para poder obtener con precisión la señal caracterizada por la corriente demanda.

En cuanto a la parte más específica de este TFG, el desarrollo software, se proponen las siguientes mejoras:

- Optimización de los drivers
    - En cuanto a espacio en memoria. Ciertos tipos de variables, como los “float”, ocupan mucho espacio. Puede mejorarse el código para evitar declarar variables de tamaños innecesarios.
    - Legibilidad y utilidad: aunque se ha tratado de desarrollar los drivers de forma que sean los más óptimos posibles y tratando de anticiparse a posibles errores, un segundo rediseño podría mejorarlo, teniendo en cuenta la experiencia adquirida durante el proceso. Además, pueden añadirse funciones no implementadas pero que si existen en el sensor, como el cálculo de potencia y la activación de una alerta cuando se alcanzan niveles críticos de corriente. Para esta última, sin embargo, también es un requisito la comunicación SMBUS en vez de I2C.
    - En cuanto a adaptación a la aplicación: para la medida de consumo en una red real, sería necesaria una memoria externa con la que el sensor pudiese comunicarse por I2C, almacenando la información de las medidas tomadas según una rutina establecida anteriormente. Esto evitaría que el propio nodo se mantuviera ocupado cada vez que se mide la corriente, pudiendo adquirir los datos sin alterar el estado del nodo. Tras un tiempo asignado, el nodo podría leer los datos almacenados en la memoria y enviarlos al dispositivo central.
- Desarrollo de una aplicación para el usuario. No ha sido posible desarrollarla en el tiempo establecido para este proyecto. No obstante, en futuras actualizaciones, podría añadirse la visualización del consumo, tensión y potencia a través de la Shell de Visual Studio.

BIBLIOGRAFÍA

[1] The Valley. Digital Business School., "Claves Digitales 03: Internet de las cosas. El mundo hiperconectado", 2016.

[2] José Antonio Millán, "Vocabulario de ordenadores e Internet," 2003. [Online]. Available: http://jamillan.com/v_nodo.htm. [Accessed: 23-Sep-2017].

[3] IEEE Computer Society. LAN/MAN Standards Committee., Institute of Electrical and Electronics Engineers., and IEEE-SA Standards Board., "IEEE standard for information technology : telecommunications and information exchange between systems--local and metropolitan area networks--specific requirements. Part 15.4, Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (WPANs)" 2006. [Online]. Available: http://www.ie.itcr.ac.cr/acotoc/Ingenieria/Lab%20TEM%20II/Antenas/Especificacion%20802%2011-2007.pdf [Accessed: 15-Aug-2017].

[4] U. Amozarrain, Trabajo de Fin de Máster"Low Power WiFi: A study on power consumption for Internet of Things," Master in Innovation and Research in Informatics Computer Networks and Distributed Systems. Universidad Politécnica de Cataluña, Febrero 2015. [Online]. Available: http://upcommons.upc.edu/bitstream/handle/2099.1/25583/104901.pdf?sequence=1 [Accessed: 15-Aug-2017].

[5] "Application note avr2130: Lightweight Mesh Developer Guide. Atmel MCU Wireless" 2014. [Online]. Available: http://www.atmel.com/Images/Atmel-42028-Lightweight-Mesh-Developer-Guide_Application-Note_AVR2130.pdf [Accessed: 15-Aug-2017].

[6] "Application note avr2131: Lightweight Mesh Getting Started Guide AVR2131: Lightweight Mesh Getting Started Guide, Atmel MCU Wireless"2014. [Online]. Available: http://www.atmel.com/Images/Atmel-42029-Lightweight-Mesh-Getting-Started-Guide_Application-Note_AVR2131.pdf [Accessed: 27-Aug-2017].

[7] "Powering IoT Devices: Technologies and Opportunities - IEEE Internet of Things."2015. [Online]. Available: https://iot.ieee.org/newsletter/november-2015/powering-iot-devices-technologies-and-opportunities.html [Accessed: 27-Aug-2017].

[8] "Tendencias de diseño: baterías para Internet de las Cosas (IoT) - Electrónica," 2016. [Online]. Available: http://www.interempresas.net/Electronica/Articulos/171251-Tendencias-de-diseno-baterias-para-Internet-de-las-Cosas-(IoT).html [Accessed: 27-Aug-2017].

[9] Zach Wendt,"5 Essential Factors for Choosing the Right Battery" https://www.arrow.com/en/research-and-events/articles/choosing-the-right-battery-for-your-internet-of-things-application [Accessed: 27-Aug-2017].

[10] "Las 20 preguntas más frecuentes sobre baterías - TAB," 2017. [Online]. Available: http://www.tabspain.com/corporativo/preguntas-mas-frecuentes-sobre-baterias/#p9. [Accessed: 27-Aug-2017].

[11] Arrow.com," 2017. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/choosing-the-right-battery-for-your-internet-of-things-application?dclid=CIC_npS3_9QCFRdDDAodbz8CBA. [Accessed: 27-Aug-2017].

[12] José Rafael Lajara Vizcaíno. Tesis Doctoral, "Modelado y optimización de energía en redes de sensores inalámbricas para la medida de parámetros medioambientales," Universidad Politécnica de Valencia, Junio 2014. [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/39371/Lajara%20-%20Modelado%20y%20optimizaci%C3%B3n%20de%20energ%C3%ADa%20en%20redes%20de%20sensores%20inal%C3%A1mbricas%20para%20la%20medida%20de%20p....pdf?sequence=1 [Accessed: 15-Aug-2017].

[13] K. Furset and P. Hoffman, "High pulse drain impact on CR2032 coin cell battery capacity," 2011. [Online]. Available: https://www.dmcinfo.com/Portals/0/Blog%20Files/High%20pulse%20drain%20impact%20on%20CR2032%20coin%20cell%20battery%20capacity.pdf [Accessed: 15-Aug-2017].

[14]] N. Jay Tyzzer, Senior Field Application Engineer- Americas, "Extending battery life in ultra low power wireless applications," 2013. [Online]. Available: http://www.low-powerdesign.com/121312-article-extending-battery-life.htm. [Accessed: 27-Aug-2017].

[15] Z. Abbas and W. Yoon, "A Survey on Energy Conserving Mechanisms for the Internet of Things: Wireless Networking Aspects," Sensors, vol. 15, no. 10, pp. 24818–24847, Sep. 2015.

[16] S. Kamath and J. L. Lindh, "Application Note AN092. Measuring Bluetooth® Low Energy Power Consumption." [Online]. Available: http://www.ti.com/lit/an/swra347a/swra347a.pdf . [Accessed: 27-Aug-2017].

[17] "Bluetooth Sleep Modes," 2009. [Online]. Available: http://www.althos.com/tutorial/bluetooth-tutorial-sleep-modes.html. [Accessed: 27-Aug-2017].

[18] M. Lorenzo, "testing, analizing and modelling of lithium-ion batteries," 2016. [Online]. Available:

[19] Óscar Escobar Muñoz, "Diseño Hardware de un Sistema Eficiente para la Medida del Consumo de Energía en Nodos Inalámbricos de Redes de Sensores," Grado en Ingeniería Electrónica Industrial y Automática. Universidad Carlos III de Madrid. Septiembre 2017. [Online]. Available:

[20] Fernando Cuadri Carvajo, Tesis Doctoral "estimación del estado de carga para toyota prius: coulomb counting,"Unversidad Politécnica de Sevilla, Abril 2010. [Online]. Available: http://bibing.us.es/proyectos/abreproy/11872/fichero/Memoria+(Castellano_e_ingles)%252FResumen_castellano%252Fcapitulo1.pdf [Accessed: 26-Sep-2017].

[21] U. Agarwal, K. Patel, and P. Reddy, "AN92584 Designing for Low Power and Estimating Battery Life for BLE Applications Santosh S Associated Project: Yes Associated Part Family: CY8C4XX7-BL, CY8C4XX8-BL, CYBL1XX6X, CYBL1XX7X Software Version: PSoC ® Creator™ 3.3," 2017. [Online]. Available: https://www.google.es/search?q=AN92584+Designing+for+Low+Power+and+Estimat&oq=AN92584+Designing [Accessed: 26-Sep-2017].

[22] David Ferrer Alayeto, Trabajo Fin de Grado"Circuito para la medición de la carga y la salud de baterías," Universidad Politécnica de Cataluña, Sep. 2011. [Online]. Available: http://upcommons.upc.edu/handle/2099.1/13294 [Accessed: 26-Sep-2017].

[23] M. Jensen, "Coin cells and peak current draw."2010. [Online]. Available: http://www.ti.com/lit/wp/swra349/swra349.pdf [Accessed: 26-Sep-2017].

[24] Atmel Corporation, "Microcontroller Atmel AVR ATmega256RFR2 Xplained Pro User Guide," 2015. [Online]. Available: http://www.atmel.com/Images/Atmel-8393-MCU_Wireless-ATmega256RFR2-ATmega128RFR2-ATmega64RFR2_Datasheet.pdf [Accessed: 26-Sep-2017].

[25] Atmel Corporation, "ATMEGA256RFR2 Xplained Pro design documentation release rev3," 2014. [Online]. Available: http://www.atmel.com/webdoc/mega256rfr2xplainedpro/mega256rfr2xplainedpro.GettingStarted.Documentation.html [Accessed: 26-Sep-2017].

[26] Atmel Corporation, "ATmega256/128/64RFRS Microcontroller with Low Power 2.4GHz Transceiver," 2014. [Online]. Available: http://www.atmel.com/Images/Atmel-8393-MCU_Wireless-ATmega256RFR2-ATmega128RFR2-ATmega64RFR2_Datasheet.pdf [Accessed: 26-Sep-2017].

[27] Atmel Corporation, "Software Atmel Studio USER GUIDE." 2016. [Online]. Available:

[28] A. López, "Arquitecturas CISC y RISC – Arquitectura de computadoras UNAH," 2016. [Online]. Available: https://allanlopezunah.wordpress.com/2016/06/04/arquitecturas-cisc-y-risc/. [Accessed: 04-Sep-2017]]

[29] Paola Andrea Rojas Contreras, "WATCHDOG TIMER," 2004. [Online]. Available: http://www2.elo.utfsm.cl/~lsb/elo325/clases/WATCHDOG%20TIMER.pdf

[30] R. Pindado, "Phase Locked-Loop (PLL): Fundamento y aplicaciones." [Online]. Available: http://www.jcee.upc.es/JCEE2001/PDFs2001/pindado.pdf [Accessed: 26-Sep-2017].

[31] Atmel Corporation, "ATmega256RFR2 - Wireless Modules." [Online]. http://www.microchip.com/wwwproducts/en/ATmega256rfr2. [Accessed: 29-Aug-2017].

[32] Atmel Corporation, "AVR JTAG ICE User Guide," 2001. [Online]. Available: http://www.atmel.com/images/doc2475.pdf [Accessed: 26-Sep-2017].

[33] Microchip Technology Inc., "Pac1710/20 Single and Dual High-Side Current-Sense Monitor with Power Calculation," 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/20005386A.pdf [Accessed: 26-Sep-2017].

[34] P. M. Gómez, "I2C." [Online]. Available: http://www.sase.com.ar/2013/files/2013/09/SASE2013-I2C-P-Gomez.pdf [Accessed: 23-Sep-2017].

[35] J. M. Irazabal and S. Blozis, "I2C Bus Overview," Des. TecForum, 2003. https://www.nxp.com/docs/en/supporting-information/design_con_2003_tecforum_i2c_bus_overview.pdf [Accessed: 22-Sep-2017].

[36] Microchip Technology Inc., "PAC1710 and PAC1720 High-Side Current Sensors Evaluation Board User's Guide," 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/50002367B.pdf [Accessed: 22-Sep-2017].

[37] "IEEE Standard Test Access Port and Boundary-Scan Architecture IEEE-SA Standards Board," 2001. [Online]. Available: http://fiona.dmcs.pl/~cmaj/JTAG/JTAG_IEEE-Std-1149.1-2001.pdf [Accessed: 22-Sep-2017].

[38] José Ángel Miranda Calero. Trabajo de Fin de Máster "Wireless Network Sensor Design and Development for secure enviroments". Máster en Sistemas Electrónicos y Aplicaciones. Universidad Carlos III de Madrid. Septiembre 2016.

[39] Ismael Granados Llorente. Trabajo Fin de Grado "Desarrollo e implementación de drivers para aplicación IoT. Diseño y desarrollo de red de sensores para seguridad basada en IEEE802.15.4". Grado en Ingeniería Electrónica Industrial y Automática. Universidad Carlos III de Madrid. Septiembre 2016.

[40] Energizer Holdings, "1-800-383-7323 USA/CAN ENERGIZER CR2450." [Online]. Available: http://data.energizer.com/pdfs/cr2450.pdf [Accessed: 22-Sep-2017].

[41] "ASF Source Code Documentation." [Online]. Available: http://asf.atmel.com/docs/latest/mega/html/group__related__projects.html. [Accessed: 01-Sep-2017]

[42] N. Lichtmaier, "Normas de programación en C/C++." [Online]. Available: http://www.reloco.com.ar/prog/normas.html. [Accessed: 26-Sep-2017].

[43] Universitat Politècnica de Catalunya, "Reglas básicas de la programación en lenguaje C." [Online]. Available: [Accessed: 26-Sep-2017].http://extropynow.weebly.com/uploads/1/6/4/1/16411724/manualbasico_referenciaparac.pdf. [Accessed: 26-Sep-2017].

ANEXO

A1. CÓDIGOS ELABORADOS DURANTE EL DESARROLLO DEL PROYECTO

a. DRIVERS PAC1710

PAC1710.H

```
/**
 * \file PAC1710.h
 *
 * \brief PAC1710 Sensor Driver Support Header File \brief PAC1710 Current and
Voltage Sensor
 *
 * $Id: PAC1710.h Iván Garrido Vega $
 */

/* Only available for ATMEGA256RFR2 */
#ifdef PLATFORM_XPLAINED_PRO_ATMEGA256RFR2

#ifndef __PAC1710_H__
#define __PAC1710_H__

#include "stdint.h"
#include "sysTimer.h"

/*****************************************************************/
/**\name                          MACROS
            */
/*****************************************************************/
/*****************************************************************/
/**\name      GET AND SET REGISTERS BITS VALUES FUNCTIONS        */
/*****************************************************************/
//PAC1710_GET_REG_BITS_VALUE
//Get only the bits determined by bitname##__POS and bitname##__MSK of the
register determined by regvar
#define PAC1710_GET_REG_BITS_VALUE(regvar, bitname)((regvar & bitname##__MSK) >>
bitname##__POS)

//PAC1710_SET_REG_BITS_VALUE
//Set only the bits determined by bitname##__POS and bitname##__MSK to val of the
register determined by regvar
#define PAC1710_SET_REG_BITS_VALUE(regvar, bitname, val)((regvar &
~bitname##__MSK) | ((val<<bitname##__POS)&bitname##__MSK))

/*****************************************************************/
/**\name      COMMON USED CONSTANTS
       */
/*****************************************************************/
/*************************************************/
/**\name      MODE SELECTION VARIABLES  */
/*************************************************/
#define PAC1710_STANDBY_MODE  (3) //BITS 1-0. Set standby mode
#define PAC1710_V_MODE  (2)             //BITS 1-0. Set Vsource measure   mode
#define PAC1710_I_MODE  (1)             //BITS 1-0. Set Vsense measure mode
#define PAC1710_IV_MODE (0)             //BITS 1-0. Set Vsource-Vsense measure
mode

/*************************************************/
/**\name      CONVERSION RATE VARIABLES  */
/*************************************************/
#define PAC1710_CONV_RATE_1SEC    (0) // BITS 1-0. 1s conversion rate
#define PAC1710_CONV_RATE_2SEC    (1)    // BITS 1-0. 2s conversion rate
#define PAC1710_CONV_RATE_4SEC    (2) // BITS 1-0. 4s conversion rate
```

```
#define PAC1710_CONV_RATE_CONT    (3) // BITS 1-0. Continuous conversion rate
(DEFAULT)

/*************************************************/
/**\name      VSOURCE TIME VARIABLES  */
/*************************************************/
#define PAC1710_VSOURCE_TIME_2F5 (0)//BITS 3-2. Vsource sample time is 2.5ms
#define PAC1710_VSOURCE_TIME_5    (1)    //BITS 3-2. Vsource sample time is 5ms
#define PAC1710_VSOURCE_TIME_10 (2) //BITS 3-2. Vsource sample time is 10ms
#define PAC1710_VSOURCE_TIME_20 (3) //BITS 3-2. Vsource sample time is 20ms
(DEFAULT)

/*************************************************/
/**\name      VSOURCE AVERAGE VARIABLES  */
/*************************************************/
#define PAC1710_VSOURCE_AVR_OFF (0)      //BITS 1-0.   Vsource average is off
(DEFAULT)
#define PAC1710_VSOURCE_AVR_2     (1)    //BITS 1-0. Vsource average is 2
#define PAC1710_VSOURCE_AVR_4     (2) //BITS 1-0. Vsource average is 4
#define PAC1710_VSOURCE_AVR_8     (3) //BITS 1-0. Vsource average is 8

/*************************************************/
/**\name      VSENSE TIME VARIABLES                                    */
/*************************************************/
#define PAC1710_VSENSE_TIME_2F5 (0) //BITS 6-4. Vsense sample time is 2.5ms
#define PAC1710_VSENSE_TIME_5     (1) //BITS 6-4. Vsense sample time is 5ms
#define PAC1710_VSENSE_TIME_10  (2) //BITS 6-4. Vsense sample time is 10ms
#define PAC1710_VSENSE_TIME_20  (3) //BITS 6-4. Vsense sample time is 20ms
#define PAC1710_VSENSE_TIME_40  (4) //BITS 6-4. Vsense sample time is 40ms
#define PAC1710_VSENSE_TIME_80  (5) //BITS 6-4. Vsense sample time is 80ms
(DEFAULT)
#define PAC1710_VSENSE_TIME_160 (6) //BITS 6-4. Vsense sample time is 160ms
#define PAC1710_VSENSE_TIME_320 (7) //BITS 6-4. Vsense sample time is 320ms

/*************************************************/
/**\name      VSENSE AVERAGE VARIABLES                        */
/*************************************************/
#define PAC1710_VSENSE_AVR_OFF    (0) //BITS 3-2. Vsense average is off (DEFAULT)
#define PAC1710_VSENSE_AVR_2      (1) //BITS 3-2. Vsense average is 2
#define PAC1710_VSENSE_AVR_4      (2) //BITS 3-2. Vsense average is 4
#define PAC1710_VSENSE_AVR_8      (3) //BITS 3-2. Vsense average is 8

/*************************************************/
/**\name      VSENSE RANGE VARIABLES  */
/*************************************************/
#define PAC1710_VSENSE_RANGE_10   (0) //BITS 1-0. Vsense range is +/-10mV
#define PAC1710_VSENSE_RANGE_20   (1) //BITS 1-0. Vsense range is +/-20mV
#define PAC1710_VSENSE_RANGE_40   (2) //BITS 1-0. Vsense range is +/-40mV
#define PAC1710_VSENSE_RANGE_80   (3) //BITS 1-0. Vsense range is +/-80mV
(DEFAULT)

/*************************************************/
/**\name      I2C ADDRESS DEFINITIONS                          */
/*************************************************/
#define       I2C_BUFFER_LEN      8
//Buffer lenght
#define PAC1710_DEV_ADDR (0x98)                          //Device Address
(0x9A) for PCB design (0x98) for evaluation board
#define PAC1710_SLAVE_ADDR  (PAC1710_DEV_ADDR)   // Slave Address

/*************************************************/
/**\name                  DATA SIZE                                */
```

```
/*************************************************/
#define PAC1710_WR_DATA_N_BYTES   (1)                 //Size of PAC1710 registers
is 8 bits
#define PAC1710_RESULTS_DATA_N_BYTES (2) //Vsense, Vsource and Power are saved in
2 registers of 8 bits
#define PAC1710_V_I_DATA_N_BITS (16)
#define PAC1710_REGISTERS_N_BITS (8)

/*************************************************/
/**\name                    BIT SHIFTING                                */
/*************************************************/
#define PAC1710_SHIFT_BIT_POSITION_BY_02_BITS   (2)
#define PAC1710_SHIFT_BIT_POSITION_BY_04_BITS   (4)
#define PAC1710_SHIFT_BIT_POSITION_BY_06_BITS   (6)

/*************************************************/
/**\name      ERROR CODE DEFINITIONS                                   */
/*************************************************/
#define       PAC_SUCCESS   ((uint8_t)1)
#define PAC_ERROR    ((int8_t)-1)
#define E_PAC1710_NULL_PTR  ((int8_t)-2)



/*************************************************/
/**\name      OTHER CONSTANTS                                                */
/*************************************************/
#define PAC1710_INIT_VALUE (0)
#define PAC1710_NULL (0)


/*************************************************/
/**\name      REGISTER ADDRESS DEFINITIONS  */
/*************************************************/
#define PAC1710_PRODUCT_ID_REG  (0xFD)          //Product identity address
       #define PAC1710_PRODUCT_ID_VAL  (0x57)  //Product identity value. (0x58)
for PAC1710. (0X57) for PAC17120
#define MANUFACTURER_ID_REG       (0xFE)        //Manufacture identity address
       #define MANUFACTURER_ID_VAL       (0x5D) //Manufacture identity value
#define REVISION_REG              (0xFF)        //Revision address
       #define REVISION_REG_VAL          (0x81) //Revision value
//-----------------------------------------------
#define PAC1710_MODE_ADDR         (0x00) //Mode configuration address
#define PAC1710_CONV_RATE_ADDR    (0x01) //Conversion rate configuration address
#define PAC1710_ONE_SHOT_ADDR     (0x02) //One-shot address
//#define PAC1710_MASK_ADDR       (0x03)  //Mask Alert configuration address
(Unused for I2C communication)
#define PAC1710_HLS_ADDR          (0x04) //Hight Limit Status configuration
address (Used only for conversion rate status bit)
//#define PAC1710_LLS_ADDR        (0x05) //Low Limit Status configuration address
(Unused for I2C communication)
#define PAC1710_VSOURCE_ADDR      (0x0A) //Vsource configuration address
#define PAC1710_VSENSE_ADDR       (0x0B) //Vsense configuration address
#define PAC1710_VSENSE_HB_ADDR  (0x0D)   //Vsense Hight Byte data address
#define PAC1710_VSENSE_LB_ADDR    (0x0E) //Vsense Low Byte data address
#define PAC1710_VSOURCE_HB_ADDR (0x11)   //Vsource Hight Byte data address
#define PAC1710_VSOURCE_LB_ADDR (0x12)   //Vsource Low Byte data address
//#define PAC1710_POWER_HB_ADDR   (0x15) //Power Hight Byte data address (Unused)
//#define PAC1710_POWER_LB_ADDR   (0x16) //Power Low Byte data address (Unused)

/*************************************************/
/**\name      BIT MASK, LENGTH AND POSITION DEFINITIONS  */
```

```
/*****************************************************/
// For PAC1710_GET_REG_BITS_VALUE and PAC1710_SET_REG_BITS_VALUE
/* MODE */
//MASK BIT (unused)
/*
#define PAC1710_MASKBIT__POS (5)                                        //Bit
position
#define PAC1710_MASKBIT__MSK (0x20)
       //Mask for bit 5
#define PAC1710_MASKBIT__LEN (PAC1710_REGISTERS_N_BITS)             //Address
length
#define PAC1710_MASKBIT__REG (PAC1710_MODE_ADDR)                    //Mode
configuration address
*/

//STANDBY
#define PAC1710_MODE__POS (0)                                           //Bit
position
#define PAC1710_MODE__MSK (0x03)                                    //Mask for
bits 0 and 1
#define PAC1710_MODE__LEN (PAC1710_REGISTERS_N_BITS)  //Address length
#define PAC1710_MODE__REG (PAC1710_MODE_ADDR)               //Mode configuration
address

//V MODE
#define PAC1710_V_PIN__POS (0)                                          //Bit
position
#define PAC1710_V_PIN__MSK (0x01)                                   //Mask for
bit 0
#define PAC1710_V_PIN__LEN (PAC1710_REGISTERS_N_BITS) //Address length
#define PAC1710_V_PIN__REG (PAC1710_MODE_ADDR)              //Mode configuration
address

//I MODE
#define PAC1710_I_PIN__POS (1)                                          //Bit
position
#define PAC1710_I_PIN__MSK (0x02)                                   //Mask for
bit 1
#define PAC1710_I_PIN__LEN (PAC1710_REGISTERS_N_BITS) //Address length
#define PAC1710_I_PIN__REG (PAC1710_MODE_ADDR)              //Mode configuration
address

/* CONV_RATE*/
#define PAC1710_CONV_RATE__POS (0)
       //Bit position
#define PAC1710_CONV_RATE__MSK (0x03)                                   //Mask
for bits 0 and 1
#define PAC1710_CONV_RATE__LEN (PAC1710_REGISTERS_N_BITS)    //Address length
#define PAC1710_CONV_RATE__REG  (PAC1710_CONV_RATE_ADDR)     //Conversion rate
configuration address

/*PAC1710_HIGHT_LIMIT_STATUS*/
#define PAC1710_HLS__POS (7)                                        //Bit
position
#define PAC1710_HLS__MSK (0x80)                                     //Mask for
bit 7
#define PAC1710_HLS__LEN (PAC1710_REGISTERS_N_BITS)   //Address length
#define PAC1710_HLS__REG (PAC1710_HLS_ADDR)                 //Hight Limit Status
address

/* PAC1710_VSOURCE_TIME*/
```

```
#define PAC1710_VSOURCE_TIME__POS (2)                                    //Bit
position
#define PAC1710_VSOURCE_TIME__MSK (0x0C)                              //Mask for
bits 3 and 4
#define PAC1710_VSOURCE_TIME__LEN (PAC1710_REGISTERS_N_BITS)//Address length
#define PAC1710_VSOURCE_TIME__REG (PAC1710_VSOURCE_ADDR)     //Vsource address

/* PAC1710_VSOURCE_AVERAGE*/
#define PAC1710_VSOURCE_AVERAGE__POS (0)                                 //Bit
position
#define PAC1710_VSOURCE_AVERAGE__MSK (0x03)
       //Mask for bits 0 and 1
#define PAC1710_VSOURCE_AVERAGE__LEN (PAC1710_REGISTERS_N_BITS) //Address length
#define PAC1710_VSOURCE_AVERAGE__REG (PAC1710_VSOURCE_ADDR)         //Vsource
address

/* PAC1710_VSENSE_SAMPLING_TIME */
#define PAC1710_VSENSE_SAMPLING_TIME__POS (4)
       //Bit position
#define PAC1710_VSENSE_SAMPLING_TIME__MSK (0x70)
       //Mask for bits 4,5 and 6
#define PAC1710_VSENSE_SAMPLING_TIME__LEN (PAC1710_REGISTERS_N_BITS)
       //Address length
#define PAC1710_VSENSE_SAMPLING_TIME__REG (PAC1710_VSENSE_ADDR)
       //Vsense address

/* PAC1710_VSENSE_AVERAGE */
#define PAC1710_VSENSE_AVERAGE__POS (2)
       //Bit position
#define PAC1710_VSENSE_AVERAGE__MSK (0x0C)
       //Mask for bits 3 and 4
#define PAC1710_VSENSE_AVERAGE__LEN (PAC1710_REGISTERS_N_BITS)      //Address
length
#define PAC1710_VSENSE_AVERAGE__REG (PAC1710_VSENSE_ADDR)           //Vsense
address

/* PAC1710_VSENSE_RANGE */
#define PAC1710_VSENSE_RANGE__POS (0)
       //Bit position
#define PAC1710_VSENSE_RANGE__MSK (0x03)                                 //Mask
for bits 0 and 1
#define PAC1710_VSENSE_RANGE__LEN (PAC1710_REGISTERS_N_BITS) //Address length
#define PAC1710_VSENSE_RANGE__REG (PAC1710_VSENSE_ADDR)
       //Vsense address

/*************************************************************/
/**\name                                STRUCT                          */
/*************************************************************/

struct PAC1710_T {
       //Values of configuration
       uint8_t mode;              //Mode data
       uint8_t conv_rate;         //Conversion rate data
       uint8_t conv_done;         //Conversion done bit, 0 while conv_rate is
running
       uint8_t vsource_avr; //Vsource average data
       uint8_t vsource_stime;     //Vsource sample time data
       uint8_t vsense_stime;      //Vsense sample time data
       uint8_t vsense_avr;        //Vsense average data
       uint8_t vsense_rang; //Vsense range data
       uint8_t chip_id;           //chip id of the sensor
       uint8_t dev_addr;          //device address of the sensor
```

```
	uint8_t config_reg;		//status of configuration register
	uint8_t flag;			//Flag status sensor

/*Voltage and current data (of registers)*/
	uint16_t voltage_vsource;  /* voltage over sampling*/
	int16_t current_vsense;		/* current over sampling*/
	uint16_t vsource_stime_10ms;/* vsource sample time in ms*10*/
	uint16_t vsense_stime_10ms;/* vsense sample time in ms*10*/

/*Voltage and current results*/
	float vsource_pin;  /* voltage result*/
	float ibus;			/* current result*/

/*Measure variables*/
	uint16_t measure_time_ms;		/*measure time in ms*/
	uint16_t measure_time_s;		/*measure time in s*/
	uint16_t num_measures;			/*number of measures*/
	uint16_t voltage_reg_values[100]; /*Vsource register data array*/
	float voltage_real_values[100];		/*Vsource real value array*/
	int16_t current_reg_values[100];  /*Vsense register data array*/
	float current_real_values[100];		/*Vsense real value array*/
	float time_values[100];			/*time data array*/
	uint8_t counter1;			/*Counter*/

/*Rshunt*/
	float Rshunt; /*Rsense value*/

/*Status variable*/
	int8_t status_com;

};
extern struct PAC1710_T g_PAC1710;


/*************************************************************/
/**\name	FUNCTIONS DECLARATIONS
*/
/*************************************************************/
/*************************************************************/
/**\name	FUNCTION FOR  INTIALIZATION			*/
/*************************************************************/
/*!
 *	@brief This function is used for initialize
 *	the PAC1710 sensor
 *	@param R_shunt: resistance you are using
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval <0 -> Error
*/
int8_t PAC1710_init(float R_shunt);

/*************************************************************/
/**\name	FUNCTIONS FOR POWER MODE*/
/*************************************************************/
/*!
 *	@brief This API used to get the
 *	Operational Mode from the sensor in the register 0x00 bit 0 and 1
 *	@param v_mode : The value of power mode
 *  value of 1-0 bits|    mode
 *  -----------------|-----------------
 *		00		| PAC1710_IV_MODE
 *		01			| PAC1710_I_MODE
```

```
 *           10                    | PAC1710_V_MODE
 *           11           | PAC1710_STANDBY_MODE
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_get_mode(uint8_t *v_mode);
/*!
 *     @brief This API used to set the
 *     Operational Mode from the sensor in the register 0x00 bit 0 and 1
 *     @param v_mode: The value of power mode
 *  value of 1-0 bits|    mode
 *  -----------------|------------------
 *           00           | PAC1710_ACTIVE_IV_MODE
 *           01                    | PAC1710_ACTIVE_I_MODE
 *           10                    | PAC1710_ACTIVE_V_MODE
 *           11           | PAC1710_STANDBY_MODE
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_set_mode(uint8_t v_mode);
/*!
 *     @brief This API used to set the
 *     One Shot Mode from the sensor in the register 0x02
 *     when the sensor is operating in Standby Mode
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
*/
int8_t PAC1710_one_shot_mode(void);

/*************************************************************/
/**\name      FUNCTIONS FOR CONVERSION RATE */
/*************************************************************/
/*!
 *     @brief This API used to set the
 *     Conversion rate parameters from the sensor in the register 0x1H bit 0 to 1
 *     @param v_conv_rate: The value of conversion rate register
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
 *     CONVERSION_RATE SETTINGS
 Bits 1-0
 *     1 per second: 00
 *     2 per second: 01
 *     4 per second: 10
 *     Continuous: 11 (default)
 *
 */

int8_t PAC1710_get_conv_rate(uint8_t *v_conv_rate);
/*!
 *     @brief This API used to set the
 *     Conversion rate parameters from the sensor in the register 0x1H bit 0 to 1
 *     @param v_conv_rate: The value of conversion rate register
```

```
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
 *     CONVERSION_RATE SETTINGS
 Bits 1-0
 *     1 per second: 00
 *     2 per second: 01
 *     4 per second: 10
 *     Continuous: 11 (default)
 *
 */
int8_t PAC1710_set_conv_rate(uint8_t v_conv_rate);

/*!
 *     @brief This API used to read de bit 7 of the Hight Limit Status Register
(0x04)
 *     that determines if the conv_rate is finished ('1') or not ('0')
 *     @param v_conv_done_bit: The variable where the value of conv_done bit will
be saved
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
 */
int8_t PAC1710_read_conv_done_bit(void);

/*************************************************************/
/**\name      FUNCTIONS FOR VSOURCE*/
/*************************************************************/
/*************************************************************/
/**\name      VSOURCE SAMPLING TIME SETTINGS*/
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     vsource sampling settings parameters from the sensor in the register 0xAH
bits 2 to 3
 *     @param v_vsource32: The value of bits 3-2 of vsource register
 *
 *     VOLTAGE-SAMPLING TIME SETTINGS
 *     bit 3-2
 *  2.5 ms (data=8bits: 00
 *  5ms (data=9bits): 01
 *  10ms (data=10bits): 10 (default)
 *  20ms(data=11bits): 11
 *
 *     @return results of bus communication function
 *     @retval 1-> Success
 *     @retval -1 -> Error
 *
 *
*/
int8_t PAC1710_get_vsource_samp_time(uint8_t *v_vsource32);
/*!
*     @brief This API used to set the
*     vsource sampling settings parameters from the sensor in the register 0xAH
bits 2 to 3
*
*     @param v_vsource32: The value of bits 3-2 of vsource register
*
```

```
*      VOLTAGE-SAMPLING TIME SETTINGS
*      bit 3-2
*      2.5 ms (data=8bits: 00
*      5ms (data=9bits): 01
*      10ms (data=10bits): 10 (default)
*      20ms(data=11bits): 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> Error
*
*
*/
int8_t PAC1710_set_vsource_samp_time(uint8_t v_vsource32);

/*************************************************************/
/**\name      VSOURCE AVERAGE SETTINGS*/
/*************************************************************/
/*!
*      @brief This API used to get the vsource average settings parameters
*      from the sensor in the register 0xAH bits 0 to 1
*      @param v_vsource10: The value of bits 1-0 of vsource register
*
*      VOLTAGE-AVERAGE SETTINGS
*      Bits 1-0
*      Deactivated: 00 (default)
*      2: 01
*      4: 10
*      8: 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> Error
*
*/
int8_t PAC1710_get_vsource_avr(uint8_t *v_vsource10);

/*!
*      @brief This API used to set the vsource average settings parameters
*      from the sensor in the register 0xAH bits 0 to 1
*      @param v_vsource10: The value of bits 1-0 of vsource register
*
*      VOLTAGE-AVERAGE SETTINGS
*      Bits 1-0
*      Deactivated: 00 (default)
*      2: 01
*      4: 10
*      8: 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> Error
*/
int8_t PAC1710_set_vsource_avr(uint8_t v_vsource10);

/*************************************************************/
/**\name      FUNCTIONS FOR VSENSE                                    */
/*************************************************************/
/*************************************************************/
/**\name      FUNCTION FOR CURRENT SAMPLING TIME                      */
/*************************************************************/
/*!
```

```
 *     @brief This API used to get the
 *     vsense parameters from the sensor in the register 0x0B bit 4 to 6
 *
 *     @param v_vsense64: The value of vsense register bits 6-4
 *
 *     CURRENT-SENSING SAMPLING TIME
 *     bit 7-> Unused
 *     bits 6-4
 *  2.5ms: 000
 *  5ms: 001
 *     10ms: 010
 *     20ms: 011
 *     40ms: 100
 *     80ms: 101 (default)
 *     160ms: 110
 *     320ms: 111
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
*/
int8_t PAC1710_get_vsense_stime(uint8_t *v_vsense64);
/*!
 *     @brief This API used to set the
 *     vsense parameters from the sensor in the register 0x0B bits 4 to 6
 *     @param v_vsense64: The value of vsense register bits 6-4
 *     CURRENT-SENSING SAMPLING TIME
 *     bit 7-> Unused
 *     bits 6-4
 *  2.5ms: 000
 *  5ms: 001
 *     10ms: 010
 *     20ms: 011
 *     40ms: 100
 *     80ms: 101 (default)
 *     160ms: 110
 *     320ms: 111
 *     @return results of bus communication function
 *     @retval 1-> Success
 *     @retval -1 -> Error
*/
int8_t PAC1710_set_vsense_stime(uint8_t v_vsense64);

/*************************************************************/
/**\name      FUNCTION FOR CURRENT SENSING AVERAGING*/
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     vsense sensing averaging parameters from the sensor in the register 0x0B
bit 2 to 3
 *
 *     @param v_vsense32: The value of vsense register bits 3-2
 *
 *     CURRENT-SENSING AVERAGING SETTINGS
Bits 3-2
 *     Deactivated: 00 (default)
 *     2: 01
 *     4: 10
 *     8: 11
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
```

```
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_get_vsense_avr(uint8_t *v_vsense32);
/*!
*      @brief This API used to set the
*      vsense sensing averaging parameters from the sensor in the register 0x0B
bit 2 to 3
*
*      @param v_vsense32: The value of vsense register bits 3-2
*
*      CURRENT-SENSING AVERAGING SETTINGS
Bits 3-2
*      Deactivated: 00 (default)
*      2: 01
*      4: 10
*      8: 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> Error
*
*/
int8_t PAC1710_set_vsense_avr(uint8_t v_vsense32);

/*************************************************************/
/**\name      FUNCTION FOR CURRENT SENSING RANGE*/
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     vsense sensing averaging parameters from the sensor in the register 0x0B
bit 0 to 1
 *
 *     @param v_vsense10: The value of vsense register bits 1-0
 *
 *     CURRENT-SENSING RANGE SETTINGS
 *  Bits 1-0
 *     -10mV a +10mV: 00
 *     -20mV a +20mV: 01
 *     -40mV a +40mV: 10
 *     -80mV a +80mV: 11 (default)
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_get_vsense_rang(uint8_t *v_vsense10);
/*!
 *     @brief This API used to set the
 *     vsense sensing averaging parameters from the sensor in the register 0x0B
bit 0 to 1
 *
 *     @param v_vsense10: The value of vsense register bits 1-0
 *
 *     CURRENT-SENSING RANGE SETTINGS
 *  Bits 1-0
 *     -10mV a +10mV: 00
 *     -20mV a +20mV: 01
 *     -40mV a +40mV: 10
 *     -80mV a +80mV: 11 (default)
 *
```

```
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
 *
*/
int8_t PAC1710_set_vsense_rang(uint8_t v_vsense10);

/*************************************************************/
/**\name      FUNCTIONS FOR I2C READ AND WRITE */
/*************************************************************/
/*!
 *     \Brief: The function is used as I2C bus write
 *     \Return : Status of the I2C write
 *     \param dev_addr : The device address of the sensor
 *     \param reg_addr : Address of the first register,
 *   will data is going to be written
 *     \param reg_data : The data you want to write
 *     \param cnt : The number of data byte to be write
 */
int8_t PAC1710_I2C_bus_write(uint8_t dev_addr, uint8_t reg_addr,uint8_t
*reg_data, uint8_t cnt);

 /*!
 *     @Brief: The function is used as I2C bus read
 *     @param dev_addr : The device address of the sensor
 *     @param reg_addr : Address of the first register,
 *   will data is going to be read
 *     @param reg_data : This data read from the sensor,
 *   which is hold in an array
 *     @param cnt : The no of data byte of to be read
 *     @Return : Status of the I2C read
 */
int8_t PAC1710_I2C_bus_read(uint8_t dev_addr, uint8_t reg_addr,uint8_t *reg_data,
uint8_t cnt);
/*************************************************************/
/**\name      FUNCTIONS FOR READ THE VALUES OF VSENSE AND VSOURCE*/
/*************************************************************/
/*!
 *     @brief This API read the
 *     voltage in the registers 0x11(bits 7-0) and 0x12 (bits 7-5)
 *
 *     @param v_voltage: The value of voltage
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_read_voltage(uint16_t *v_voltage);
/*!
 *     @brief This API is used to read the
 *     current in the registers 0x0D(bits 7-0) and 0x0E (bits 7-5)
 *     @param v_current: The value of current
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_read_current(int16_t *v_current);
```

```
/*****************************************************************/
/**\name      FUNCTIONS FOR CALCULATE THE VALUES OF VOLTAGE AND CURRENT*/
/*****************************************************************/
/*!
 *     @brief This API used calculate the source voltage
 *     @param *v_voltage: The value of vsource_data_register
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_calculate_voltage(float *v_voltage);
/*!
 *     @brief This API used calculate the current
 *     @param *v_voltage: The value of vsense_data_register
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_calculate_current(float *v_current);

/*****************************************************************/
/**\name                          I2C TEST FUNCTION                */
/*****************************************************************/
/*!
 *     @brief This API used to test the I2C R/W
*/
void PAC1710_test(void);

/*****************************************************************/
/**\name                          TIME FUNCTIONS                   */
/*****************************************************************/
/*!
*      @brief This API used to wait an
*      specific time in ms
*      @param wait_time_ms: waiting time in ms
*
*/
void PAC1710_wait_time_ms(uint16_t wait_time_ms);

/*!
*      @brief This API used to wait an
*      specific time in ms
*      @param wait_time_ms: waiting time in s
*
*/
void PAC1710_wait_time_s(uint16_t wait_time_s);

void shot_timer_handler(SYS_Timer_t *timer);

/*****************************************************************/
/**\name                          MEASURE FUNCTIONS                */
/*****************************************************************/
/*!
*      @brief This API used to take measures of voltage
*      and/ or current
*      @param num_measures: The number of measures
*      @param mode: The mode (I/V/IV_MODE) **This function does not one shot
mode,
*      in the standby mode. For that application use the function
one_shot_measure
```

```
*      @param avr: The average for Vsoure and Vsense measures
*      @return results of the process
*      @retval 1 -> Success
*      @retval -1 -> Error
*/
int8_t PAC1710_cont_measure(int16_t num_measures,uint8_t mode, uint8_t avr);

/*!
*      @brief This API used to take measures of voltage
*      and/ or current
*      @param measure_time_s: the time in seconds between one shot measures
*      @param num_measures: The number of measures
*
*      @return results of the process
*      @retval 1 -> Success
*      @retval -1 -> Error
*/
int8_t PAC1710_one_shot_measure(uint16_t measure_time_ms,uint16_t
measure_time_s,uint16_t num_measures);


/*************************************************************************/
/**\name                          GET INFORMATION FUNCTION
       */
/*************************************************************************/
/*!
*      @brief This API show the information of the PAC1710 by serial port
*      @param data: the information you want to see
*             0--> All
*             1--> Mode
*             2--> Conversion rate
*             3--> Vsource sample time
*             4--> Vsource average
*             5--> Vsense sample time
*             6--> Vsense average
*             7--> Vsense range
*      @return results of the process
*      @retval 1 -> Success
*      @retval -1 -> Error
*/
int8_t PAC1710_get_conf_information(uint8_t data);

#endif
#endif //PLATFORM_XPLAINED_PRO_ATMEGA256RFR2
```

PAC1710.C

```
/*
 * \file PAC1710.c
 *
 * \brief This file contains sensor Driver support file for PAC1710 sensor
 *
 *
 * $Id:PAC1710.c Iván Garrido Vega $
 */

/* Only available for ATMEGA256RFR2 */
#ifdef PLATFORM_XPLAINED_PRO_ATMEGA256RFR2

/*************************************************************************/
/*---------------------------- Includes ----------------------------------*/
```

```
/*****************************************************************************/
#include "pac1710.h"
#include <math.h>
#include <stdio.h>
#include "halTimer.h"
#include "halTwi.h"
#include "halUart.h"
#include "hal.h"


/*****************************************************************************/
/*--------------------------- Variables -------------------------------------*/
/*****************************************************************************/
static struct PAC1710_T *p_PAC1710; /**< pointer to PAC1710 */

static SYS_Timer_t PacTimers[10];
/*!
 * @brief: This variable contains the information to sensor, and init values
 */
struct PAC1710_T g_PAC1710;

/*****************************************************************************/
/*------------------------- Implementations -------------------------------*/
/*****************************************************************************/
/*!
 *     @brief This function is used for initialize
 *     the PAC1710 sensor
 *     @param R_shunt: resistance you are using
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval <0 -> Error
*/
int8_t PAC1710_init(float R_shunt)
{
	uint8_t reg_data = PAC1710_INIT_VALUE;

	/* result of communication results*/
	p_PAC1710->status_com= PAC_ERROR;

	/* assign PAC1710 ptr */
	p_PAC1710 = &g_PAC1710;

	/* assign device address*/
	g_PAC1710.dev_addr = PAC1710_DEV_ADDR;

	/* assign Rshunt*/
	p_PAC1710->Rshunt=R_shunt;

	/* read Chip Id */
	p_PAC1710->status_com =
PAC1710_I2C_bus_read(PAC1710_DEV_ADDR,PAC1710_PRODUCT_ID_REG,
&reg_data,PAC1710_WR_DATA_N_BYTES);
	p_PAC1710->chip_id = reg_data;
	//If chip_id is OK->I2C read is OK. We can continue.
	if(p_PAC1710->chip_id==PAC1710_PRODUCT_ID_VAL){
		p_PAC1710->status_com=PAC1710_set_mode(PAC1710_STANDBY_MODE);
			//Checking correct operation
			if(p_PAC1710->mode!=PAC1710_STANDBY_MODE){
				p_PAC1710->status_com=-1;
				return p_PAC1710->status_com;
			}
			else p_PAC1710->status_com=PAC_SUCCESS;
```

```
		uint8_t check_var=0;
		/***** Set conversion rate***********/
		p_PAC1710-
>status_com=PAC1710_set_conv_rate(PAC1710_CONV_RATE_CONT);

		//Checking correct operation
		p_PAC1710->status_com=PAC1710_get_conv_rate(&check_var);
		if(check_var!=PAC1710_CONV_RATE_CONT){
			p_PAC1710->status_com=-2;
			return p_PAC1710->status_com;
		}
		/***** Set vsource***********/
		p_PAC1710-
>status_com=PAC1710_set_vsource_samp_time(PAC1710_VSOURCE_TIME_10);

		//Checking correct operation
		p_PAC1710->status_com=PAC1710_get_vsource_samp_time(&check_var);
		if(check_var!=PAC1710_VSOURCE_TIME_10){
			p_PAC1710->status_com=-3;
			return p_PAC1710->status_com;
		}

		p_PAC1710-
>status_com=PAC1710_set_vsource_avr(PAC1710_VSOURCE_AVR_OFF);

		//Checking correct operation
		p_PAC1710->status_com=PAC1710_get_vsource_avr(&check_var);
		if(check_var!=PAC1710_VSOURCE_AVR_OFF){
			p_PAC1710->status_com=-3;
			return p_PAC1710->status_com;
		}

		/***** Set vsense***********/
		p_PAC1710-
>status_com=PAC1710_set_vsense_avr(PAC1710_VSENSE_AVR_OFF);


		//Checking correct operation
		p_PAC1710->status_com=PAC1710_get_vsense_avr(&check_var);
		if(check_var!=PAC1710_VSENSE_AVR_OFF){
			p_PAC1710->status_com=-4;
			return p_PAC1710->status_com;
		}
		//Setting the best range. + Resolution
		uint8_t vsense_range;
		if(p_PAC1710->Rshunt*14<20){
			vsense_range=PAC1710_VSENSE_RANGE_20;
			p_PAC1710-
>status_com=PAC1710_set_vsense_rang(PAC1710_VSENSE_RANGE_20);
		}
		else if(p_PAC1710->Rshunt*14<40){
			vsense_range=PAC1710_VSENSE_RANGE_40;
			p_PAC1710-
>status_com=PAC1710_set_vsense_rang(PAC1710_VSENSE_RANGE_40);
		}
		else {
			vsense_range=PAC1710_VSENSE_RANGE_80;
			p_PAC1710-
>status_com=PAC1710_set_vsense_rang(PAC1710_VSENSE_RANGE_80);
		}
```

```
                //Checking correct operation
                p_PAC1710->status_com=PAC1710_get_vsense_rang(&check_var);
                if(check_var!=vsense_range){
                        p_PAC1710->status_com=-5;
                        return p_PAC1710->status_com;
                }
        HAL_UartWriteString("Configuration OK\n");
        //used to verify to PAC1710 is Init
        g_PAC1710.flag = 1;
        }
        else
                p_PAC1710->status_com = PAC_ERROR;
        if  (p_PAC1710->status_com==PAC_SUCCESS){
                        for(uint8_t i=1;i==7;i++){
                                p_PAC1710->status_com=PAC1710_get_conf_information(i);
                        }
                }
        else HAL_UartWriteString("Init error");
        return p_PAC1710->status_com;
}
/*************************************************************/
/**\name      FUNCTIONS FOR POWER MODE*/
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     Operational Mode from the sensor in the register 0x00 bit 0 and 1
 *     @param v_mode : The value of power mode
 *  value of 1-0 bits|    mode
 *  -----------------|------------------
 *            00           | PAC1710_IV_MODE
 *            01                   | PAC1710_I_MODE
 *            10                   | PAC1710_V_MODE
 *            11           | PAC1710_STANDBY_MODE
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_get_mode(uint8_t *v_mode)
{
        /* used to return the communication result*/
        p_PAC1710->status_com = PAC_ERROR;
        uint8_t reg_data = PAC1710_INIT_VALUE;
        /* check the p_PAC1710 structure pointer as NULL*/
        if (p_PAC1710->flag==PAC1710_NULL) {
                return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
        }
        else {
                //Read mode register
                p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);7
                //Read mode bits
                *v_mode = PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_MODE);
                p_PAC1710->mode = *v_mode;
                //Show mode in serial port
                p_PAC1710->status_com=PAC1710_get_conf_information(1);
                }
        return p_PAC1710->status_com;
}
/*!
```

```
 *     @brief This API used to set the
 *     Operational Mode from the sensor in the register 0x00 bit 0 and 1
 *     @param v_mode: The value of power mode
 *   value of 1-0 bits|    mode
 *   -----------------|------------------
 *            00           | PAC1710_ACTIVE_IV_MODE
 *            01                  | PAC1710_ACTIVE_I_MODE
 *            10                  | PAC1710_ACTIVE_V_MODE
 *            11           | PAC1710_STANDBY_MODE
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_set_mode(uint8_t v_mode)
{
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	uint8_t v_prev_mode_uint8_t = PAC1710_INIT_VALUE;

	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;	//If the device is not
initialized, return error
	}
	else {//check if standby mode is set. Else set stanby mode
		while(p_PAC1710->mode!=PAC1710_STANDBY_MODE){
			p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&v_prev_mode_uint8_t, PAC1710_WR_DATA_N_BYTES);
			while(PAC1710_read_conv_done_bit()!=1){p_PAC1710-
>status_com=PAC_ERROR;} //Check if conversion is finished
			reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_V_PIN,1);
			p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
			p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&v_prev_mode_uint8_t, PAC1710_WR_DATA_N_BYTES);
			//--------should be xxxxxxx1
			while(PAC1710_read_conv_done_bit()!=1){p_PAC1710-
>status_com=PAC_ERROR;} //Check if conversion is finished
			reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_I_PIN,1);
			p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
			//--------should be xxxxxx11 (3)
			p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&v_prev_mode_uint8_t, PAC1710_WR_DATA_N_BYTES);
			p_PAC1710->mode=
PAC1710_GET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_MODE);
		}

		if (v_mode==PAC1710_STANDBY_MODE){ //11
			return p_PAC1710->status_com;
		}

		else if(v_mode==PAC1710_V_MODE){
			reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_V_PIN,0);
			p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
			//--------should be xxxxxx10
```

```
                    p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    p_PAC1710->mode=
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_MODE);
                    }
                    //1.Change V bit first 2. Change I bit
             else if(v_mode==PAC1710_I_MODE){
                    reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_I_PIN, 0);
                    p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    //--------should be xxxxxx01
                    p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    p_PAC1710->mode=
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_MODE);
                    }

             else if(v_mode==PAC1710_IV_MODE){
                    reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_I_PIN,0);
                    p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&v_prev_mode_uint8_t, PAC1710_WR_DATA_N_BYTES);
                    //--------should be xxxxxx01
                    reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prev_mode_uint8_t,PAC1710_V_PIN,0);
                    p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    //--------should be xxxxxx00
                    p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_MODE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    p_PAC1710->mode=
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_MODE);
             }
             else{
                    p_PAC1710->status_com=PAC_ERROR;
                    return p_PAC1710->status_com;
             }
             if(p_PAC1710->status_com==PAC_ERROR){
                    return p_PAC1710->status_com;
                    }
             else{
                    return p_PAC1710->status_com;
             }
      }
}
/*!
 *     @brief This API used to set the
 *     One Shot Mode from the sensor in the register 0x02
 *     when the sensor is operating in Standby Mode
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
*/
int8_t PAC1710_one_shot_mode(void)
{
      /* used to return the communication result*/
      p_PAC1710->status_com = PAC_ERROR;
      uint8_t prev_mode=PAC1710_INIT_VALUE;
      uint8_t reg_data = 0x01;
```

```
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	//Check if mode is standby mode. Else set standby mode
	else {
		if(p_PAC1710->mode!=PAC1710_STANDBY_MODE)
			{
			while(prev_mode!=PAC1710_STANDBY_MODE)
				{
				p_PAC1710-
>status_com=PAC1710_set_mode(PAC1710_STANDBY_MODE);
				p_PAC1710->status_com=PAC1710_get_mode(&prev_mode);
				}
			}
		//Write on one-shot register will automatically start a conversion
cycle
		p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_ONE_SHOT_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
}
return p_PAC1710->status_com;
}

/*************************************************************/
/**\name	FUNCTIONS FOR CONVERSION RATE */
/*************************************************************/
/*!
 *	@brief This API used to set the
 *	Conversion rate parameters from the sensor in the register 0x1H bit 0 to 1
 *	@param v_conv_rate: The value of conversion rate register
 *
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> Error
 *
 *	CONVERSION_RATE SETTINGS
 Bits 1-0
 *	1 per second: 00
 *	2 per second: 01
 *	4 per second: 10
 *	Continuous: 11 (default)
 *
 */
int8_t PAC1710_get_conv_rate(uint8_t *v_conv_rate)
{
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	else {
		//Read conversion rate register
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_CONV_RATE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
		//Get conversion rate bits
		*v_conv_rate =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_CONV_RATE);
```

```
		p_PAC1710->conv_rate = *v_conv_rate;
		//Show conversion rate by serial port
		p_PAC1710->status_com=PAC1710_get_conf_information(2);
}
return p_PAC1710->status_com;
}
/*!
 *	@brief This API used to set the
 *	Conversion rate parameters from the sensor in the register 0x1H bit 0 to 1
 *
 *	@param v_conv_rate: The value of conversion rate register
 *
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> Error
 *
 *	CONVERSION_RATE SETTINGS
 Bits 1-0
 *	1 per second: 00
 *	2 per second: 01
 *	4 per second: 10
 *	Continuous: 11 (default)
 *
 */
int8_t PAC1710_set_conv_rate(uint8_t v_conv_rate)
{
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	uint8_t v_prereg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;	//If the device is not
initialized, return error
	}
	else {
		//Check if standby mode is set. Else set standby mode to be able to
change conversion register
			if(p_PAC1710->mode !=PAC1710_STANDBY_MODE){
				PAC1710_set_mode(PAC1710_STANDBY_MODE);
			}
		//Read register
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_CONV_RATE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
		//Change only the bits related to conversion rate
		reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_CONV_RATE, v_conv_rate);
		//Write new data
		p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_CONV_RATE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
		if (p_PAC1710->status_com==PAC_SUCCESS){
			p_PAC1710->conv_rate = v_conv_rate;
		}
	}
	return p_PAC1710->status_com;
}

/*!
 *	@brief This API used to read de bit 7 of the Hight Limit Status Register
(0x04)
 *	that determines if the conv_rate is finished ('1') or not ('0')
 *
```

```
 *     @param v_conv_done_bit: The variable where the value of conv_done bit will
be saved
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success, conversion done
 *     @retval 0 -> Success, conversion in process
 *     @retval -1 -> PAC_ERROR
 *
 */
int8_t PAC1710_read_conv_done_bit(void){
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       uint8_t conv_done_bit;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
              else {
                     //Read register
                     p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_HLS_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                     if(p_PAC1710->status_com==1){
                            //Get only the value of the indicated bits
                            conv_done_bit =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_HLS);
                            p_PAC1710->conv_done = conv_done_bit;
                            return conv_done_bit;
                     }
                     else{return PAC_ERROR;}
              }

}

/*************************************************************/
/**\name      FUNCTIONS FOR VSOURCE*/
/*************************************************************/
/*************************************************************/
/**\name      VSOURCE SAMPLING TIME SETTINGS*/
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     vsource sampling settings parameters from the sensor in the register 0xAH
bits 2 to 3
 *
 *     @param v_vsource32: The value of bits 3-2 of vsource register
 *
 *     VOLTAGE-SAMPLING TIME SETTINGS
 *     bit 3-2
 * 2.5 ms (data=8bits: 00
 * 5ms (data=9bits): 01
 * 10ms (data=10bits): 10 (default)
 * 20ms(data=11bits): 11
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
 *
*/
int8_t PAC1710_get_vsource_samp_time(uint8_t *v_vsource32)
```

```
{
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;	//If the device is not
initialized, return error
	}
	else {
		//Read register
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
		//Get bits value
		*v_vsource32 =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_VSOURCE_TIME);
		p_PAC1710->vsource_stime = *v_vsource32;
		//Show by serial port the value of vsource sample time
		p_PAC1710->status_com=PAC1710_get_conf_information(3);
	}
	return p_PAC1710->status_com;
}
/*!
*	@brief This API used to set the
*	vsource sampling settings parameters from the sensor in the register 0xAH
bits 2 to 3
*
*	@param v_vsource32: The value of bits 3-2 of vsource register
*
*	VOLTAGE-SAMPLING TIME SETTINGS
*	bit 3-2
*	2.5 ms (data=8bits: 00
*	5ms (data=9bits): 01
*	10ms (data=10bits): 10 (default)
*	20ms(data=11bits): 11
*
*	@return results of bus communication function
*	@retval 1 -> Success
*	@retval -1 -> PAC_ERROR
*
*
*/
int8_t PAC1710_set_vsource_samp_time(uint8_t v_vsource32)
{
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	uint8_t v_prereg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;	//If the device is not
initialized, return error
	}
	else {
			//read vsource register
			p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
			//change only vsource_sample_time bits
			reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_VSOURCE_TIME,v_vsource32);
			//Rewrite register
```

```
                    p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                    if (p_PAC1710->status_com==PAC_SUCCESS){
                           p_PAC1710->vsource_stime= v_vsource32;
             }
       }
       return p_PAC1710->status_com;
}

/*************************************************************/
/**\name      VSOURCE AVERAGE SETTINGS*/
/*************************************************************/
/*!
*      @brief This API used to get the
*      vsource average settings parameters from the sensor in the register 0xAH
bits 0 to 1
*
*      @param v_vsource10: The value of bits 1-0 of vsource register
*
VOLTAGE-AVERAGE SETTINGS
 Bits 1-0
 *     Deactivated: 00 (default)
 *     2: 01
 *     4: 10
 *     8: 11
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
 *
 */
int8_t PAC1710_get_vsource_avr(uint8_t *v_vsource10)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
             return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
       else {
             //Read register
             p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
             //Get vsource average bits value
             *v_vsource10 =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_VSOURCE_AVERAGE);
             p_PAC1710->vsource_avr = *v_vsource10;
             //Show by serial port
             p_PAC1710->status_com=PAC1710_get_conf_information(4);
       }
       return p_PAC1710->status_com;
}

/*!
*      @brief This API used to set the
*      vsource average settings parameters from the sensor in the register 0xAH
bits 0 to 1
*
*      @param v_vsource10: The value of bits 1-0 of vsource register
```

```
*
VOLTAGE-AVERAGE SETTINGS
Bits 1-0
*      Deactivated: 00 (default)
*      2: 01
*      4: 10
*      8: 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> PAC_ERROR
*
*
*/
int8_t PAC1710_set_vsource_avr(uint8_t v_vsource10)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       uint8_t v_prereg_data = PAC1710_INIT_VALUE;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
       else {
              //Read register
              p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
              //Set the new value of vsense sample time bits
              reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_VSOURCE_AVERAGE,v_vsource10);
              //Rewrite register
              p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
              if (p_PAC1710->status_com==PAC_SUCCESS){
                     p_PAC1710->vsource_avr= v_vsource10;
              }
       }
       return p_PAC1710->status_com;
}

/*************************************************************/
/**\name      FUNCTIONS FOR VSENSE*/
/*************************************************************/
/*************************************************************/
/**\name      FUNCTION FOR CURRENT SAMPLING TIME */
/*************************************************************/
/*!
 *     @brief This API used to get the
 *     vsense parameters from the sensor in the register 0x0B bit 4 to 6
 *
 *     @param v_vsense64: The value of vsense register bits 6-4
 *
 *     CURRENT-SENSING SAMPLING TIME
 *     bit 7-> Unused
 *     bits 6-4
 *  2.5ms: 000
 *  5ms: 001
 *     10ms: 010
 *     20ms: 011
 *     40ms: 100
```

```
 *     80ms: 101 (default)
 *     160ms: 110
 *     320ms: 111
 *
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
 *
*/
int8_t PAC1710_get_vsense_stime(uint8_t *v_vsense64)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
       else {
              //read vsense register
              p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
              //get value of vsense sample time bits
              *v_vsense64 =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_VSENSE_SAMPLING_TIME);
              p_PAC1710->vsense_stime = *v_vsense64;
              //Show sample time by serial port
              p_PAC1710->status_com=PAC1710_get_conf_information(5);
       }
       return p_PAC1710->status_com;
}
/*!
 *     @brief This API used to set the
 *     vsense parameters from the sensor in the register 0x0B bits 4 to 6
 *
 *     @param v_vsense64: The value of vsense register bits 6-4
 *
 *     CURRENT-SENSING RANGE SETTINGS
 *     bit 7-> Unused
 *     bits 6-4
 *  2.5ms: 000
 *  5ms: 001
 *     10ms: 010
 *     20ms: 011
 *     40ms: 100
 *     80ms: 101 (default)
 *     160ms: 110
 *     320ms: 111
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
*/
int8_t PAC1710_set_vsense_stime(uint8_t v_vsense64)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
```

```
	uint8_t v_prereg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	else {
		//Read vsense register
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
		//Set the new value of vsense sample time bits
		reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_VSENSE_SAMPLING_TIME,v_vsense64
);
		//Write new data
		p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
		if (p_PAC1710->status_com==PAC_SUCCESS){
			p_PAC1710->vsense_stime= v_vsense64;
		}
	}
	return p_PAC1710->status_com;
}

/*************************************************************/
/**\name	FUNCTION FOR CURRENT SENSING AVERAGING*/
/*************************************************************/
/*!
 *	@brief This API used to get the
 *	vsense sensing averaging parameters from the sensor in the register 0x0B
bit 2 to 3
 *
 *	@param v_vsense32: The value of vsense register bits 3-2
 *
 *	CURRENT-SENSING AVERAGING SETTINGS
Bits 3-2
 *	Deactivated: 00 (default)
 *	2: 01
 *	4: 10
 *	8: 11
 *
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> PAC_ERROR
 *
*/
int8_t PAC1710_get_vsense_avr(uint8_t *v_vsense32)
{
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	else {
		//Read register
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
		//Get vsense average bits value
```

```
            *v_vsense32 =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_VSENSE_AVERAGE);
            p_PAC1710->vsense_avr = *v_vsense32;
            //Show vsense average by serial port
            p_PAC1710->status_com=PAC1710_get_conf_information(6);
}
return p_PAC1710->status_com;
}
/*!
*      @brief This API used to set the
*      vsense sensing averaging parameters from the sensor in the register 0x0B
bit 2 to 3
*
*      @param v_vsense32: The value of vsense register bits 3-2
*
*      CURRENT-SENSING AVERAGING SETTINGS
Bits 3-2
*      Deactivated: 00 (default)
*      2: 01
*      4: 10
*      8: 11
*
*      @return results of bus communication function
*      @retval 1 -> Success
*      @retval -1 -> PAC_ERROR
*
*
*/
int8_t PAC1710_set_vsense_avr(uint8_t v_vsense32)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       uint8_t v_prereg_data = PAC1710_INIT_VALUE;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
       else {
              //Read vsense register
              p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
              //Set new value of vsense range
              reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_VSENSE_AVERAGE,v_vsense32);
              //Rewrite register
              p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
       if (p_PAC1710->status_com==PAC_SUCCESS){
              p_PAC1710->vsense_avr= v_vsense32;
              }
       }
       return p_PAC1710->status_com;
}

/*************************************************************/
/**\name      FUNCTION FOR CURRENT SENSING RANGE*/
/*************************************************************/
/*!
 *     @brief This API used to get the
```

```
 *     vsense sensing averaging parameters from the sensor in the register 0x0B
bit 0 to 1
 *
 *     @param v_vsense10: The value of vsense register bits 1-0
 *
 *     CURRENT-SENSING RANGE SETTINGS
 Bits 1-0
 *     -10mV a +10mV: 00
 *     -20mV a +20mV: 01
 *     -40mV a +40mV: 10
 *     -80mV a +80mV: 11 (default)
 *
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
*/
int8_t PAC1710_get_vsense_rang(uint8_t *v_vsense10)
{
       /* used to return the communication result*/
       p_PAC1710->status_com = PAC_ERROR;
       uint8_t reg_data = PAC1710_INIT_VALUE;
       /* check the p_PAC1710 structure pointer as NULL*/
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
              else {
                     //Read Vsense regiter
                     p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
                     // Read vsense range value
                     *v_vsense10 =
PAC1710_GET_REG_BITS_VALUE(reg_data,PAC1710_VSENSE_RANGE);
                     p_PAC1710->vsense_rang = *v_vsense10;
                     //Show by serial port
                     p_PAC1710->status_com=PAC1710_get_conf_information(7);
       }
       return p_PAC1710->status_com;
}
/*!
 *     @brief This API used to set the
 *     vsense sensing averaging parameters from the sensor in the register 0x0B
bit 0 to 1
 *
 *     @param v_vsense10: The value of vsense register bits 1-0
 *
 *     CURRENT-SENSING RANGE SETTINGS
 Bits 1-0
 *     -10mV a +10mV: 00
 *     -20mV a +20mV: 01
 *     -40mV a +40mV: 10
 *     -80mV a +80mV: 11 (default)
 *
 *
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> PAC_ERROR
 *
 *
*/
```

```
int8_t PAC1710_set_vsense_rang(uint8_t v_vsense10)
{

	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data = PAC1710_INIT_VALUE;
	uint8_t v_prereg_data = PAC1710_INIT_VALUE;
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	else {
		//Read vsense regiter
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&v_prereg_data, PAC1710_WR_DATA_N_BYTES);
		//Set vsense range new value
		reg_data
=PAC1710_SET_REG_BITS_VALUE(v_prereg_data,PAC1710_VSENSE_RANGE,v_vsense10);
		//Rewrite register
		p_PAC1710->status_com = PAC1710_I2C_bus_write(p_PAC1710-
>dev_addr,PAC1710_VSENSE_ADDR,&reg_data, PAC1710_WR_DATA_N_BYTES);
	if (p_PAC1710->status_com==PAC_SUCCESS){
			p_PAC1710->vsense_rang= v_vsense10;
		}
	}
	return p_PAC1710->status_com;
}
/*************************************************************/
/**\name	FUNCTIONS FOR I2C READ AND WRITE */
/*************************************************************/
 /*!
 *	@Brief: The function is used as I2C bus write
 *	@param dev_addr : The device address of the sensor
 *	@param reg_addr : Address of the first register,
 *   will data is going to be written
 *	@param reg_data : The data you want to write
 *	@param cnt : The number of data byte to be write
 *
 *	@Return : Status of the I2C write
 */
int8_t PAC1710_I2C_bus_write(uint8_t dev_addr, uint8_t reg_addr,uint8_t
*reg_data, uint8_t cnt)
{
	int8_t write_status=PAC_SUCCESS;
	uint8_t array[2];
	uint8_t stringpos = 0;
	array[0] = reg_addr;	//First value of the array is the address of the
register to be written
	for (stringpos = PAC1710_INIT_VALUE; stringpos < cnt; stringpos++) {
		array[stringpos + 1] = *(reg_data + stringpos);
	}
	dev_addr = PAC1710_DEV_ADDR;
	twi_package_t packet = {
		.addr[0]	=  reg_addr,		/* TWI slave memory address data
*/
		.addr_length	= (uint8_t)0,		/* TWI slave memory address data
size */
		.chip		= (char)dev_addr, /* TWI slave bus address */
		.buffer		= &array[0],		/* transfer data source buffer */
		.length		= cnt + 1			/* transfer data size
(bytes) */
```

```
	};
	//I2C write rutine. (I2C HalTwi function)
		if (HAL_TwiMaster_Write(TWI_BASE_ADDRESS,&packet) == TWI_SUCCESS) {
		 write_status=PAC_SUCCESS;
		return write_status;
		}
	return (int8_t)write_status;
}

 /*!
 *	@Brief: The function is used as I2C bus read
 *	@param dev_addr : The device address of the sensor
 *	@param reg_addr : Address of the first register, will data is going to be
read
 *	@param reg_data : This data read from the sensor, which is hold in
 *   an array
 *	@param cnt : The no of data byte of to be read
 *
 *  @Return : Status of the I2C read
 */
int8_t PAC1710_I2C_bus_read(uint8_t dev_addr, uint8_t reg_addr,uint8_t *reg_data,
uint8_t cnt)
{
	int32_t iPAC_ERROR = PAC_SUCCESS;
	uint8_t stringpos = PAC1710_INIT_VALUE;
	dev_addr = PAC1710_DEV_ADDR;
	twi_package_t packet = {
		.addr[0]	=  reg_addr,			/* TWI slave memory
address data */
		.addr_length   = (uint8_t)0,			/* TWI slave memory
address data size */
		.chip		= ( char )dev_addr,	/* TWI slave bus address */
		.buffer		= &reg_addr,			/* transfer data
source buffer */
		.length		= 1				/* transfer data
size (bytes) */

	};
	if (HAL_TwiMaster_Write(TWI_BASE_ADDRESS,&packet) == TWI_SUCCESS)
	{
		uint8_t data_received[4] = {0};
		twi_package_t packet_received = {
			.addr[0]	= reg_addr,		/* TWI slave memory address
data */
			.addr_length  = (uint8_t)0,		/* TWI slave memory address
data size */
			.chip		= (char)dev_addr, /* TWI slave bus address */
			.buffer		= data_received,	/* transfer data
destination buffer */
			.length		= cnt			/* transfer data
size (bytes) */
		};
		if (HAL_TwiMaster_Read(TWI_BASE_ADDRESS,&packet_received) ==
TWI_SUCCESS)
		{
			for (stringpos = PAC1710_INIT_VALUE; stringpos < cnt;
stringpos++)
			{
				*(reg_data + stringpos) = data_received[stringpos];
			}
		}
		else
```

```
		{
			iPAC_ERROR = PAC_ERROR;
		}
	}
	else
	{
		iPAC_ERROR = PAC_ERROR;
	}
	return (int8_t)iPAC_ERROR;
}
/*************************************************************/
/**\name	FUNCTIONS FOR READ THE VALUES OF VSENSE AND VSOURCE*/
/*************************************************************/
/*!
 *	@brief This API read the
 *	voltage in the registers 0x11(bits 7-0) and 0x12 (bits 7-5)
 *
 *	@param v_voltage: The value of voltage
 *
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> Error
 *
*/
int8_t PAC1710_read_voltage(uint16_t *v_voltage){
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;	//If the device is not
initialized, return error
	}
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data[2]= {0,0};
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710 == PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;
		} else {
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSOURCE_HB_ADDR,reg_data,PAC1710_RESULTS_DATA_N_BYTES);
		*v_voltage=(uint16_t)(reg_data[0]<<8)|(uint16_t)(reg_data[1]);
		p_PAC1710->voltage_vsource = *v_voltage;
	}
	return p_PAC1710->status_com;
}
/*!
 *	@brief This API is used to read the
 *	current in the registers 0x0D(bits 7-0) and 0x0E (bits 7-5)
 *	@param v_current: The value of current
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> Error
 *
*/
int8_t PAC1710_read_current(int16_t *v_current){
	/* used to return the communication result*/
	p_PAC1710->status_com = PAC_ERROR;
	uint8_t reg_data[2]= {0,0};
	/* check the p_PAC1710 structure pointer as NULL*/
	if (p_PAC1710 == PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;
		} else {
		p_PAC1710->status_com = PAC1710_I2C_bus_read(p_PAC1710-
>dev_addr,PAC1710_VSENSE_HB_ADDR,reg_data,PAC1710_RESULTS_DATA_N_BYTES);
```

```
            *v_current=(uint16_t)(reg_data[0]<<8)|(uint16_t)(reg_data[1]);
            p_PAC1710->current_vsense= *v_current;
}
return p_PAC1710->status_com;
}

/*************************************************************************/
/**\name      FUNCTIONS FOR CALCULATE THE VALUES OF VOLTAGE AND CURRENT*/
/*************************************************************************/
/*!
 *     @brief This API used calculate the source voltage
 *     @param *v_voltage: The value of vsource_data_register
 *     @return results of bus communication function
 *     @retval 1 -> Success
 *     @retval -1 -> Error
 *
*/
int8_t PAC1710_calculate_voltage(float *v_voltage){
      if (p_PAC1710->flag==PAC1710_NULL) {
            return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
      }
      //Initialize variables
      int8_t rslt= 0;
      uint8_t bit_shift_right=0;
      float denv2=0;
      float denv1=0;
      float fsv=0;
      //Set vsource denominator 2 and set time value in ms*10
      //of sample time. Dependent of vsource sample time
      if(p_PAC1710->vsource_stime==0){
            denv2=255;
            p_PAC1710->vsource_stime_10ms=25;
            bit_shift_right=PAC1710_V_I_DATA_N_BITS-8;
      }
      else if(p_PAC1710->vsource_stime==1){
            denv2=511;
            p_PAC1710->vsource_stime_10ms=50;
            bit_shift_right=PAC1710_V_I_DATA_N_BITS-9;
      }
      else if(p_PAC1710->vsource_stime==2){
            denv2=1023;
            p_PAC1710->vsource_stime_10ms=100;
            bit_shift_right=PAC1710_V_I_DATA_N_BITS-10;
      }
      else if(p_PAC1710->vsource_stime==3){
            denv2=2047;
            p_PAC1710->vsource_stime_10ms=200;
            bit_shift_right=PAC1710_V_I_DATA_N_BITS-11;
      }
      else{
            rslt = PAC_ERROR;
            return rslt;
      }
      //Set Vsource denominator 1
      denv1=denv2+1;

      //CALCULATE FULL-SCALE VOLTAGE
      fsv=(uint16_t)40-((uint16_t)40/denv2);

      //CALCULATE BUS VOLTAGE
      *v_voltage=(p_PAC1710->voltage_vsource>>bit_shift_right)*fsv/denv1;
```

```
	p_PAC1710->vsource_pin=*v_voltage;
	return rslt;
}
/*!
 *	@brief This API used calculate the current
 *	@param *v_voltage: The value of vsense_data_register
 *	@return results of bus communication function
 *	@retval 1 -> Success
 *	@retval -1 -> Error
 *
*/
int8_t PAC1710_calculate_current(float *v_current){
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
	//Initialize variables
	float rslt= 0;
	float deni=0;
	float fsc=0;
	float fsr=0;
	uint8_t bit_shift_right=0;
	//Set vsense denominator and sample time value ms*10. Dependent of
vsense_time
	//Bit in use depends of sample time too
	if(p_PAC1710->vsense_stime==0){
		deni=63;
		bit_shift_right=9;
		p_PAC1710->vsense_stime_10ms=25;
	}

	else if(p_PAC1710->vsense_stime==1){
		deni=127;
		bit_shift_right=8;
		p_PAC1710->vsense_stime_10ms=50;
	}
	else if(p_PAC1710->vsense_stime==2){
		deni=255;
		bit_shift_right=7;
		p_PAC1710->vsense_stime_10ms=100;
	}
	else if(p_PAC1710->vsense_stime==3){
		deni=511;
		bit_shift_right=6;
		p_PAC1710->vsense_stime_10ms=200;
	}
	else if(p_PAC1710->vsense_stime==4){
		deni=1023;
		bit_shift_right=5;
		p_PAC1710->vsense_stime_10ms=400;
	}
	else if(p_PAC1710->vsense_stime==5){
		deni=2047;
		bit_shift_right=4;
		p_PAC1710->vsense_stime_10ms=800;
	}
	else if(p_PAC1710->vsense_stime==6){
		deni=2047;
		bit_shift_right=4;
		p_PAC1710->vsense_stime_10ms=1600;
	}
	else if(p_PAC1710->vsense_stime==7){
```

```
		deni=2047;
		bit_shift_right=4;
		p_PAC1710->vsense_stime_10ms=3200;
	}
	//Set range numerical value
	switch(p_PAC1710->vsense_rang){
		case(0):fsr=10;break;
		case(1):fsr=20;break;
		case(2):fsr=40;break;
		case(3):fsr=80;break;
		default:rslt = PAC_ERROR;
		return rslt;
	}

		fsc=fsr/p_PAC1710->Rshunt;

		//CALCULATE BUS CURRENT
		if(p_PAC1710->current_vsense<0)
		{
			p_PAC1710->current_vsense=-p_PAC1710->current_vsense; //If
is negative, turn positive
		}
		//Use only the bits of the sample time selected
		p_PAC1710->current_vsense=p_PAC1710-
>current_vsense>>bit_shift_right;
		//Calculate current. See data sheet
		*v_current=fsc*(p_PAC1710->current_vsense);
		*v_current=*v_current/deni;
		p_PAC1710->ibus=*v_current;
		return rslt;
}
/*************************************************************************/
/**\name                          I2C TEST FUNCTION
		*/
/*************************************************************************/
/*!
 *     @brief This API used to test the I2C R/W
*/
void PAC1710_test(void){
		int8_t status_com = PAC_ERROR;
		uint8_t configuracion=0x00;
		uint8_t reg_addr = 0x00;
		uint8_t i=0;
		uint8_t dev_addr= PAC1710_DEV_ADDR;

		//CHECKING THAT I2C READ AND WRITE IS CORRECT
		//----------READ-------
		HAL_UartWriteString( "----CHECKING FUNCION DE LECTURA-----\n\n");
		uint8_t product_information[3]={0};
		reg_addr=PAC1710_PRODUCT_ID_REG;

	status_com=PAC1710_I2C_bus_read(dev_addr,reg_addr,&product_information[0],
1);
		if(product_information[0]==PAC1710_PRODUCT_ID_VAL &&
status_com==1){//product_information[0] should be 58
			reg_addr=MANUFACTURER_ID_REG;

	status_com=PAC1710_I2C_bus_read(dev_addr,reg_addr,&product_information[1],
1);
			if(product_information[1]==MANUFACTURER_ID_VAL &&
status_com==1){//product_information[1] should be 5D
				reg_addr=REVISION_REG;
```

```
	status_com=PAC1710_I2C_bus_read(dev_addr,reg_addr,&product_information[2],
1);
				if(product_information[2]==REVISION_REG_VAL &&
status_com==1){//product_information[2] should be 81
					HAL_UartWriteString( "I2C READ OK\n");
				}
				else HAL_UartWriteString( "I2C READ PAC_ERROR\n");
			}
			else HAL_UartWriteString( "I2C READ PAC_ERROR\n");
		}
		else HAL_UartWriteString( "I2C READ PAC_ERROR\n");

		//----WRITE-----
		HAL_UartWriteString( "----CHECKING FUNCION DE ESCRITURA-----\n\n");
		reg_addr=PAC1710_MODE_ADDR;
		uint8_t mode_addr_val[2]={0};
		configuracion=0x1B;

	status_com=PAC1710_I2C_bus_read(dev_addr,reg_addr,&mode_addr_val[0],1);

	status_com=PAC1710_I2C_bus_write(dev_addr,reg_addr,&configuracion,1);

	status_com=PAC1710_I2C_bus_read(dev_addr,reg_addr,&mode_addr_val[1],1);
		if(mode_addr_val[0]!=mode_addr_val[1]){
			HAL_UartWriteString( "I2C WRITE OK\n");
		}
		else HAL_UartWriteString( "I2C WRITE PAC_ERROR\n");
		status_com=0;

		if (p_PAC1710->flag!=1){
			p_PAC1710 = &g_PAC1710;
		}

		//-----------------------------------------------------------------
--------------------------------
		//CHECKING VSOURCE SET AND GET
		HAL_UartWriteString( "CHECKING VSOURCE SAMPLE TIME\n\n");
		uint8_t vsource_samp_time[PAC1710_VSOURCE_TIME_20+1]= {0};
		for(i=PAC1710_VSOURCE_TIME_2F5;i<=PAC1710_VSOURCE_TIME_20;i++){
			status_com=PAC1710_set_vsource_samp_time(i);

	status_com=PAC1710_get_vsource_samp_time(&vsource_samp_time[i]);
			if (i==vsource_samp_time[i]){
				HAL_UartWriteString( "vsource_sample_time is correct:
");
				status_com=PAC1710_get_conf_information(3);
			}
			else HAL_UartWriteString("PAC_ERROR setting\n");
		}

		//VSOURCE_AVERAGE
		HAL_UartWriteString( "CHECKING VSOURCE AVERAGE\n\n");
		uint8_t vsource_avr[PAC1710_VSOURCE_AVR_8 +1]= {0};
		for(i=PAC1710_VSOURCE_AVR_OFF;i<=PAC1710_VSOURCE_AVR_8;i++){
			status_com=PAC1710_set_vsource_avr(i);
			status_com=PAC1710_get_vsource_avr(&vsource_avr[i]);
			if (i==vsource_avr[i]){
				HAL_UartWriteString( "vsource_sample_avr is correct:
");
				status_com=PAC1710_get_conf_information(4);
			}
```

```
		else HAL_UartWriteString("PAC_ERROR setting\n");
	}
	//-------------------------------------------------------------------
----
	//CHECKING VSENSE SET AND GET
	//VSENSE_SAMPLE_TIME
	HAL_UartWriteString( "CHECKING VSENSE SAMPLE TIME\n\n");
	uint8_t vsense_samp_time[PAC1710_VSENSE_TIME_320 +1]= {0};
	for(i=PAC1710_VSENSE_TIME_2F5;i<=PAC1710_VSENSE_TIME_320;i++){
		status_com=PAC1710_set_vsense_stime(i);
		status_com=PAC1710_get_vsense_stime(&vsense_samp_time[i]);
		if (i==vsense_samp_time[i]){
			HAL_UartWriteString( "vsense_samp_time is correct: ");
			status_com=PAC1710_get_conf_information(5);
		}
		else HAL_UartWriteString("PAC_ERROR setting\n");
	}

	//VSENSE_AVERAGE
	HAL_UartWriteString( "CHECKING VSENSE AVERAGE\n\n");
	uint8_t vsense_avr[PAC1710_VSENSE_AVR_8 +1]= {0};
	for(i=PAC1710_VSENSE_AVR_OFF;i<=PAC1710_VSENSE_AVR_8;i++){
		status_com=PAC1710_set_vsense_avr(i);
		status_com=PAC1710_get_vsense_avr(&vsense_avr[i]);
		if (i==vsense_avr[i]){
			HAL_UartWriteString("vsense_average is correct: ");
			status_com=PAC1710_get_conf_information(6);
		}
		else HAL_UartWriteString("PAC_ERROR setting\n");
	}

	//VSENSE RANGE
	HAL_UartWriteString( "CHECKING VSENSE RANGE\n\n");
	uint8_t vsense_rang[PAC1710_VSENSE_RANGE_80 +1]= {0};
	for(i=PAC1710_VSENSE_RANGE_10;i<=PAC1710_VSENSE_RANGE_80;i++){
		status_com=PAC1710_set_vsense_rang(i);
		status_com=PAC1710_get_vsense_rang(&vsense_rang[i]);
		if (i==vsense_rang[i]){
			HAL_UartWriteString("vsense_range is correct: ");
			status_com=PAC1710_get_conf_information(7);
		}
		else HAL_UartWriteString("PAC_ERROR setting\n");
	}

	//CHECKING MODE SET AND GET
	HAL_UartWriteString( "CHECKING MODE\n\n");
	uint8_t mode[PAC1710_STANDBY_MODE +1]= {0};
	for(i=PAC1710_IV_MODE;i<=PAC1710_STANDBY_MODE;i++){
		status_com=PAC1710_set_mode(i);
		status_com=PAC1710_get_mode(&mode[i]);
		if (i==mode[i]){
			HAL_UartWriteString("mode is correct: ");
			status_com=PAC1710_get_conf_information(1);
		}
		else HAL_UartWriteString("PAC_ERROR setting\n");
	}

	//CHECKING CONVERSION RATE SET AND GET
	HAL_UartWriteString( "CHECKING CONVERSION RATE\n\n");
	status_com=PAC1710_set_mode(PAC1710_STANDBY_MODE);
	uint8_t conv_rate[PAC1710_CONV_RATE_CONT +1]= {0};
	for(i=PAC1710_CONV_RATE_1SEC;i<=PAC1710_CONV_RATE_CONT;i++){
```

```
                        status_com=PAC1710_set_conv_rate(i);
                        status_com=PAC1710_get_conv_rate(&conv_rate[i]);
                        if (i==conv_rate[i]){
                                HAL_UartWriteString("conv_rate is correct: ");
                                status_com=PAC1710_get_conf_information(2);
                        }
                        else HAL_UartWriteString("PAC_ERROR setting\n");

                        status_com=PAC1710_set_conv_rate(PAC1710_CONV_RATE_CONT);
                }

}

/*************************************************************************/
/**\name                              TIME FUNCTIONS
                        */
/*************************************************************************/
/*!
*       @brief This API used to wait an
*       specific time in ms
*       @param wait_time_ms: waiting time in ms
*
*/
void PAC1710_wait_time_ms(uint16_t wait_time_ms){
        uint16_t j=0;
        for(j=0;j<=wait_time_ms*10;j++){
                HAL_Delay(100);
        }
}

/*!
*       @brief This API used to wait an
*       specific time in ms
*       @param wait_time_ms: waiting time in s
*
*/
void PAC1710_wait_time_s(uint16_t wait_time_s){
        uint16_t j=0;
        uint16_t i=0;
        for(j=0;j<=wait_time_s*10;j++){
                for(i=0;i<1000;i++){
                HAL_Delay(100);
                }
        i=0;
        }
}

/*************************************************************************/
/**\name                              MEASURE FUNCTIONS
                */
/*************************************************************************/
/*!
*       @brief This API used to take measures of voltage
*       and/ or current
*       @param num_measures: The number of measures
*       @param mode: The mode (I/V/IV_MODE) **This function does not one shot
mode,
*       in the standby mode. For that application use the function
one_shot_measure
*       @param avr: The average for Vsoure and Vsense measures
*       @return results of the process
*       @retval 1 -> Success
```

```
*      @retval -1 -> Error
*/
int8_t PAC1710_cont_measure(int16_t num_measures,uint8_t mode, uint8_t avr){
       if (p_PAC1710->flag==PAC1710_NULL) {
              return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
       }
       //Initialize variables
       int8_t status_com=PAC_ERROR;
       uint8_t avr_counter=0;
       int8_t conv_done=0;

       char voltage_str[30] = {0};
       char current_str[30] = {0};
       char time_data_str[10] = {0};
       char time_data_str2[10] = {0};

       uint16_t voltage_e[num_measures];
       uint16_t voltage_d[num_measures];
       uint16_t current_e[num_measures];
       uint16_t current_d[num_measures];
       uint16_t time_e[num_measures];
       uint16_t time_d[num_measures];

       p_PAC1710->time_values[0]=0;
       //Set mode
       status_com=PAC1710_set_mode(mode);
       //Set average
       status_com=PAC1710_set_vsource_avr(avr);
       status_com=PAC1710_set_vsense_avr(avr);
       //I mode rutine
       if(mode==PAC1710_I_MODE){
              for(uint8_t i=0; i<num_measures;i++){
                     if(avr==0){

       while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                     }
                     else{
                            while(avr_counter<avr){

       while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                   avr_counter++;
                            }
                     }
                     status_com=PAC1710_read_current(&p_PAC1710-
>current_reg_values[i]);
                     status_com=PAC1710_calculate_current(&p_PAC1710-
>current_real_values[i]);
                     p_PAC1710->time_values[i]=p_PAC1710->time_values[i-
1]+(float)(p_PAC1710->vsense_stime_10ms/10);
                     avr_counter=0;
              }
       }
       //Vmode routine
       else if(mode==PAC1710_V_MODE){
              for(uint8_t i=0; i<num_measures;i++){
                     if(avr==0){

       while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                     }
                     else{
                            while(avr_counter<avr){
```

```
        while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                        avr_counter++;
                                }
                        }
                        status_com=PAC1710_read_voltage(&p_PAC1710-
>voltage_reg_values[i]);
                        status_com=PAC1710_calculate_voltage(&p_PAC1710-
>voltage_real_values[i]);
                        p_PAC1710->time_values[i]=p_PAC1710->time_values[i-
1]+(float)(p_PAC1710->vsource_stime_10ms/10);
                        avr_counter=0;
                }
        }
        //IV mode rutine
        else if(mode==PAC1710_IV_MODE){
                uint8_t avr_counter2=0;
                for(uint8_t i=0; i<num_measures;i++){
                        if(avr==0){

        while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                status_com=PAC1710_read_current(&p_PAC1710-
>current_reg_values[i]);
                                status_com=PAC1710_calculate_current(&p_PAC1710-
>current_real_values[i]);
                                p_PAC1710->time_values[i]=p_PAC1710->time_values[i-
1]+(float)(p_PAC1710->vsense_stime_10ms/10);


        while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                status_com=PAC1710_read_voltage(&p_PAC1710-
>voltage_reg_values[i]);
                                status_com=PAC1710_calculate_voltage(&p_PAC1710-
>voltage_real_values[i]);
                                p_PAC1710->time_values[i]=p_PAC1710->time_values[i-
1]+(float)(p_PAC1710->vsource_stime_10ms/10);
                                }

                        else{
                                while(avr_counter2<avr){

        while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                        avr_counter2++;
                                        }

        while(conv_done!=1){conv_done=PAC1710_read_conv_done_bit();}
                                status_com=PAC1710_read_current(&p_PAC1710-
>current_reg_values[i]);
                                status_com=PAC1710_calculate_current(&p_PAC1710-
>current_real_values[i]);
                                status_com=PAC1710_read_voltage(&p_PAC1710-
>voltage_reg_values[i]);
                                status_com=PAC1710_calculate_voltage(&p_PAC1710-
>voltage_real_values[i]);
                                p_PAC1710->time_values[i]=p_PAC1710->time_values[i-
1]+((p_PAC1710->vsource_stime_10ms/10)+(float)(p_PAC1710-
>vsense_stime_10ms/10))*avr;
                                avr_counter2=0;
                                }
                        }
                }
        else{
```

```
			status_com=PAC_ERROR;
			return status_com;
		}

		p_PAC1710->counter1=0;

		//Calcule vsource and current values. x_e is the integer part and _d is
the decimal part
		for(uint8_t i=0; i<num_measures;i++){

			voltage_e[i]=(uint16_t)p_PAC1710->voltage_real_values[p_PAC1710-
>counter1];
			voltage_d[i]=((uint16_t)(p_PAC1710->voltage_real_values[i]*100))-
((uint16_t)(p_PAC1710->voltage_real_values[i])*100);

			current_e[i]=(uint16_t)p_PAC1710->current_real_values[i];
			current_d[i]=((uint16_t)(p_PAC1710->current_real_values[i]*1000))-
((uint16_t)(p_PAC1710->current_real_values[i])*1000);

			time_e[i]=(uint16_t)p_PAC1710->time_values[i];
			time_d[i]=((uint16_t)(p_PAC1710->time_values[i]*10))-
((uint16_t)(p_PAC1710->time_values[i])*10);
		}
		//Show by serial port. Format:
		//time1;current1;
		//time2;current2;...

		HAL_UartWriteString("VOLTAGE MEASURES\n");
		for(uint8_t i=0; i<num_measures;i++){
			sprintf(time_data_str,"%i",time_e[i]);
			HAL_UartWriteString(time_data_str);
			HAL_UartWriteString(",");
			sprintf(time_data_str,"%i",time_d[i]);
			HAL_UartWriteString(time_data_str);
			time_data_str[i]=0;
			HAL_UartWriteString(";");

			sprintf(voltage_str,"%d",voltage_e[i]);
			HAL_UartWriteString(voltage_str);
			HAL_UartWriteString(",");
			sprintf(current_str,"%d",voltage_d[i]);
			HAL_UartWriteString(voltage_str);
			time_data_str2[i]=0;
			HAL_UartWriteBytes("\n",1);
		}

		HAL_UartWriteString("\nCURRENTE MEASURES\n");
		for(uint8_t i=0; i<num_measures;i++){
			sprintf(time_data_str2,"%i",time_e[i]);
			HAL_UartWriteString(time_data_str2);
			HAL_UartWriteString(",");
			sprintf(time_data_str2,"%i",time_d[i]);
			HAL_UartWriteString(time_data_str2);
			HAL_UartWriteString(";");

			sprintf(current_str,"%d",current_e[i]);
			HAL_UartWriteString(current_str);
			HAL_UartWriteString(",");
			sprintf(current_str,"%d",current_d[i]);
			HAL_UartWriteString(current_str);
			time_data_str2[i]=0;
			HAL_UartWriteBytes("\n",1);
```

```
	}
	return status_com;
}

/*!
*	@brief This API used to take measures of voltage
*	and/ or current
*	@param measure_time_s: the time in seconds between one shot measures
*	@param num_measures: The number of measures
*
*	@return results of the process
*	@retval 1 -> Success
*	@retval -1 -> Error
*/
int8_t PAC1710_one_shot_measure(uint16_t measure_time_ms,uint16_t
measure_time_s,uint16_t num_measures){
	if (p_PAC1710->flag==PAC1710_NULL) {
		return E_PAC1710_NULL_PTR;		//If the device is not
initialized, return error
	}
		//Initialize variables
		int8_t status_com=PAC_ERROR;
		uint16_t voltage_reg_values[100];
		float time_values[100];
		time_values[0]=0;
		char voltage_str[30] = {0};
		char current_str[30] = {0};
		char time_data_str[10] = {0};
		char time_data_str2[10] = {0};
		float voltage_real_values[100];
		int16_t current_reg_values[100];
		float current_real_values[100];

		for(int i=1; i<num_measures+1;i++){
			if(measure_time_ms>0) PAC1710_wait_time_ms(measure_time_ms);
			if(measure_time_s>0) PAC1710_wait_time_s(measure_time_s);
			time_values[i]=time_values[i-
1]+measure_time_ms+measure_time_s*1000+(float)(p_PAC1710-
>vsource_stime_10ms/10)+(float)(p_PAC1710->vsense_stime_10ms/10);
			status_com=PAC1710_one_shot_mode();
			status_com=PAC1710_read_current(&current_reg_values[i-1]);
			status_com=PAC1710_calculate_current(&current_real_values[i-
1]);
			status_com=PAC1710_read_voltage(&voltage_reg_values[i-1]);
			status_com=PAC1710_calculate_voltage(&voltage_real_values[i-
1]);
			}
		int i=0;
		uint16_t voltage_e[num_measures];
		uint16_t voltage_d[num_measures];
		uint16_t current_e[num_measures];
		uint16_t current_d[num_measures];
		uint16_t time_e[num_measures];
		uint16_t time_d[num_measures];
		for(i=0; i<num_measures;i++){

		voltage_e[i]=(uint16_t)voltage_real_values[i];
		voltage_d[i]=((uint16_t)(voltage_real_values[i]*10))-
((uint16_t)(voltage_real_values[i])*10);

		current_e[i]=(uint16_t)current_real_values[i];
```

```
			current_d[i]=((uint16_t)(current_real_values[i]*1000))-
((uint16_t)(current_real_values[i])*1000);

			time_e[i]=(uint16_t)time_values[i];
			time_d[i]=((uint16_t)(time_values[i]*10))-
((uint16_t)(time_values[i])*10);
			}
			HAL_UartWriteString("VOLTAGE MEASURES\n");
			for(i=0; i<num_measures;i++){
				sprintf(time_data_str,"%i",time_e[i]);
				HAL_UartWriteString(time_data_str);
				HAL_UartWriteString(",");
				sprintf(time_data_str,"%i",time_d[i]);
				HAL_UartWriteString(time_data_str);
				time_data_str[i]=0;
				HAL_UartWriteString(";");

				sprintf(voltage_str,"%d",voltage_e[i]);
				HAL_UartWriteString(voltage_str);
				HAL_UartWriteString(",");
				sprintf(current_str,"%d",voltage_d[i]);
				HAL_UartWriteString(voltage_str);
				time_data_str2[i]=0;
				HAL_UartWriteBytes("\n",1);
			}

			HAL_UartWriteString("\nCURRENTE MEASURES\n");
			for(i=0; i<num_measures;i++){
				sprintf(time_data_str2,"%i",time_e[i]);
				HAL_UartWriteString(time_data_str2);
				HAL_UartWriteString(",");
				sprintf(time_data_str2,"%i",time_d[i]);
				HAL_UartWriteString(time_data_str2);
				HAL_UartWriteString(";");

				sprintf(current_str,"%d",current_e[i]);
				HAL_UartWriteString(current_str);
				HAL_UartWriteString(",");
				sprintf(current_str,"%d",current_d[i]);
				HAL_UartWriteString(current_str);
				time_data_str2[i]=0;
				HAL_UartWriteBytes("\n",1);
			}
			return status_com;
}



/*****************************************************************/
/**\name                        GET INFORMATION FUNCTION
	*/
/*****************************************************************/
/*!
*	@brief This API show the information of the PAC1710 by serial port
*	@param data: the information you want to see
*		0--> All
*		1--> Mode
*		2--> Conversion rate
*		3--> Vsource sample time
*		4--> Vsource average
*		5--> Vsense sample time
*		6--> Vsense average
```

```
*            7--> Vsense range
*      @return results of the process
*      @retval 1 -> Success
*      @retval -1 -> Error
*/
int8_t PAC1710_get_conf_information(uint8_t data){
      if (p_PAC1710->flag==PAC1710_NULL) {
            return E_PAC1710_NULL_PTR;        //If the device is not
initialized, return error
      }
      int8_t status;
            switch(data){
                  case(1):
                  HAL_UartWriteString("\nPAC1710 MODE \n");
                  switch(p_PAC1710->mode){
                        case(0):
                              HAL_UartWriteString("Voltage+ current measure
mode\n");
                        break;
                        case(1):
                              HAL_UartWriteString("Current measure mode\n");
                        break;
                        case(2):
                              HAL_UartWriteString("Voltage measure mode\n");
                        break;
                        case(3):
                              HAL_UartWriteString("Standby mode\n");
                        break;
                        default:
                        status=PAC_ERROR;    //Values not defined returns
error
                        break;
                  }
                  break;
                  case(2):
                  HAL_UartWriteString("\nCONVERSION RATE \n");
                  switch(p_PAC1710->conv_rate){
                        case(0):
                              HAL_UartWriteString("Conversion rate is: 1
conversion per second\n");
                        break;
                        case(1):
                              HAL_UartWriteString("Conversion rate is: 2
conversion per second\n");
                        break;
                        case(2):
                              HAL_UartWriteString("Conversion rate is: 4
conversion per second\n");
                        break;
                        case(3):
                              HAL_UartWriteString("Conversion rate is:
continuous\n");
                        break;
                        default:
                              status=PAC_ERROR;    //Values not defined
returns error
                        break;
                  break;
                  }
                  break;
                  case(3):
                        HAL_UartWriteString("\nVSOURCE SAMPLING TIME \n");
```

```
                        switch(p_PAC1710->vsource_stime){
                                case(0):
                                        HAL_UartWriteString("VSOURCE sampling
time is 2.5 ms\n");
                                break;
                                case(1):
                                        HAL_UartWriteString("VSOURCE sampling
time is 5 ms\n");
                                break;
                                case(2):
                                        HAL_UartWriteString("VSOURCE sampling
time is 10 ms\n");
                                break;
                                case(3):
                                        HAL_UartWriteString("VSOURCE sampling
time is 20 ms\n");
                                break;
                                default:
                                        status=PAC_ERROR;    //Values not defined
returns error
                                break;
                        }
                        break;
                case(4):
                        HAL_UartWriteString("\nVSOURCE AVERAGE \n");
                        switch(p_PAC1710->vsource_avr){
                                case(0):
                                        HAL_UartWriteString("VSOURCE average is
off\n");
                                break;
                                case(1):
                                        HAL_UartWriteString("VSOURCE average is
2\n");
                                break;
                                case(2):
                                        HAL_UartWriteString("VSOURCE sampling
time is 4\n");
                                break;
                                case(3):
                                        HAL_UartWriteString("VSOURCE sampling
time is 8\n");
                                break;
                                default:
                                        status=PAC_ERROR;    //Values not defined
returns error
                                break;
                        }
                        break;
                case(5):
                        HAL_UartWriteString("\nVSENSE SAMPLING TIME \n");
                        switch(p_PAC1710->vsource_avr){
                                case(0):
                                        HAL_UartWriteString("VSENSE sampling
time is 2.5 ms\n");
                                break;
                                case(1):
                                        HAL_UartWriteString("VSENSE sampling
time is 5 ms\n");
                                break;
                                case(2):
                                        HAL_UartWriteString("VSENSE sampling
time is 10 ms\n");
```

```
                                        break;
                                        case(3):
                                                HAL_UartWriteString("VSENSE sampling
time is 20 ms\n");
                                        break;
                                        case(4):
                                                HAL_UartWriteString("VSENSE sampling
time is 40 ms\n");
                                        break;
                                        case(5):
                                                HAL_UartWriteString("VSENSE sampling
time is 80 ms\n");
                                        break;
                                        case(6):
                                                HAL_UartWriteString("VSENSE sampling
time is 160 ms\n");
                                        break;
                                        case(7):
                                                HAL_UartWriteString("VSENSE sampling
time is 320 ms\n");
                                        break;
                                        default:
                                                status=PAC_ERROR;    //Values not defined
returns error
                                        break;
                                }
                                break;
                        case(6):
                        HAL_UartWriteString("\nVSENSE AVERAGE \n");
                        switch(p_PAC1710->vsource_avr){
                                case(0):
                                        HAL_UartWriteString("VSENSE average is off\n");
                                break;
                                case(1):
                                        HAL_UartWriteString("VSENSE average is 2\n");
                                break;
                                case(2):
                                        HAL_UartWriteString("VSENSE sampling time is
4\n");
                                break;
                                case(3):
                                        HAL_UartWriteString("VSENSE sampling time is
8\n");
                                break;
                                default:
                                        status=PAC_ERROR;    //Values not defined
returns error
                                break;
                        }
                        break;
                        case(7):
                        HAL_UartWriteString("\nVSENSE RANGE \n");
                        switch(p_PAC1710->vsource_avr){
                                case(0):
                                        HAL_UartWriteString("VSENSE range is -10 to +10
mV\n");
                                break;
                                case(1):
                                        HAL_UartWriteString("VSENSE average is -20 to
+20 mV\n");
                                break;
                                case(2):
```

```
                                    HAL_UartWriteString("VSENSE average is -40 to
+40 mV\n");
                            break;
                            case(3):
                                    HAL_UartWriteString("VSENSE average is -80 to
+80 mV\n");
                            break;
                            default:
                                    status=PAC_ERROR;    //Values not defined
returns error
                            break;
                    }
                    break;
            default:
                    status=PAC_ERROR;    //Values not defined returns error
                    break;
            }
      return status;
}


#endif //PLATFORM_XPLAINED_PRO_ATMEGA256RFR2
```

b. SCRIPT DE MATLAB

```
tabla_sense=xlsread('Archive_name.xlsx');
tabla_sense(:,1)=tabla_sense(:,1)-tabla_sense(2,1);
tabla_sense(1,:)=[];
time=tabla_sense(:,1);
vsense=tabla_sense(:,2);
% grid on
% plot(time,vsense,'r')
% hold on
% xlabel('Time(s)')
% ylabel('Vsense(mV)')
%-----------------------------
%RANGE=0.02%0.08
%BITS=2046
Rshunt=1;
%FSC=RANGE/Rshunt
%IBUS=FSC*vsense/BITS
IBUS=vsense/Rshunt;
IBUS_mA=IBUS;
%-----------------------------
graphic=plot(time,IBUS_mA);
xlabel('Time(s)')
ylabel('IBUS(mA)')
%-----------------------------
[rows, colums] = size(IBUS_mA);
%C_boton=650; %mAh
C_boton=650*3600; %mAs

integration=zeros(1,2);
for i=1:(rows-1)
    altura=abs(IBUS_mA(i+1)-IBUS_mA(i));
    base=abs(time(i+1)-time(i));
    integration(1)=base*altura;
```

```
    integration(2)=integration(2)+integration(1);
end
Capacidad_consumida=integration(2);
Capacidad_restante=C_boton-Capacidad_consumida;
SoC=Capacidad_restante/C_boton;
Capacidad_consumida
Capacidad_restante
SoC
```

d. CÓDIGO PARA EL ANÁLISIS DE LOS MODOS DE SUEÑO

Se trata de un código de ejemplo que ofrece Atmel para el aprendizaje de uso y programación del microcontrolador Atemaga256rfr2 Xplainded Pro. No se ha incluido en esta sección, al ser propiedad de Atmel. Sin embargo, puede verse en el siguiente enlace a la página propietaria:

http://asf.atmel.com/docs/3.21.0/common.services.basic.sleepmgr.example.atmega256rfr2_xplained_pro/html/index.html

c. CÓDIGO DE PRUEBA DE MODO ACTIVO, SUEÑO Y TRANSMISIÓN

```
int main(void)
{
  SYS_Init();
 // initialize timer
 timer2_init();

 //Use system time to obtain different random values with rand
 srand (time(NULL));

//App main infinite loop
  while (1)
  {
       set_random_time(); //Set random numbers to make the pseudo random time values

       //SLEEP MODE
       do{
       sleep_rutine();//Go to sleep
       }
       while(tot_overflow<sleep_time);   //1000 son unos 4.5 seg
       sleep_disable();                  // Return to active MODE
       tot_overflow=0;                   //Reset timer overflow counter
       timer2_init();                    //Configure timer 2
       RF_send_configuration();          //Configure RF parameters

       //ACTIVE
       // For loop stablish sending time by active_time variable. Each call of
       Hal_Dela(1000) represents 1000ms. Active_time max_value: 60 (1 minute)
       for(uint16_t i=0;i<=active_time*1000;i++){
              HAL_Delay(1000);
        }

       //SENDING
        do{
             RF_rutine();   //Start to send data
        }
             while(tot_overflow<send_time); // stablish sending time by
             //send_time variable. tot_overflow is increased until reach send_time
```

```
        tot_overflow=0; //Reset timer overflow counter
  }
}


// initialize timer 2, interrupt and counter
void timer2_init()
{
       TIMSK2 = 1<<TOIE2;   //enable TC2 overflow interrupt. When reach the maximum
//count, the timer counter will be reset and the overflow interrupt will be called
       TCCR2B = 0x05;       //divide clock by 128. (Prescaler)
       do{} while(ASSR & (1<<TCR2BUB));

       PRR0   |= (0 << PRTIM2);    // Active TimervCounter2 module
       TCNT2 = 0;                 //Reset timer counter
       tot_overflow = 0;          //Reset timer overflow counter
}

// TIMER0 overflow interrupt service routine
// called whenever TCNT0 overflows
void sleep_rutine(void){

              TRXPR = 1 << SLPTR; // disable transceiver
              set_sleep_mode(SLEEP_MODE_PWR_SAVE);     //Set sleep mode: Power Save
              TCNT2 = 0;                               //Reset timer counter
                     do{}while(ASSR & (1<<TCN2UB));    //Waiting for ending to
                                                       //change registers
              sleep_enable();                          //Go to sleep
              sleep_cpu(); // go to deep-sleep (powersave)

}
void RF_send_configuration(void){
         TRXPR = 0 << SLPTR;       // Set RF output
         uint16_t temp;
       //RF configurations
         uint8_t allParam = 0x00;
         temp = (uint16_t)strtol("1234", NULL, 16);
         NWK_SetPanId(temp);
         allParam |= 0x01;
         temp = (uint16_t)strtol("0", NULL, 16);
         NWK_SetAddr(temp);
         allParam |= 0x02;
         PHY_SetChannel(11);
         allParam |= 0x04;
         nwkIb.netControlParam &= (~0x07);
         nwkIb.netControlParam |= NWK_CONTROL_PANCOORDINATOR;
}

void RF_rutine(void){
       APP_TaskHandler();
       SYS_TaskHandler();
       if  ((  nwkIb.addr  ==  APP_ADDR_PANCOORDINATOR  )  &&  (  app_State  ==
APP_STATE_INITIAL_NET_NOT_STARTED )){
              app_commandID = 1;//Time between packets: 1ms
              app_State = APP_STATE_SEND_DEMO;
       }
}
void set_random_time(void){
       //Generate random values
       w=rand() %101;
       x=rand() %101;
       y=rand() %101;
```

```
	z=rand() %101;
	//To set a random time of the 3 assigned to each mode, compare the random
	//values. The comparing routine is different in each case to avoid
	//relationships between time modes
	if(w>=x){
		if(y>=x)
			sleep_time=500;
		else
			sleep_time=800;
	}
	else
		sleep_time=2500;
	if(z>=y){
		if(w>=y)
			active_time=3;
		else
			active_time=1;

	}
	else active_time=10;
	if(w>=z){
		if(z>=x){
			send_time=8000;
		}
		else send_time=4000;
	}
	else send_time=6000;

}
ISR(TIMER2_OVF_vect)
{
	tot_overflow++;

}
```

A2. PLANIFICACIÓN

En el siguiente apartado se hablará sobre la planificación seguida para la elaboración de este proyecto. Se hablará con más detalle del apartado Software, mientras el apartado de Hardware se especifica en [19].

1.: Estudio e investigación del estado del arte y del problema a tratar

- Comprensión de las características de la red de nodos, alimentación→10 días
- Estudio de las distintas formas de medidas de consumo, métodos para disminuir el consumo→10 días

2.: Elección de componentes y fabricación de la parte de hardware necesaria

- Elección del sensor que reúna las características necesarias→ 5 días
- Creación en OrCAD layout y PSPICE de la huella del sensor, esquemático y PCB →10 días
- Fabricación de la PCB, taladrado y soldado de componentes. Pruebas de continuidad eléctrica→ 3 días

3.: Diseño y prueba de los drivers básicos necesarios para la comunicación I2C sensor-nodo

- Estudio de las características de la placa de Evaluación PAC1710/20→ 2 días
- Instalación y comprensión de la herramienta Atmel Studio → 2 días
- Primer diseño de las funcione básicas para la escritura y lectura de los distintos registros del sensor, tratamiento y análisis de datos → 7 días
- Prueba sencilla de la comunicación I2C con las funciones TWI ya implementadas en el código proporcionado →20 días[26]
- Programación y diseño de las distintas funciones necesarias para dar una funcionalidad inicial al sensor →4 días
- Prueba de las funciones básicas de lectura y escritura en la placa de evaluación → 2 días
- Prueba de las funciones básicas de lectura y escritura y la comunicación con la PCB→ 2 días

4.: Toma de medidas. Ampliaciones de código necesarias.

- Búsqueda e implantación del código y montaje necesarios para la medida de los distintos modos de sueño del nodo → 2 días
- Medidas con la placa de evaluación PAC1710/20 de los distintos modos de sueño. Análisis de resultados con Excel y Matlab → 2 días
- Búsqueda e implantación del código y montaje necesarios para la medida del consumo durante la transmisión de información por RF → 2 días
- Medidas con la placa de evaluación PAC1710/20 del consumo durante la transmisión de información por RF → Análisis de resultados con Excel y Matlab → 2 días
- Modificación y mejora de los drivers para la toma de datos → 2 días
- Prueba de los drivers, comunicación y toma de datos con la placa de evaluación y la PCB elaborada → 2 días

5.: Elaboración de la documentación del proyecto

- Memoria→(40 días)
- Diapositivas de presentación→ (10 días)
- Repaso, correcciones y preparación→ (4 días)

[26] Éste ha sido uno de los puntos críticos del proyecto. Se tuvo que crear una nueva PCB, ya que tras varios intentos de establecer comunicación con el microprocesador, la primera se dañó. Esto trajo consigo el aplazamiento de las tareas siguientes.

A3. MARCO REGULADOR

En esta sección se trata de dar una breve explicación de los protocolos y normas relacionados con el proyecto.

a. IEEE 802.15.4

"IEEE 802.15.4 define el protocolo y la interconexión compatible para dispositivos de comunicación de datos que utilizan transmisiones de radiofrecuencia (RF) de corto alcance de baja velocidad de datos, baja potencia y baja complejidad en una red de área personal inalámbrica (WPAN)." [3]

Sus características son:

- Red WPAN de baja velocidad
- Velocidad de datos máxima de 250.000 bits /s
- Potencia máxima de 1 mW.
- Debido a la baja potencia, el rango es de decenas de metros
- Bajo coste de fabricación
- Opera en tres bandas de radiofrecuencia distintas, dependiendo de las regulaciones locales de diferentes partes del mundo:
    - 902 - 928 MHz en Estados Unidos
    - 868 - 868,8 MHz en Europa
    - 2400 - 2483,5 MHz en el resto del mundo

A pesar del corto rango, debido a que usa muy poca potencia y a su bajo coste, esta tecnología resulta muy interesante para redes inalámbricas de sensores en aplicaciones IoT. [3] [4]

b. Atmel lightweight mesh *[5] [6]*

Lightweight Mesh es el protocolo de red de malla inalámbrica de baja potencia de Atmel. Fue diseñado para satisfacer las necesidades de aplicaciones de conectividad inalámbrica, tales como sistemas de alarma y seguridad, automatización de edificios o el control remoto.

Algunas de sus características son:

- Simplicidad de configuración y uso
- Hasta 65535 nodos en una red
- 15 puntos finales de aplicación independientes
- No se requiere un nodo dedicado para iniciar una red
- Dos tipos distintos de nodos:
    - Enrutamiento (dirección de red <0x8000)
    - No enrutamiento (dirección de red ≥ 0x8000). Reciben información y envían datos a cualquier otro nodo
- No hay relación padre-hijo entre nodos
- Huella pequeña (8KB de Flash y 4KB de RAM para una aplicación típica)

Topología de red

La topología de una red típica se muestra en la *Figura A.1.* Los nodos de enrutamiento (azul) forman el núcleo de la red, actuando como extensores del corto rango que tiene el

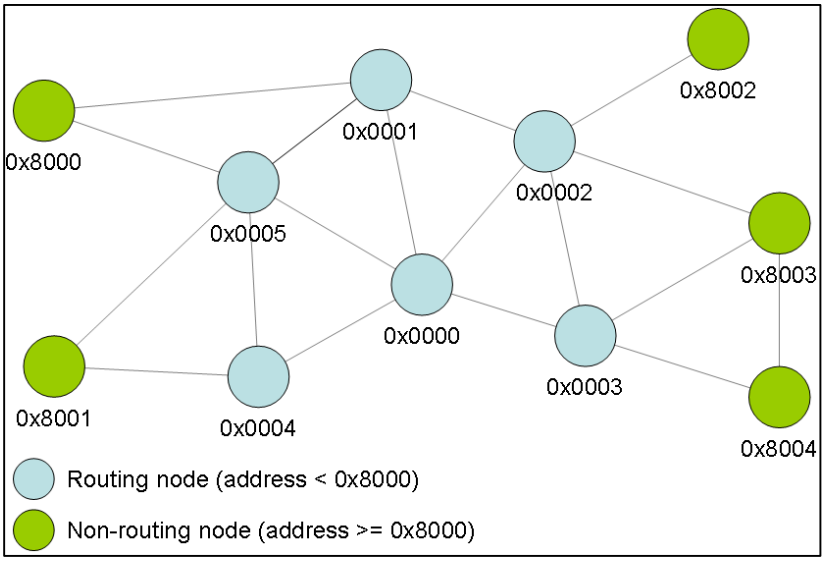

Figura A.1 Topología de red Lightweight Mesh

protocolo. Por otro lado, los nodos de no enrutamiento (verde) se sitúan al borde de la red y se encargan de la comunicación de datos.

ii. Arquitectura

La arquitectura de alto nivel de Lightweight Mesh se presenta en la *Figura A.2*. El código de Lightweight Mesh se divide en un número de niveles lógicos proporcionando cada uno un conjunto de API (*Application Programming Interface*) accesibles. El resto de aplicaciones necesarias deben ser añadidas por el usuario.

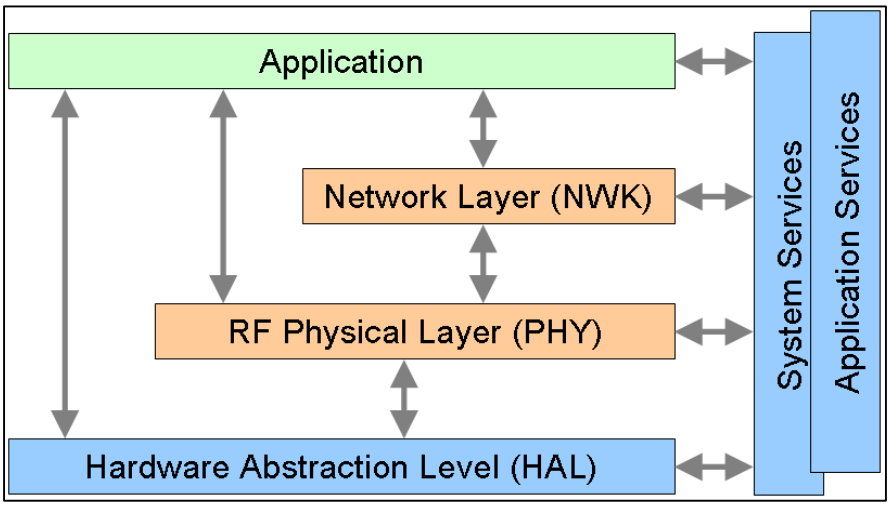

Figura A.2 Arquitectura Lightweight Mesh

La utilidad de cada una se resume a continuación:

- **Capa de abstracción de hardware (HAL)**: proporciona funcionalidad básica dependiente del hardware. Por ejemplo, también se encuentran aquí todas las funciones necesarias para la comunicación I2C.
- **La capa física de radio (PHY):** proporciona funciones para el acceso del transceptor de radio.
- **La capa de red (NWK):** funcionalidad de la pila o "*stack*" central.

- **Servicios del sistema**: funciones comunes para todas las capas, necesarias para el funcionamiento normal de la "*stack*". Aquí se encuentran por ejemplo los temporizadores básicos y los parámetros de configuración por defecto.
- **Los servicios de aplicación**: contiene módulos que pese a no ser imprescindibles para el funcionamiento normal, son típicamente utilizados en la mayoría de las aplicaciones.

c. JTAG y la norma IEEE 1149.1

La interfaz JTAG es un controlador que permite a los dispositivos AVR de Atmel la programación y depuración del código a implementar en el chip. Esta interfaz cumple con el estándar IEEE 1149.1. 95[32]

El estándar ***IEEE 1149.1., Standard Test Access Port and Boundary-Scan Architecture*** (Puerto de Acceso de Prueba Estándar y Arquitectura de Exploración de Límites) define la lógicas de pruebas que pueden incluirse en un circuito integrado para probar las conexiones y el circuito en sí mismo durante el funcionamiento. Además, permite modificar durante la ejecución la actividad del circuito.

Tiene un puerto especial denominado *Puerto de Acceso de Prueba* (**TAP**). Tanto las instrucciones como la información obtenida durante las pruebas se envían por comunicación serie a otro componente, donde se realiza el análisis de los resultados. [37]

d. Protocolo de comunicación I2C

El protocolo de comunicación I2C permite la comunicación serie entre dos dispositivos en modo *Half-Duplex* (los datos fluyen en ambas direcciones pero no a la vez). Necesita de resistencias de Pull-Up para los dos hilos que lo forman SDA o línea de Datos y SCL o línea de reloj.

El número máximo de dispositivos lo determina su capacidad de 400pF y puede operar a distintas velocidades.

Hay un apartado más extenso en la memoria, donde se explica con más detalle este protocolo, en el caso de la aplicación desarrollada: 2.4.5. PROTOCOLO DE COMUNICACIÓN I2C.

e. Reglas de programación en C ¡Error! No se encuentra el origen de la referencia.¡Error! No se encuentra el origen de la referencia.

Este es un conjunto de reglas para la buena programación que se han tratado de seguir en la elaboración de los drivers de este proyecto. No se trata de una normativa regulada, pero considero que es importante tener algún criterio a seguir a la hora de programar, ya que puede ahorrar tanto errores propios como de los demás, y facilitar mucho el trabajo y la legibilidad.

- **Documentación:**
  - Cada función deberá estar documentada en el código. Se ha seguido el criterio de generación de documentación de JavaDoc, que también se usa en C, cuya forma es:

```
/*!
 *  Descripción breve de la función
 *       @param entrada_1 breve descripción
 *       @param entrada_2 breve descripción
 *       @return Valor_de_salida
 *       [adicional: valores típicos de salida]
 */
Valor_de_salida nombre(entrada_1, entrada_2);
```

Esto tiene la ventaja a su vez que el programa de Atmel Studio entiende esta forma de documentación, mostrándola cada vez que se selecciona la función, ayudando así a la correcta llamada de las mismas.

- **Control de errores y compilación:**
  - El código no debe tener warnings, los cuales se tratarán como errores.
  - Todas las funciones tendrán sus prototipos, en los archivos de cabecera (*header*) (.h).
- **Legibilidad:**
  - Las funciones y variables más importantes deben tener nombres con los que sea fácil deducir su utilizad.
  - Las variables que se usen por ejemplo, como contadores, no deben tener nombres extensos.
  - Si están compuestos de varias palabras, usar guiones o mayúsculas para separarlas.
  - Para agilizar la lectura, una función no debe ocupar más de una página, realizando subrutinas en el caso de que sea necesario. Se debe aislar unidades funcionales y codificarlas en funciones aparte aún si esas funciones solamente serán llamadas en un solo lugar. Esto contribuye en gran manera a la lectura posterior del código.

En cuanto al resto de generalidades del lenguaje C, se ha tomado como referencia **¡Error! No se encuentra el origen de la referencia.**.

A4. ENTORNO SOCIO-ECONÓMICO

En este proyecto se busca el desarrollo de una aplicación que permita medir el consumo de energía. Un sistema eficaz de medida de consumo permitiría no sólo el uso eficiente de aquéllos dispositivos que necesiten de una batería, sino también de aquellos que estén conectados a la red.

Conociendo el consumo del dispositivo a medir en cada modo de funcionamiento permitiría optimizar sus funciones reduciendo todos aquellos procesos que pese a no ser fundamentales estén utilizando gran parte de la alimentación. Con ello se conseguiría un uso eficiente de la energía de la red e incluso una mejoría en la duración del dispositivo.

Por otro lado, todo ello cobra mucha más importancia si se requiere el uso de baterías. Sabiendo de forma precisa el consumo en cada instante de tiempo. Ello permite, en primer lugar, elegir la batería adecuada para la aplicación, teniendo en cuenta sus dimensiones, capacidad y perfil de carga/ descarga. En segundo lugar, determinando el punto óptimo en el que la batería puede alimentar el dispositivo sin que se vea dañada, se podría alargar significativamente tanto la vida de la batería como del producto, pudiendo sustituir más tarde la batería y generando a la larga menos residuos. Esto último implica a su vez que se puedan utilizar baterías que sustituyan a otras contaminantes, como la de Ni-Cd, o peligrosas a ciertas temperaturas, como la de ión-litio.

Finalmente, aplicando todas estas mejoras, podrían surgir nuevos productos anteriormente descartados al no ser energéticamente viables.

El impacto que podría tener un buen sistema de medida de consumo en los próximos años, se hace visible con los siguientes datos:

- La consultora Mckinsey estimó el pasado año que para 2025 el potencial de IoT alcanzará entre 3,5 y 9,8 billones de euros (un 11% de la economía mundial en ese momento). Como se había mencionado en la introducción, algunos prevén que en 2020 existan alrededor de 50.000 millones de dispositivos inteligentes. [1]
- Nos encaminamos hacia un mundo más "conectado" en todos los aspectos. *Wearables* enfocados a la salud, elementos del hogar para aportar seguridad, ahorro y comodidad, marketing de proximidad personalizado, drones para mitigar los efectos de la enfermedad de la mosca Tse-Tsé… Muchos de estos dispositivos, se alimentan por medio de baterías. Por un lado, dado su composición, muchas de estas fuentes de energía son contaminantes y generan residuos tóxicos muy perjudiciales para el medio ambiente. Por otro, están limitados a un tiempo de vida dado por su capacidad y condiciones de uso, hasta tal punto que suponen uno de los principales retos a resolver para el avance tecnológico. [1]
- Pese al predicho auge de estas tecnologías, por otro lado, la reducción del consumo es importante en todo elemento eléctrico.

A continuación se muestra un esquema sobre todo lo mencionado anteriormente (*Figura A.3.*):

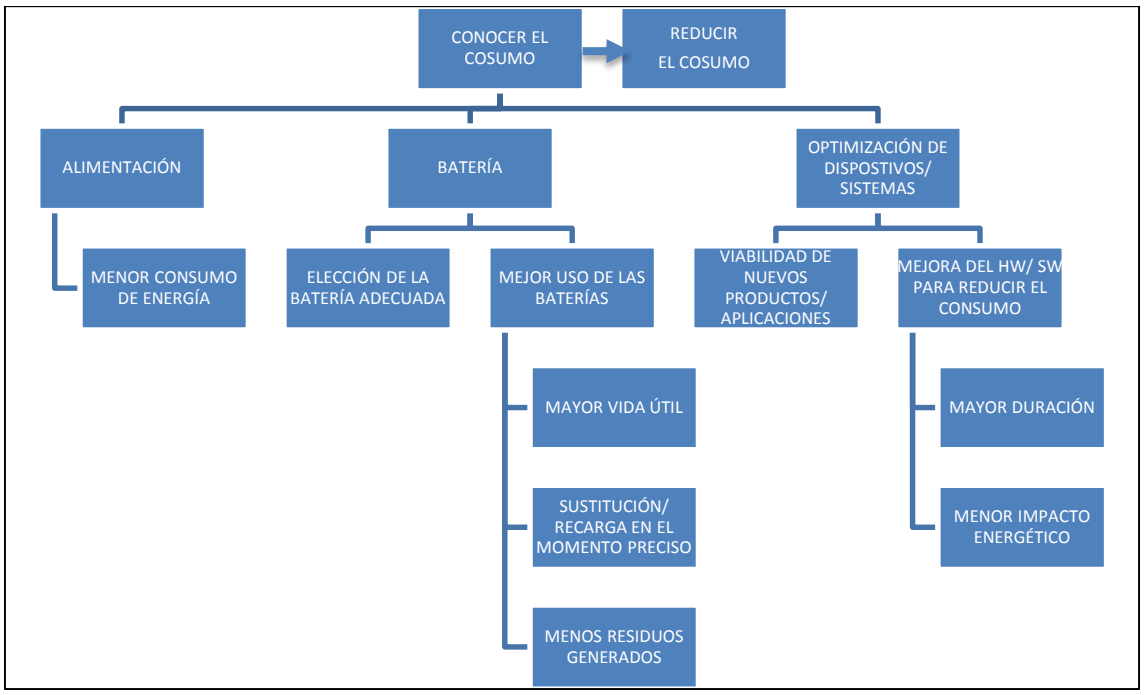

Figura A.3. Esquema sobre los beneficios de un buen conocimiento del consumo

A5. PRESUPUESTO

Las herramientas y recursos utilizados para la realización del proyecto, así como su coste aproximado, se exponen a continuación:

- **Recursos hardware: (referencia de TFG Óscar Escobar Muñóz) [19]**

RECURSOS HARDWARE		
RECURSO	**FUNCIÓN**	**COSTE[27]**
Instrumentación de medida, comunicación puerto serie y otros componente electrónicos	Polímetro, osciloscopio, detector de paquetes de datos (Sniffer), conector UART/USB	1457,00€ **463,34€**
Ordenador Acer Aspire	PC utilizado para el desarrollo del proyecto	900€ **300 €**
Placa de evaluación PAC1710/20	Medida del consumo	**140€**
Microcontrolador ATMEGA256RFR2 Xplained Pro	Nodo sobre el que se realizan las medidas de consumo.	**64.30€**
Diseño completo del prototipo de la PCB, componentes (sensores PAC1710, resistencias, condensadores…)	Conexionado entre los distintos dispositivos	**23€**
SUBTOTAL R. HARDWARE: 2584,3 € (990, 64€)		

Tabla A.1. Presupuesto. Recursos Hardware

[27] En los costes se presenta el coste total arriaba a la derecha y el conste teniendo en cuenta amortizaciones abajo a la derecha. En el subtotal, el valor entre paréntesis es el coste con amortizaciones.

- **Recursos software:**

RECURSOS SOFTWARE		
RECURSO	**FUNCIÓN**	**COSTE**
Licencia OrCAD PSIPCE y Layout	Elaboración de la huella del sensor, esquemático y diseño de la PCB	Licencia de un año: 7000 € Uso (2 meses): **1166,67 €**
Licencia Matlab R2010a	Análisis de los resultados	Licencia de un año: 6000 € Uso (3 meses): **1500 €**
Atmel Studio	Entorno de desarrollo de la aplicación. Programación, depurado y configuración de la placa Atmega256rfr Xplained Pro	**0 €**
Packet Sniffer	Visualización de los paquetes enviados durante la comunicación por RF	**0€**
Termite	Comunicación por puerto serie con el micro	**0€**
Software de Microchip ChipMan	Toma de medidas, análisis del funcionamiento del sensor	**0 €**
Paquete Microsoft (Office, Excel, Power Point)	Realización de la memoria, presentación y cálculos	**0€ (Licencia estudiante gratuita)**
SUBTOTAL R. SOWTWARE: 23000 € (2666,67)		

Tabla A.2. Presupuesto. Recursos Software

- **Recursos humanos:**

RECURSOS HUMANOS		
RECURSO	**FUNCIÓN**	**COSTE**
Estudiante ingeniero electrónico	Elaboración del proyecto	**(300 horas + 200 horas)*20€/hora= 10.000€**
SUBTOTAL R. HUMANOS: 10.000 €		

Tabla A.3. Presupuesto. Recursos Humanos

RESUMEN DE COSTES	
R. HARDWARE	**990,64 €**
R. SOFTWARE	**2666,67 €**
R. HUMANOS	**10.000 €**
SUBTOTAL	**13.657,28 €**
(21% IVA)	**2.868,03 €**
TOTAL	**17.525,31 €**

Tabla A.4. Resumen de costes