Universidad Zaragoza
1542

Trabajo Fin de Máster

Multisensor Wireless

Autor/es

Gabriel Aznar Lapuente

Director/es

Roberto Casas Nebra

Rubén Blasco Marín

Escuela de Ingeniería y Arquitectura

DICIEMBRE - 2016

MULTISENSOR WIRELESS

RESUMEN

El presente Trabajo Fin de Master versa sobre el estudio de la técnica, diseño e implementación de un nodo sensor inalámbrico capaz de capturar imagen, sonido, temperatura, humedad y presencia, de forma autónoma en cualquier estancia del hogar.

La captura de todo el conjunto de datos que el sensor es capaz de capturar, permitirá, mediante un procesamiento individualizado o combinando los datos de varios senosres, definir la actividad que se está llevando a cabo en la estancia del hogar en la que ha sido instalado.

Este nodo sensor también es capaz de enviar los datos capturados a través de un canal de comunicación inalámbrico a un concentrador, el cual se ocupe del tratado de datos o simplemente de almacenarlos. Con este envío de datos se consigue poder tener almacenados estos datos para su posterior consulta, así como poder procesar un conjunto de datos capturados por varios nodos sensores.

Este dispositivo cuenta también con cierta capacidad de procesamiento, así como una buena eficiencia energética con el fin de alargar al máximo posible la vida útil del nodo sensor determinada por la capacidad de la batería. Podrán emplearse técnicas de “energy harvesting” con el fin de poder recoger energía del entorno, como por ejemplo de luz solar, para recargar las baterías.

La finalidad última de este nodo sensor inalámbrico es la monitorización de la actividad en el hogar de personas enfermas con problemas o trastornos de memoria. Con los datos de esta monitorización se pretende ser capaces de detectar que estos enfermos dejan de realizar ciertos hábitos. También servirá para que los enfermos puedan acceder a los datos almacenados de su actividad diaria.

Este Trabajo Fin de Master se enmarca dentro del proyecto de investigación titulado *MEMORY LANE* en el que participa el grupo de investigación HOWLab de la Universidad de Zaragoza. Este proyecto pretende crear un “life-blog” para personas ancianas susceptibles de padecer problemas de memoria, permitiéndoles acceder al contenido del mismo en función de su contexto.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. GABRIEL AZNAR LAPUENTE,

con nº de DNI 73092807A en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) MASTER, (Título del Trabajo) MULTISENSOR WIRELESS,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de NOVIEMBRE de 2016

Fdo: GABRIEL AZNAR LAPUENTE

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1.- Introducción

1. MARCO DE REALIZACIÓN DEL TFM

El presente TFM, *Multisensor Wireless*, se enmarca en el proyecto de investigación *MEMORY LANE*[1], en el que participa el grupo de investigación HOWLab de la Universidad de Zaragoza. Dicho proyecto pretende proveer de una herramienta para realizar, de forma automática y no intrusiva, un "life-blog" para personas ancianas susceptibles de padecer problemas de memoria, permitiéndoles acceder al contenido del mismo en función del contexto.

Una de las tareas de este proyecto es la implementación de una plataforma hardware que permita capturar los datos necesarios para conseguir la finalidad del proyecto.

2. OBJETIVOS

Como objetivo de este TFM, se plantea el desarrollo de una plataforma hardware inalámbrica inteligente, con capacidad de procesamiento; que capture, a priori, temperatura, humedad, sonido, presencia e imágenes.

3. METODOLOGÍA

Para llevar a cabo este trabajo se comenzará por realizar un estado del arte donde se estudien los desarrollos de última tecnología relacionados con el fin del presente trabajo. Una vez realizado este estado del arte, del cual se extraerán las conclusiones sobre cuál es la solución más adecuada que se debe adoptar, se continuará por realizar una búsqueda tecnológica de componentes electrónicos para conseguir dar forma al reto tecnológico planteado.

Con los componentes electrónicos seleccionados, se continuará realizando el diseño del PCB, su fabricación y su posterior montaje de componentes, para continuar con la validación y puesta en funcionamiento del hardware. Una vez comprobado el hardware, se implementara el firmware necesario para que el dispositivo cumpla con las funcionalidades mínimas requeridas

[1] http://howlab.unizar.es/MemoryLane

en la aplicación final del dispositivo. Por último se realizarán pruebas de campo con el fin de validar el funcionamiento global del dispositivo.

4. PLANIFICACIÓN

Se puede dividir el trabajo planteado al inicio de este TFM en cuatro fases claramente diferenciadas:

1. Búsqueda científico-tecnológica para intentar dilucidar cuál es la mejor solución para el reto planteado así como su innovación respecto al estado de la cuestión.
2. Diseño e implementación de un prototipo.
3. Experimentación.
4. Difusión de resultados.

5. ESTRUCTURA DE LA MEMORIA

Se ha considerado oportuno el dividir la memoria en distintos capítulos, los cuales versan sobre temas claramente diferenciados. Estos capítulos siguen el orden cronológico del trabajo realizado, siguiendo la planificación del punto anterior.

- **Capítulo I.- Introducción**: En este capítulo presenta al lector los objetivos del presente trabajo, así como la motivación y la organización del mismo.
- **Capítulo II.- Estado del Arte**: Versa acerca de la consulta científico-tecnológica del estado de la técnica, y el planteamiento de la solución para el reto tecnológico planteado.
- **Capítulo III.- Diseño del prototipo**: En este apartado se aborda todo el trabajo realizado para el diseño y desarrollo hardware y firmware del prototipo.
- **Capítulo IV.- Validación**: Muestra de los ensayos y resultados más relevantes realizados con el prototipo.

Para facilitar la lectura y comprensión del presente trabajo, al inicio de la memoria, se ha incorporado un índice de contenidos, mientras que al final de la misma se ha incluido la bibliografía consultada y, para los lectores menos familiarizados con la temática, un glosario de términos con las definiciones de los más utilizados.

Capítulo 2.- Estado del arte

6. ESTADO DE LA TÉCNICA

Ante el reto de desarrollar un dispositivo electrónico que cumpla con los objetivos propuestos para el presente proyecto, es necesario realizar un estudio del estado de la técnica y las tecnologías existentes en el mercado, a fin de seleccionar la solución que más se ajuste a las necesidades del proyecto.

Examinando la aplicación propuesta se cree conveniente indagar en soluciones de redes inalámbricas de sensores que presenten una fácil instalación y crean poca intrusión en el entorno donde se instalen. En concreto, debe estudiarse lo relacionado con técnicas utilizadas en monitorización de individuos y gestión de la energía disponible en estas redes, así como su implementación.

Las redes de sensores inalámbricas proporcionan una solución de bajo coste y de fácil instalación para desplegar sensores en zonas muy extensas o que se encuentran separadas físicamente. En concreto las redes inalámbricas de sensores de imagen (video o fotografía), suponen una alternativa real a los métodos tradicionales de video-vigilancia o captura de imágenes a distancia que son tecnológicamente bastante complejos y costosos.

El poder usar este tipo de redes de sensores de imagen, supone un importante paso para la creación de entornos inteligentes en los que se busca el reconocimiento de actividades o ciertos patrones mediante la captura de imágenes. El uso de este tipo de sensores puede extenderse en ámbitos tan dispares como la agricultura, el hogar, el tráfico urbano o en carretera, uso militar etc.[1][2]. Las principales ventajas de utilizar una red inalámbrica de sensores, frente a los sistemas tradicionales, son el bajo coste, la escalabilidad, la fácil instalación y el no generar un gran intrusismo en el lugar donde son instalados.

En las redes inalámbricas de sensores, debe prestarse un especial cuidado en el diseño del nodo sensor en cuanto a consumo energético y gestión de la energía se refiere. Esto se debe a que son dispositivos normalmente alimentados por baterías y, por tanto, la energía disponible está limitada. Este punto toma especial relevancia en las redes de sensores inalámbricas de imagen, ya que es una aplicación en la que se precisa de un procesado de datos importante o de transmisión masiva de datos. Ambas tareas se traducen directamente en un consumo energético elevado. Cuando se diseña una red de sensores, se debe poder estimar la vida útil de la red, utilizando para este cálculo la estimación de consumo y la energía disponible en cada uno de los nodos[3].

En cuanto a la gestión de energía disponible, existen dos conceptos principales en los que se pueden basar la estrategia para alargar la vida de la redes de sensores. Por un lado el concepto de eficiencia energética, utilizar la menor cantidad de energía posible, y por otro lado el concepto de "energy harvesting"[4][5], el cual aprovecha la energía que pueda existir en el entorno del nodo para recargar la batería. Con una combinación de ambos conceptos se puede alargar la vida de la red de sensores, e incluso en el mejor de los casos suplir toda la energía consumida por la batería.

Para seleccionar una estrategia de gestión óptima de la energía disponible, en primer lugar se debe valorar el coste energético que supone el procesar una serie de datos y enviarlos, frente a transmitir los mismos datos sin procesar. Una vez obtenida la imagen en uno de los nodos sensores de la red existen tres opciones. En la primera opción, la imagen es procesada y a través de la red se envía únicamente el resultado del procesamiento. En segundo lugar, se puede realizar una compresión de la imagen y para enviarla posteriormente. Y por último, existe la opción de enviar la imagen original sin realizar ninguna operación de procesamiento. A priori, la opción más eficiente es procesar toda la información en el propio nodo, y enviar únicamente los resultados, aunque esta eficiencia dependerá de cómo de complejo sea el procesamiento.

En la comparación del procesado hardware frente al software, en los prototipos consultados, se han conseguido procesados hardware mediante el uso de FPGAs, ASICs o CPLDs[5][6] más eficientes que los procesados software. También el procesado hardware es el procesado más rápido, por lo que se convierte en la solución óptima para aplicaciones en las que el tiempo de respuesta sea crítico. Estos sistemas de procesado tipo hardware, pese en algún caso ser más eficientes que los software, requieren de un gran desarrollo y contienen algoritmos muy complejos, lo que no los hacen apropiados para desarrollos de bajo coste.

Como alternativa al hardware, el procesado puede realizarse a través de software mediante el uso de un microcontrolador de bajo consumo o un DSP[7]. Aunque el tiempo de respuesta empeora respecto al procesado hardware, la principal ventaja de esta opción es que existen librerías y ejemplos de código abierto [8][9] que reducen considerablemente el coste de desarrollo necesario. Podría mejorarse todavía más la eficiencia de estos sistemas optimizando el software. Esta solución es buena para aplicaciones de bajo coste y en las que el tiempo de respuesta no sea crítico.

Otra opción es utilizar una selección híbrida de ambas soluciones, empleando procesado hardware y software conjuntamente. Usando el software no son necesarios desarrollos muy complejos, y usando el hardware para el procesamiento más sencillo se reduzca lo máximo posible el tiempo y el consumo [10][11].

Aunque a priori procesar los datos supone un menor consumo frente a enviarlos, hay aplicaciones en las que se requiere llegar a un compromiso de envío frente a procesado. Esto se debe a la complejidad del procesado que se quiere llevar a cabo, o la naturaleza propia de la aplicación[10][12]. Siguiendo este principio en cada aplicación debe determinarse hasta qué punto puede realizarse el procesamiento de las imágenes en el nodo sensor, teniendo en cuenta también las prestaciones que se hayan seleccionado.

En aplicaciones en las que sea necesario el uso de algoritmos muy complejos la mejor opción será realizar una compresión de la imagen o los datos en el nodo sensor para su posterior envío y tratado en un servidor externo, mientras que en aplicaciones que requieran un algoritmo relativamente sencillo, la mejor opción será realizar el tratado de la imagen en el propio nodo sensor y transmitir datos únicamente de tipo verdadero-falso, o se ha detectado o no una determinada acción.

La forma más drástica de reducir la energía consumida por el nodo sensor es dejarlo de alimentar parcialmente o totalmente en los periodos en los que no se produce ninguna actividad en la escena y por tanto no es necesaria la captura de datos. La complejidad de este método reside en ser capaces de averiguar cuando se ha producido un evento que debe ser monitorizado utilizando la menor cantidad de energía posible.

Figura 1.- Radiación Infrarroja de una persona vista desde una cámara termográfica. Fuente: NASA/IPAC

En aplicaciones donde se pretende monitorizar personas o animales, una buena opción para conseguir esta detección es el uso de Sensores Infrarrojos Pasivos (PIR) [13]. Estos sistemas de muy bajo consumo, en torno a 200 µA, detectan los rayos infrarrojos emitidos por cualquier cuerpo u objeto que tenga una temperatura elevada respecto al entorno. De esta forma, se puede detectar cuando el objeto o persona que se quiere monitorizar entra en escena. Entonces se alimenta el resto de la electrónica y se comienza a adquirir, eliminando así la mayor parte del consumo energético entre evento y evento. En la siguiente figura puede apreciarse la radiación infrarroja de una persona vista desde una cámara termográfica:

Cuando no sea posible la utilización de un sistema PIR debido a que el objeto a detectar no irradia ningún tipo de haz infrarrojo, se puede emplear otra técnica que consiste en el empleo de dos cámaras de distinta resolución [7]. En esta técnica, el nodo sensor usa una cámara de baja resolución para detectar cambios en la escena, y otra de mayor resolución, una vez detectado el comienzo de la acción, para obtener un mayor detalle. El principio de este método es que el procesar una imagen de un tamaño menor implica menor coste energético. Por tanto, se reduce considerablemente el consumo energético.

Otra estrategia posible es un método de adquisición de imagen en el que es la captura se realiza pixel a pixel [14]. Se basa en la eliminación de frames de imagen como tal y la actualización únicamente de los píxeles que varían en la escena, conservando el resto de píxeles de la captura anterior. Se podría decir que consiste en un detector de eventos pixel a pixel. La ventaja de este método es que únicamente se actualizan los píxeles que han sufrido algún cambio reduciendo así el consumo.

7. JUSTIFICACIÓN DE LA SOLUCIÓN ADOPTADA

Dentro del proyecto *MEMORY LANE* es necesario capturar la actividad de la persona afectada por problemas de memoria a fin de conseguir realizar una monitorización continua, así como tener un registro de las actividades y rutinas dentro del hogar. Para intentar determinar los actos y rutinas de estas personas es necesario obtener datos de imagen, sonido, temperatura, humedad y presencia de las distintas estancias del hogar.

Una de las tareas de este proyecto de investigación es desarrollar un hardware que cumpla con las necesidades específicas del proyecto, y haga viable la implantación del sistema completo en cualquier hogar.

En el estudio del estado de la técnica realizado no se han encontrado plataformas de hardware similares que cumplan con todos los requisitos marcados en la aplicación, por lo cual, el presente trabajo pretende desarrollar un diseño electrónico seleccionando las características que más se ajusten. Esto permitirá tener un buen control de la energía disponible, lo que se traduce directamente en una prolongación de la vida útil del nodo sensor. También permitirá el integrar lo mejor posible este nodo sensor en las diferentes formas que se precisan para llevar a cabo la finalidad del proyecto.

Con estos criterios se decide crear una plataforma basada en un microcontrolador de bajo consumo, el cual, además de controlar y adquirir los datos de todos los sensores con un consumo energético mínimo, provea también de una cierta capacidad de procesamiento. A su vez, este microcontrolador también dotará al dispositivo de gran flexibilidad y simplicidad en el procesamiento mediante el desarrollo del firmware a partir de librerías ya existentes.

En cuanto a los sensores, se buscará la solución que tenga la mejor relación entre prestaciones y consumo, teniendo en cuenta una vez más las necesidades y los objetivos del proyecto. En base a los objetivos propuestos, y al estado del arte expuesto en este capítulo el nodo sensor deberá contener los siguientes sensores:

- **Cámara:** capturar imágenes con las que poder identificar la actividad del individuo, el entorno etc. así como guardar los recuerdos en forma de imagen.
- **Micrófono:** capturar el sonido ambiente para reconocer la actividad.
- **PIR:** detectar la presencia de personas en el entorno.
- **Sensores de temperatura y humedad:** reconocer la situación ambiental de temperatura y humedad del entorno con el fin de detectar hábitos o anomalías en el uso de sistemas de calefacción y refrigeración, apertura de ventanas etc.

Capítulo 3.- Diseño del prototipo

1. DISEÑO DEL HARDWARE

Para facilitar la comprensión y el diseño electrónico completo del dispositivo, se ha abordado el diseño del hardware de una forma modular, separando las partes de la electrónica por bloques con una funcionalidad concreta. De esta misma forma, se han separado los componentes en los distintos planos de los esquemas que se adjuntan en el *ANEXO 1: Esquemas PCB.*

Comenzando por los bloques más específicos, se han ido seleccionando componentes y diseñando el circuito electrónico hasta completar las necesidades a cubrir. En base a estos bloques pequeños se ha abordado el diseño de los bloques más generales que alimentan y controlan a todo el dispositivo.

1.1. DIAGRAMA DE BLOQUES

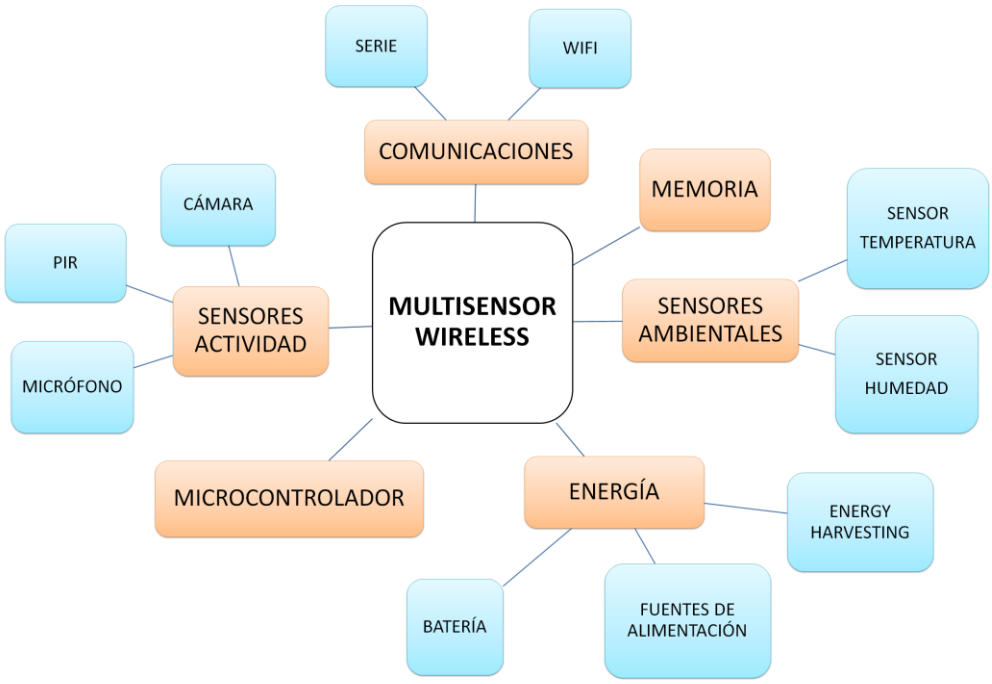


Figura 2 Diagrama de bloques nodo sensor

En el anterior diagrama podemos ver los bloques funcionales que forman el diseño electrónico completo. Las funcionalidades de cada uno de estos son:

- **Sensores de actividad:** se encarga de tomar fotos, capturar el sonido ambiente, y determinar la presencia de personas en el entorno en el que se instale el dispositivo.
- **Sensores ambientales:** se encarga de realizar la monitorización de las condiciones ambientales del lugar en el que esté instalado el dispositivo. A través de él se podrá conocer los datos de temperatura y humedad relativa de la estancia.
- **Comunicaciones:** bloque encargado de la transmisión de datos recogidos por el dispositivo, así como del interface de comunicación necesario para la configuración y el funcionamiento final.
- **Memoria:** se ocupa de almacenar los datos recogidos para su posterior procesamiento o envío.
- **Microcontrolador:** controla todos los periféricos de los apartados anteriores, y procesa la información en los casos en los que sea necesario.
- **Energía:** bloque encargado de alimentar adecuadamente al resto del dispositivo, con las especificaciones de voltaje y corriente necesarios. Además se encarga de recolectar energía del medio en la ubicación donde se coloque el dispositivo y almacenarla para su posterior uso.

1.2. SENSORES DE ACTIVIDAD

1.2.1 SENSOR DE IMAGEN

Se plantea la necesidad de capturar imágenes con una velocidad máxima de 1 foto/minuto. Un aumento en esta tasa de captura de imágenes se traduce directamente en un aumento del consumo energético, sin aportar más información relevante para la aplicación.

Para conseguir dicho objetivo se seleccionado la cámara *LinkSprite JPEG 2M Pixel Color Camera Serial Interface(TTL level)*[15], la cual incorpora una compresión a formato JPEG. La comunicación con esta cámara se realiza a través de UART**¡Error! No se encuentra el origen de la referencia.**. La conexión se realiza a través de un conector JST de 4 pines como puede comprobarse en la siguiente figura:

Figura 3 Detalle frontal de la cámara JPEG. Fuente: www.linksprite.com

Con esta solución, se evita el tener que desarrollar un interface paralelo sincronizado, necesario para una cámara VGA tradicional. Aunque la velocidad de transferencia de datos es mucho menor, se cumple con las necesidades siendo la velocidad máxima marcada por el fabricante de 15 FPS.

Además, al incorporar la compresión JPEG, la transmisión de datos se reduce considerablemente, y una vez guardada la imagen, esta puede ser reproducida fácilmente en cualquier dispositivo que admita JPEG. Como se ha estudiado en el estado de la técnica, reducir el tamaño de los datos a transmitir supone un ahorro energético considerable. Estaríamos ante un ejemplo de compromiso entre procesamiento y envío de datos que permitirá un uso eficiente de la energía empleada para tomar y transferir una foto.

1.2.2 SENSOR DE AUDIO

En el contexto del proyecto, se hace necesario también el poder capturar el sonido ambiental del lugar donde vaya a localizarse el nodo sensor.

No existen especificaciones concretas acerca de las características de sensibilidad que debe presentar el sensor. Sin embargo, al tratarse de sonido ambiente que puede provenir de cualquier dirección, es importante tener en cuenta la direccionalidad [17]. Es fundamental para esta aplicación el utilizar un micrófono omnidireccional, es decir, un micrófono que tenga la misma sensibilidad en todas las direcciones.

Teniendo en cuenta las características requeridas y poniendo la atención en el consumo energético, se ha seleccionado un micrófono de tecnología MEMS, MP33AB01H[16][17], de ST Semiconductor. Este micrófono cuenta con una salida de señal analógica, la cual, deberá ser acondicionada y convertida a señal digital por un conversor analógico digital con la suficiente resolución, y con una frecuencia de muestreo que satisfaga Teorema de muestreo de Nyquist-Shannon0.

Para la adecuación de la señal analógica proveniente del micrófono MEMS se ha utilizado la etapa preamplificadora de audio SSM2167[19] de Analog Devices. Esta etapa cuenta con relación de compresión variable configurable, límite automático anti-saturación y eliminación de ruido ambiente.

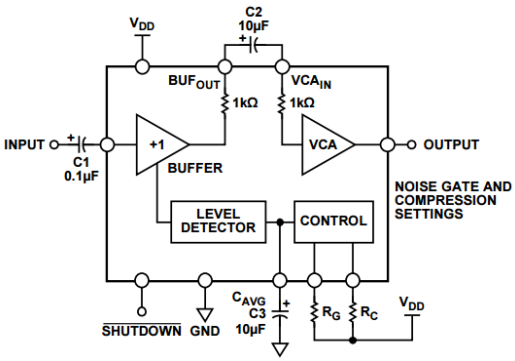


Figura 4 Diagrama de bloques etapa preamplificadora de audio. Fuente: Analog Devices

Con esta etapa se consigue amplificar los sonidos de más débiles, sin llegar a saturar la entrada del conversor analógico digital con los sonidos de mayor intensidad. Esta etapa amplificadora es ideal para la aplicación que se busca, ya que los sonidos provenientes de la actividad del individuo en la mayoría de los casos serán sonidos de poco volumen [20].

1.2.3 SENSOR DE PRESENCIA

La aplicación propia del proyecto requiere de un sensor que indique cuando presencia humana dentro del radio de acción del sensor. Además, teniendo en cuenta las características de bajo consumo que se buscan, se puede utilizar este sensor para dejar de alimentar al resto de la electrónica y alimentar únicamente los dispositivos que sean necesarios, cuando sean necesarios.

Los sensores PIR, detectan la presencia por la radiación infrarroja que emiten los cuerpos que se encuentran a una temperatura distinta a la del resto de la escena. Este tipo de sensores son los utilizados generalmente para detección de personas en sistemas de alarma, encendido automático de luces, etc. Presentan una buena respuesta en cuanto a consumo energético se refiere, en comparación con otros sistemas de detección basados en procesamiento de imagen o tecnologías similares.

Los sensores PIR se caracterizan, principalmente, por el ángulo y distancia máxima de detección. En base a las necesidades del proyecto se ha seleccionado el sensor EKMC1601111 [21] de Panasonic.

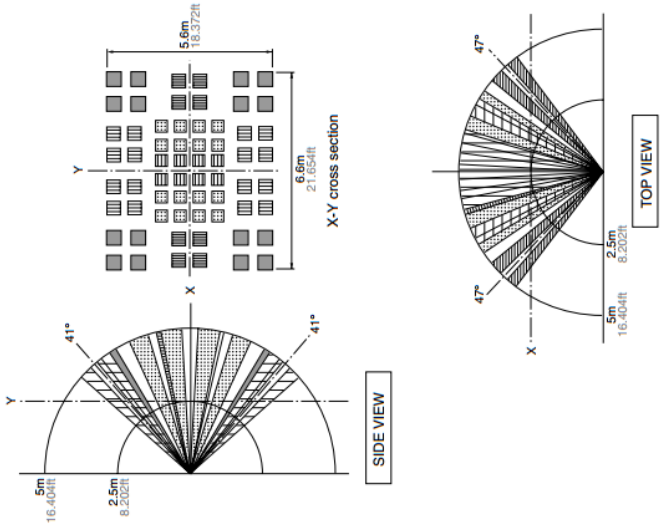


Figura 5 Zonas de detección sensor PIR. Fuente: Panasonic

Como se puede observar en la figura anterior, este sensor PIR tiene un alcance máximo en torno a 5 metros de distancia, con un ángulo de apertura de ± 47º en el plano horizontal y ± 41º en el plano vertical.

1.2.4 RECURSOS DEL MICROCONTROLADOR NECESARIOS PARA CONTORLAR EL BLOQUE DE MONITORIZACIÓN DE LA ACTIVIDAD

Componente	Comunicación	Nº de pines
LinkSprite JPEG 2M	UART (Software Flow Control) Pin Enable	3
MP33AB01H	ADC 12 bits Pin Enable	2
EKMC1601111	Interrupción Externa	1

Tabla 1 Recursos del microcontrolador para la monitorización de la actividad

1.2.5 REQUISITOS ENERGÉTICOS NECESARIOS DEL BLOQUE DE MONITORIZACIÓN DE LA ACTIVIDAD

Componente	Tensión (v)	Consumo max. (mA)	Potencia max. (mW)
LinkSprite JPEG 2M	3.3 o 5	120 (3.3v)	396
MP33AB01H	1.5 – 3.6	0.25 (2v)	0.5
EKMC1601111	3 - 7	0.3	0.99

Tabla 2 Requisitos energéticos para la monitorización de la actividad

1.3. SENSORES AMBIENTALES

Además de los movimientos, imágenes y sonidos provenientes de las personas a monitorizar, existen otras variables como las ambientales de temperatura y humedad que pueden ayudar a determinar la actividad realizada por un individuo. Con estos factores puede determinarse, por ejemplo, si la persona ha dejado una ventana abierta, una estufa encendida, etc.

1.3.1 SENSOR DE TEMPERATURA Y HUMEDAD RELATIVA

Es necesario conocer la temperatura ambiente y la humedad relativa del lugar donde vaya a ser instalado el dispositivo. Para ello se ha añadido un sensor de temperatura y humedad con comunicación digital por bus I2C, el cual, cuenta con linealización y calibración incorporada. El sensor seleccionado es el SHT3x-DIS [22] de SENSIRION.

Este sensor nos ofrece una precisión de +/- 2%RH y +/- 0.3°C. Además, este sensor incluye una alerta configurable, la cual se activa al superarse el umbral configurado en un registro.

El contar con un sensor de temperatura y humedad es un factor a tener en cuenta en la selección de la caja envolvente de la electrónica, así como en la situación de los componentes, ya que el entorno del sensor debe ser el más parecido al ambiente fuera de la caja. Por esto, deberá evitarse el colocar elementos susceptibles de producir mucho calor cerca del sensor, así como seleccionar una caja que cuente con ranuras de ventilación para la libre circulación del aire, de tal manera que la temperatura en el interior de la caja sea lo más parecida posible al exterior y se eviten fenómenos como la condensación.

1.3.2 RECURSOS DEL MICROCONTROLADOR NECESARIOS PARA CONTORLAR EL BLOQUE DE SENSORES AMBIENTALES

Componente	Comunicación	Nº de pines
SHT31	I2C Pin Enable Interrupción	4

Tabla 3 Recursos del microcontrolador para variables ambientales

1.3.3 REQUISITOS ENERGÉTICOS NECESARIOS DEL BLOQUE DE SENSORES AMBIENTALES

Componente	Tensión (v)	Consumo máx. (mA)	Potencia máx. (mW)
SHT31	2.4 – 5.5	0.8 (3.3v)	2.64

Tabla 4 Requisitos energéticos para monitorización de variables ambientales

1.4. COMUNICACIONES

1.1.1. COMUNICACIÓN WIFI

Considerando que la aplicación final del dispositivo va ser su instalación en hogares o zonas habitadas, en la mayoría de las ocasiones, en el punto donde vaya a ser instalado se contará con un punto de acceso WIFI, o la posibilidad de instalar uno fácilmente para contar con conexión directa a Internet.

Figura 6 Ejemplo de red doméstica con punto de acceso WIFI. Fuente: Ok Geet[2]

Tener acceso a Internet a través de una conexión WIFI abre la puerta a multitud de acciones que pueden realizarse consultando o enviando datos de la web. Existen muchas actividades posibles para este proyecto gracias al acceso a Internet como puede ser el almacenamiento directo de datos en la nube, la consulta de la fecha y hora de internet, poder actualizar configuraciones y modos desde cualquier lugar del mundo, etc.

Aunque la tecnología WIFI no es la más eficiente que existe desde un punto de vista energético, sí que puede llegar a conseguirse bajos consumos gestionando de manera eficiente el tiempo de conexión[23].

En base a estas consideraciones, se ha seleccionado como tecnología de comunicación inalámbrica WIFI, para lo que se ha empleado el módulo WIFI – UART RN117 de Microchip [32].

[2] http://okgeek.net/how-to-use-a-WiFi-network-at-home/

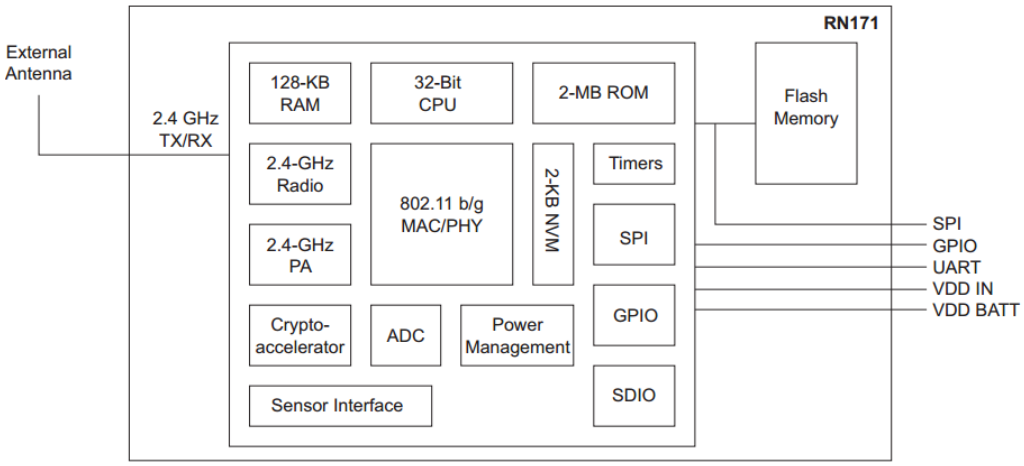


Figura 7 Diagrama de bloques del módulo WIFI RN171. Fuente: Microchip

Este módulo permite distintos protocolos de aplicación como TCP, UDP, DHCP, DNS, ICMP, ARP, cliente HTTP, y cliente FTP. En cuanto a la configuración del punto de acceso WIFI puede ser configurado a través de comandos serie o mediante el uso de Wi-Fi® Protected Setup (WPS).

Cuenta con la opción de conectar una antena cerámica o ruteada en la PCB, o bien la opción de conectar una antena externa a un conector UFL que contiene la PCB del módulo. Microchip ofrece la posibilidad de utilizar la antena PCB desarrollada para su placa de evaluación *RN171 Evaluation Kit*. Esta antena está disponible para descargar junto con las librerías de símbolo y huella para Altium Designer en la web del fabricante[32]. En la siguiente figura pude apreciarse dicha antena en el módulo de evaluación:

Figura 8 Detalle antena PCB placa de evaluación RN171-EK. Fuente: Microchip

En situaciones en las que se cuente con una señal WIFI débil debido a las atenuaciones propias de un hogar, enviando un comando serie al módulo se puede seleccionar la utilización de una antena externa que mejore la señal.

1.4.1 INTERFACE SERIE

A fin de poder contar con un interface sencillo y rápido de configuración, así como para utilizarlo de herramienta de depuración en el diseño del firmware del dispositivo, es conveniente contar con un interface serie que permita la comunicación del dispositivo con un PC.

Para conseguir dicho interface se ha habilitado la salida del puerto USB del microcontrolador así como una UART en el conector auxiliar del dispositivo.

1.1.2. RECURSOS DEL MICROCONTROLADOR NECESARIOS PARA CONTORLAR EL BLOQUE DE COMUNICACIONES

Componente	Comunicación	Nº de pines
RN171	UART (Hardware Flow Control) Pin Enable	5
Interface Serie	UART USB	6

Tabla 5 Recursos del microcontrolador para el bloque de comunicaciones

1.1.3. REQUISITOS ENERGÉTICOS NECESARIOS DEL BLOQUE DE COMUNICACIONES

Componente	Tensión (v)	Consumo máx. (mA)	Potencia máx. (mW)
RN171	3.3v	190(3.3v)	627
Interface Serie	-	-	-

Tabla 6 Requisitos energéticos para el bloque de comunicaciones

1.2. MEMORIA

Es necesario contar una memoria no volátil de suficiente capacidad para poder almacenar los datos provenientes de los sensores. Las tarjetas de memoria SD ofrecen una buena solución para la aplicación que se busca, ya que además de poder contar con una gran capacidad de almacenamiento, permiten su extracción para la lectura o escritura en otro dispositivo externo de manera sencilla. Existen distintos formatos de tarjeta (SD, miniSD y microSD):

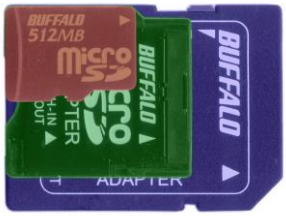


Figura 9 Distintos tamaños de tarjeta SD. Fuente: Wikipedia[3]

Para este caso se ha seleccionado una tarjeta de tamaño microSD. Estas tarjetas cuentan con una comunicación serie SPI a través de la cual se realiza la escritura/lectura de datos. Existen también varias versiones de las tarjetas, las cuales determinan la velocidad de transmisión del bus SPI[24][25][26].

Al tratarse de un elemento que puede conectarse y desconectarse con el dispositivo en funcionamiento, se ha añadido un elemento de protección TVS y ESD en las líneas del bus SPI, con el fin de evitar daños en la electrónica provocados por sobretensiones o descargas electroestáticas.

1.2.1. RECURSOS DEL MICROCONTROLADOR NECESARIOS PARA CONTORLAR EL BLOQUE DE MEMORIA

Componente	Comunicación	Nº de pines
Tarjeta microSD	SPI Pin Enable	5

Tabla 7 Recursos del microcontrolador para la memoria

[3] https://es.wikipedia.org/wiki/Secure_Digital#/media/File:MicroSD_MemoryCard_002.jpg

1.2.2. REQUISITOS ENERGÉTICOS NECESARIOS DEL BLOQUE DE MONITORIZACIÓN DE MEMORIA

Componente	Tensión (v)	Consumo máx. (mA)	Potencia máx. (mW)
Tarjeta Serie	3.3v	45 (3.3v)	150mW

Tabla 8 Recursos energéticos para la memoria

1.3. MICROCONTROLADOR

El microcontrolador es el elemento encargado de dotar de inteligencia al nodo sensor, además de encargarse de gestionar los datos obtenidos de los sensores.

1.4.2 Resumen de características y requisitos necesarios del microcontrolador

Para poder controlar todos los periféricos que se han descrito con las necesidades del proyecto, es necesario seleccionar un microcontrolador cuyas prestaciones encajen con estas. La principal característica de este microcontrolador debe ser el compromiso entre bajo consumo energético y capacidad de procesamiento. A continuación se muestra una comparativa de algunos de los microcontroladores de Atmel:

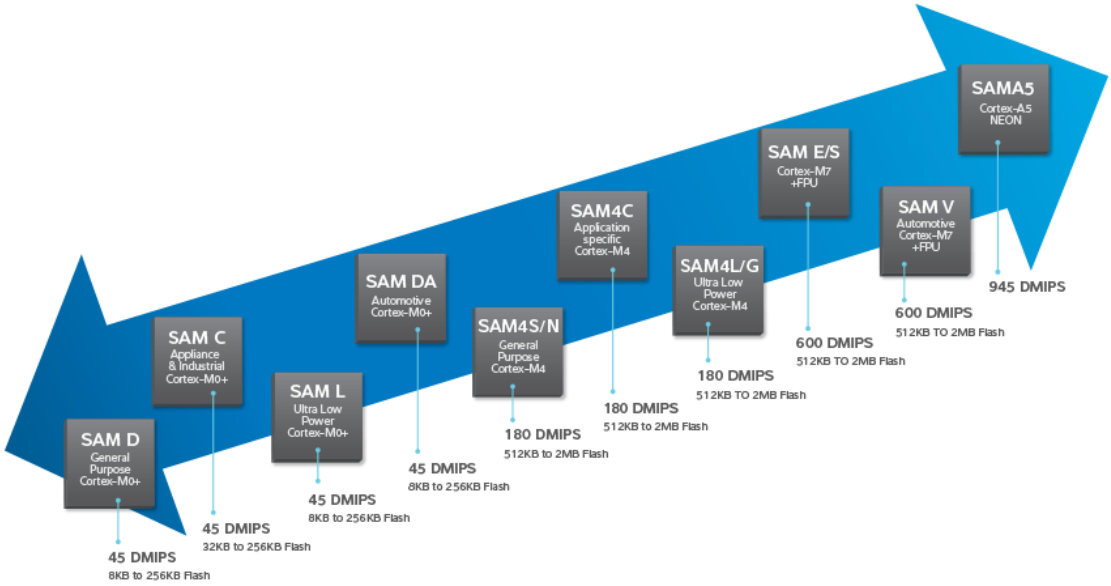


Figura 10.- Comparativa microcontroladores Atmel ARM

Este compromiso entre prestaciones y consumo, permitirá reducir al máximo la energía necesaria para desempeñar la función final del dispositivo, es decir, hacerlo más eficiente. Además, debe cumplir con los requisitos de periféricos necesarios y capacidades necesarias para el manejo del resto de la electrónica.

Tabla 9.- Requisitos necesarios del microcontrolador

Componente	Comunicación	Nº de pines
LinkSprite JPEG 2M	UART (Software Flow Control) Pin Enable	3
MP33AB01H	ADC 12 bits Pin Enable	2
EKMC1601111	Interrupción Externa	1
SHT31	I2C Pin Enable Interrupción	4
RN171	UART (Hardware Flow Control) Pin Enable	5
Interface Serie	UART USB	6
Tarjeta microSD	SPI Pin Enable	5

1.4.3 Microcontrolador Atmel SAM4LC4C

La familia SAM4L de Atmel, basada en arquitectura ARM Cortex®-M4, ofrece unas buenas prestaciones de consumo energético combinadas con la capacidad de procesamiento 1.25 DMIPS/MHz. La frecuencia máxima de operación del microcontrolador es de 48MHz, por tanto, este microcontrolador cuenta con una capacidad máxima de procesamiento de 60 DMIPS. Es una solución óptima para la aplicación propuesta.

Además, Atmel proporciona un entorno de desarrollo gratuito, Atmel Studio, que incorpora librerías HAL (Hardware Abstraction Layer) para este microcontrolador, así como otras librerías que facilitan y reducen considerablemente el tiempo de desarrollo del firmware. Todas estas librerías se incluyen dentro de lo que Atmel denomina Atmel Software Framework[45].

Dentro de la familia SAM4L, la versión que más se ajusta a las necesidades del proyecto es el ATSAM4LC4C[31]:

Requisitos	ATSAM4LC4C
1 x SPI	1 x SPI
3 x UART	4 x UART
1 x I2C	1 x I2S
1 x USB	1 x USB
1 x ADC 12 bits	16 canales ADC 300 Ksps
1 x Interrupción	75 x Interrupción

Tabla 10 Requisitos generales del microcontrolador

En términos de consumo energético, el SAM4LC4 está diseñado para su uso en aplicaciones de muy bajo consumo, consiguiendo un consumo en activo de 90µA/MHz, y pudiendo configurarse modos de apagado de periféricos, RAM y CPU con consumos de hasta 0.9µA según los datos del fabricante.

1.4. ENERGÍA

1.4.1. POTENCIA MÁXIMA

Para estimar el consumo máximo teórico del dispositivo se han sumado todos los consumos máximos que indica el fabricante de los distintos periféricos. Estos consumos no se mantienen durante todo el periodo de uso, sino que se alcanzan en las situaciones donde más energía demanda cada uno de los periféricos. Por ejemplo, el módulo de comunicaciones WiFi alcanza el máximo consumo en la transmisión de los datos.

Componente	Tensión (v)	Consumo máx. (mA)	Potencia máx. (mW)
RN171	3.3v	190(3.3v)	627
Interface Serie	-	-	-
SHT31	2.4 – 5.5	0.8 (3.3v)	2.64
Tarjeta SD	3.3v	45 (3.3v)	150
LinkSprite JPEG 2M	3.3 o 5	120 (3.3v)	396
MP33AB01H	1.5 – 3.6	0.25 (2v)	0.5
EKMC1601111	3 - 7	0.3	0.99
ATSAM4LC4C	3.3	10(3.3v)	33

Tabla 11 Potencia máxima de todos los componentes

Potencia máxima teórica total = 1,21 W

La fuente de alimentación del dispositivo debe ser capaz de suministrar esta potencia de manera continua, pese a que esta situación no debe darse en un funcionamiento normal. Además, a esta potencia máxima se le aplica un factor de seguridad que asegure que nunca se pueda dar un sobreconsumo mayor que la potencia máxima que pueden suministrar las fuentes.

Al tratarse de una aplicación donde la fuente principal de energía va a ser una batería, el voltaje de la fuente primaria va a estar en un valor muy cercano al voltaje final que será necesario para alimentar toda la electrónica. Por esta razón se deben emplear fuentes de baja caída de voltaje salida LDO (Low DropOut). Se considera como dropout la mínima diferencia de voltaje que puede haber entre la entrada y la salida del regulador.

El módulo de comunicaciones RN171 presenta un consumo claramente superior al resto de componentes. Además por la naturaleza de la aplicación este componente no deberá estar alimentado continuamente por lo que se decide dedicar un regulador de tensión LDO exclusivamente para esta parte de la electrónica.

Teniendo en cuenta estas consideraciones, para la alimentación del módulo de comunicaciones se selecciona el regulador REG103[27] de Texas Instruments. Este regulador es capaz de suministrar una potencia máxima de 1,65 W con un dropout de tan solo 115 mV. Esto supone un factor de seguridad superior a 2 con respecto al consumo energético máximo que indica el fabricante del módulo de comunicaciones. Además cuenta con una entrada de "ENABLE" que permite habilitar y deshabilitar el voltaje de salida. En la siguiente figura se muestra el detalle del esquemático:

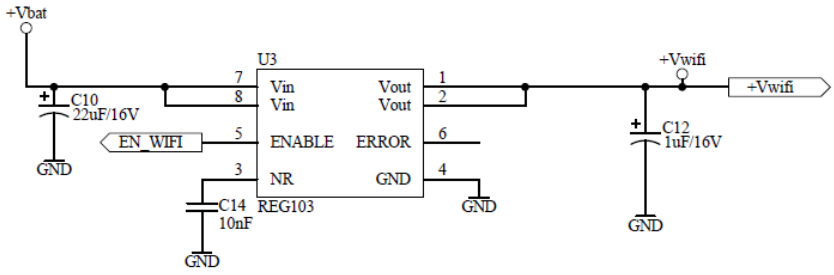


Figura 11 Detalle esquema eléctrico regulador lineal REG103

Para alimentar el resto de la electrónica, incluido el microprocesador, es necesaria aproximadamente la mitad de potencia que la que requiere el módulo de comunicaciones, por lo que utilizaremos un solo regulador de tensión para el resto de electrónica. Nuevamente, teniendo en cuenta todo lo expuesto, se opta por usar el regulador lineal REG102[28] de Texas Instuments. Este regulador es capaz de suministrar 825 mW con 150mV de dropout. En la siguiente figura se muestra el detalle del esquemático:

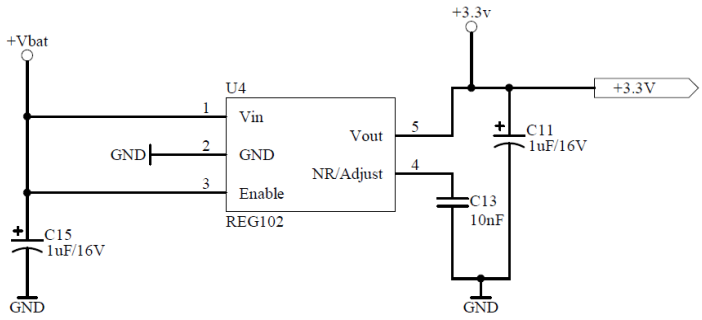


Figura 12 Detalle esquema eléctrico regulador lineal REG102

1.4.2. CONSUMO ENERGÉTICO Y DIMENSIONAMIENTO DE LA BATERÍA

Para el dimensionamiento de la batería ha de tenerse en cuenta el uso inteligente que se va a hacer del dispositivo, es decir, las distintas partes de la electrónica que van a estar alimentadas o no en función de la necesidad de uso. El control de esta activación y desactivación se realizará a través del microcontrolador que será el encargado de decidir que partes deben estar activas en cada momento.

Para conseguir ese control sobre la alimentación de los diferentes dispositivos, en el caso de los componentes de mayor consumo, se ha incorporado electrónica adicional a fin de ser capaces de controlar la conexión y desconexión de alimentación. En el caso de los periféricos de menor consumo como son el micrófono MEMS o el sensor de temperatura y humedad, se han alimentado directamente desde un puerto del microcontrolador. Cuando se quiere habilitar estos periféricos, se configura el pin de salida a nivel alto, mientras que cuando se quiere deshabilitar se pone este pin en alta impedancia con lo que se evita el paso de cualquier corriente.

Además de tener en cuenta las cuestiones relacionadas con el consumo en energético, ha de tenerse en cuenta también otros aspectos como la relación entre: capacidad de la batería, tamaño y precio. Ha de llegarse a un compromiso razonable que haga viable el funcionamiento normal de la aplicación y el coste.

Existen diferentes tecnologías de batería que dependiendo de la aplicación se ajustan más o menos a las necesidades. En el caso de un nodo sensor inalámbrico de estas características, hoy en día la tecnología que más encaja con la aplicación es el ion de Litio (Li-Ion) o Litio-Polímero (LiPo). Estas tecnologías presentan una elevada capacidad energética combinada con un efecto memoria muy pequeño, lo que permite someterlas a un número elevado de ciclos de carga y descarga sin mermar apenas su capacidad[33][34].

Se ha seleccionado las tecnologías Li-Ion o LiPo para el desarrollo de este proyecto, ya que ambas tecnologías pueden ser compatibles al presentar tensiones nominales similares.

En cuanto al dimensionamiento de la capacidad necesaria, es un apartado que debe abordarse una vez se haya depurado por completo el firmware para la aplicación final. En la fase de desarrollo hardware de este TFM, con los datos puramente teóricos de consumo, y sin conocer cuál va a ser la eficiencia energética conseguida con la depuración del firmware, se ha procurado dejar el mayor espacio físico posible para poder albergar la batería.

Otro aspecto que debe tenerse en cuenta en la selección de la batería es que la corriente máxima de pico que puede suministrar, debe ser superior a la corriente máxima de pico que

puede darse en el consumo del nodo sensor. Aun teniendo en cuenta este aspecto, en el diseño se han añadido condensadores de tántalo que permitan una descarga de energía rápida para cubrir la energía necesaria en los picos cortos de mayor intensidad.

1.4.3. ENERGY HARVESTING

Además de conseguir alargar la vida útil del nodo sensor utilizando las técnicas de eficiencia energética, en ocasiones es posible tener un dispositivo con una vida útil ilimitada. Para ello es necesario recargar las baterías con la energía existente en el medio donde se instale. La mayor fuente de energía con la que se va a contar en la mayoría de los casos va a ser la luz, bien proveniente del sol o de fuentes artificiales.

Para esta aplicación se incorpora el cargador de baterías LTC4121-4.2 de Linear Technology [35] orientado a su uso con células solares que cuenta con control de carga (MPPT). Este cargador es compatible con tecnología tanto Li-Ion como LiPo. El control de carga MPPT controla la corriente y el voltaje de entrada, para situarlo al 80% del máximo que es capaz de administrar la célula solar. Este es el punto donde las células solares tienen mayor eficiencia y por tanto de donde puede extraerse la mayor cantidad de energía posible. Además, a esta regulación el MPPT añade un algoritmo que permite corregir las pequeñas variaciones de este punto debido a variaciones puntuales de la radiación lumínica, como pueden ser las provocadas por las nubes. Con todo esto, un cargador con tecnología MPPT se extrae la máxima energía posible de la célula solar.

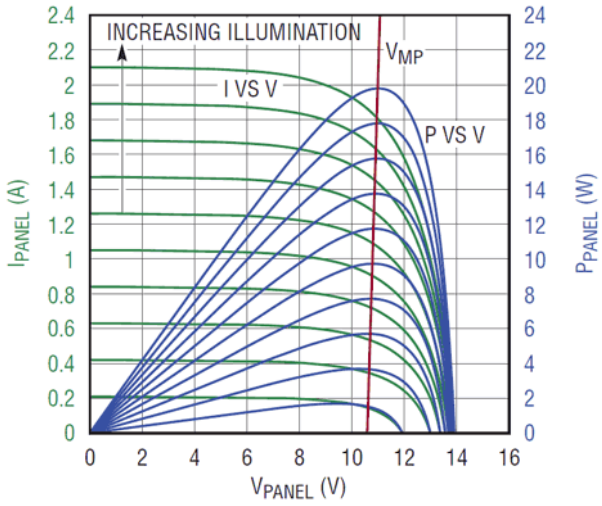


Figura 13 Eficiencia del uso de la técnica MPPT Fuente: Linear Technology[4]

[4] https://learn.adafruit.com/usb-dc-and-solar-lipoly-charger/design-notes

Para las pruebas con el prototipo, y dada la imposibilidad de acceder a una célula solar especial para interiores, se configura el cargador de batería para la validación del hardware con una célula solar de exteriores 9V 150MA SOLAR CELL de Sundance Solar [39].

Las células de interior, como su propio nombre indica, están optimizadas para su utilización en el interior de los edificios donde la radiación solar es menor que en el exterior, y además existen otras fuentes de radiación proveniente de la iluminación propia del edificio. Un ejemplo de estas células son la familia IXOLAR™ HIGH EFFICIENCY SOLARBIT & SOLARMD, las cuales capturan un rango muy amplio de frecuencias, lo que las hace ideales para usos combinados de interiores y exteriores[36].

1.5. DISEÑO DE PCB

Todo el diseño electrónico se ha plasmado en un esquema electrónico que puede consultarse en el *ANEXO 1: Esquemas PCB.*Para la implementación del esquemático, y trasladar este a una placa de circuito impreso se ha realizado un diseño usando el programa Altium Designer. El diseño del PCB ha sido realizado a 4 capas, utilizando los dos planos internos para masa y alimentación respectivamente con el fin de reducir la superficie de PCB necesaria.

Con el objetivo de conseguir una instalación viable del dispositivo en una vivienda, se ha buscado una envolvente de ABS que cumpliese con las necesidades del dispositivo. Tras la selección de esta envolvente, se ha diseñado el contorno de PCB conforme las medidas necesarias para la integración de todo el conjunto.

Además de crear las huellas de cada componente a partir de las especificaciones de cada fabricante, también se han utilizado los modelos 3D de encapsulados con el fin de comprobar viabilidad de la integración de todo el conjunto. En las siguientes figuras puede observarse el modelo 3D de la PCB una vez diseñada:

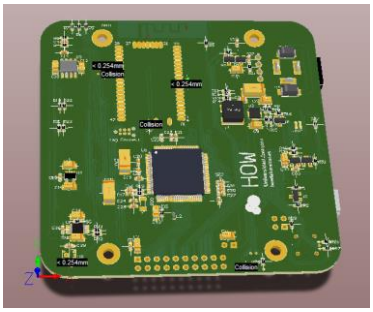


Figura 14 Diseño 3D TOP

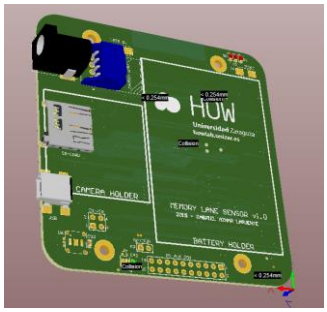


Figura 15 Diseño 3D BOTTOM

En cuanto al trazado de pistas, se ha intentado llevar todas las pistas de la capa TOP en una misma dirección, y las pistas de la capa BOTTOM en sentido perpendicular a las anteriores. Se ha seguido esta técnica a fin de aprovechar al máximo el espacio en la PCB y evitar bucles grandes. Esta direccionalidad de las pistas puede apreciarse claramente en las siguientes figuras:

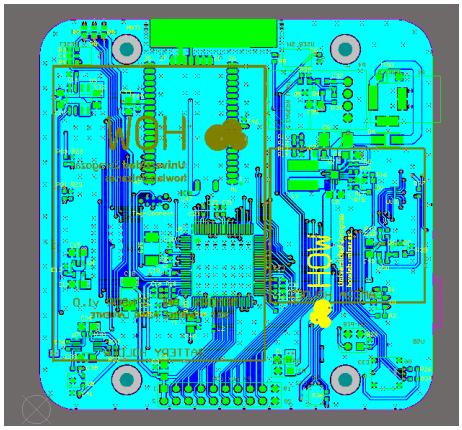


Figura 16 Diseño 2D TOP

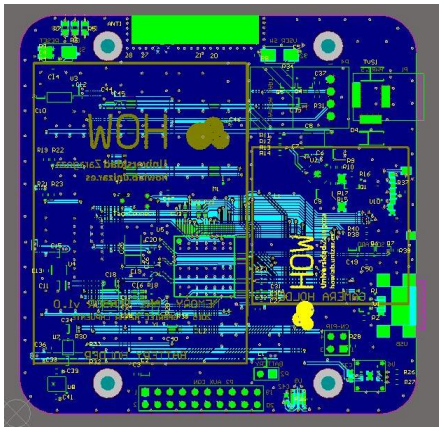


Figura 17 Diseño 2D BOTTOM

Para reducir al máximo el espacio de PCB necesario se ha sustituido el conector de depuración típico de un microcontrolador ARM 20 pines por un conector de espacio reducido de 6 pines que utiliza un protocolo SWD (Sigle Wire Debug) para la depuración del microcontrolador. Este conector del fabricante Tag-Connect [37] sigue una filosofía de conexión tipo cama de pinchos, en la cual la PCB no precisa de un conector de entrada sino que se disponen una serie de pads de cobre limpio sobre los que se apoyan los pinchos del conector de depuración.

Tras la puesta en funcionamiento del hardware, se han detectado distintos fallos de diseño, los cuales han sido corregidos sobre la misma PCB cortando y cortocircuitando pistas. Estos cambios están reflejados al final del *ANEXO 1: Esquemas PCB.*

2. DESARROLLO DEL FIRMWARE

2.1. INTRODUCCIÓN

Para poder realizar el control de todos los periféricos, dotar de inteligencia, y gestionar de la manera más eficiente la energía, se debe desarrollar un firmware para el microcontrolador SAM4LC4.

Dependiendo de los requisitos y del nivel de desarrollo de la aplicación final del proyecto MEMORY LANE este firmware deberá hacer un uso inteligente de los sensores, la transmisión de datos, el procesamiento de datos o los modos de bajo consumo del microcontrolador; para conseguir la mayor eficiencia energética posible, y alargar así al máximo la vida útil del nodo sensor.

El desarrollo del firmware orientado a conseguir dicha eficiencia energética, es un punto que requiere de un estudio extenso para conseguir unos buenos resultados.

El objetivo de este TFM es un programa más simple, que ejecute las funciones básicas para el proyecto *MEMORY LANE*, aplicando algoritmos de bajo consumo simples, con el objeto de demostrar la viabilidad de utilizar la plataforma hardware desarrollada. Este firmware debe ser capaz de capturar datos de todos los sensores incluidos en el prototipo así como de almacenarlos en la memoria SD y enviarlos a la nube.

Para el desarrollo de este programa se han utilizado como base las librerías ASF de Atmel. Como entorno de desarrollo y depuración se ha utilizado el software gratuito Atmel Studio 6.2.

2.2. Estructuración del código

Desde el principio del desarrollo, y siguiendo la filosofía de las librerías de abstracción hardware de las que provee el ASF de Atmel, se ha estructurado el código en varias capas de abstracción, separando las librerías por cada uno de los periféricos utilizados.

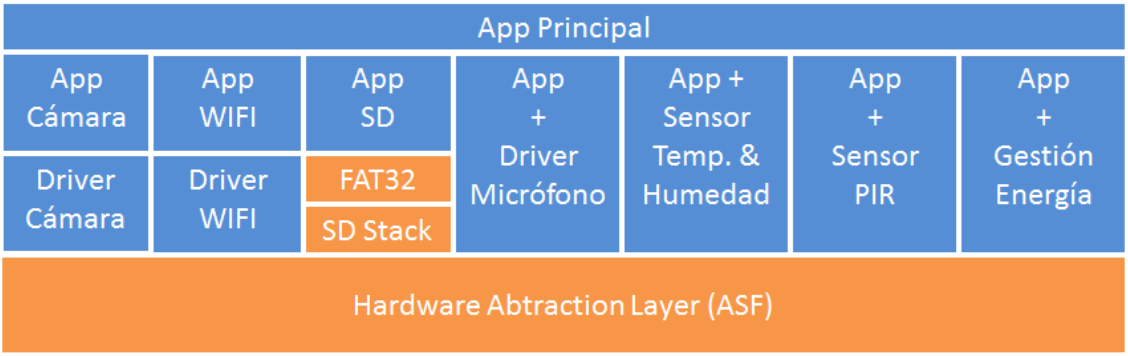


Figura 18.- Diagrama de capas del firmware

En la *Figura 18.- Diagrama de capas del firmware* se muestran las distintas capas en las que se ha organizado el firmware, y según el color podemos distinguir dos tipos de librerías:

- En color azul aparecen las librerías desarrolladas expresamente para el proyecto.
- En color naranja aparecen las librerías contenidas en el ASF de Atmel, las cuales únicamente han tenido que ser adaptadas para el hardware.

También puede observare como las librerías se organizan en distintas capas según el nivel de abstracción, de menor a mayor nivel:

- **HAL:** provee de las funciones para modificar los valores de los registros del microcontrolador de manera más sencilla.
- **Driver:** configura los periféricos del microcontrolador para realizar correctamente las comunicaciones u obtención de valores desde los sensores.
- **App:** provee de todas las funciones necesarias para configurar los sensores y obtener los datos provenientes de estos. Utiliza funciones de la capa de drivers o de otra librería App.
- **App Principal:** contiene el algoritmo para gestionar la activación o desactivación de sensores, así como envío y gestión de datos. Utiliza las funciones de las App.

En los casos de librerías en las que las capas de *"driver"* y "*app"* tienen un tamaño reducido, se ha decido unir ambas en un solo fichero con el fin de simplificar su uso. Este es el caso del micrófono, el PIR o el sensor de temperatura y humedad.

Se puede consultar el resumen de las funciones desarrolladas en el *ANEXO 2: Resumen de las funciones implementadas en cada librería.*

2.3. FUNCIONALIDAD DEL FIRMWARE DEL PROTOTIPO

El firmware del prototipo contiene dos modos de funcionamiento claramente diferenciados:

- **Modo disparo por detección:** cuando se detecta presencia a través del sensor PIR en el entorno del sensor, se alimentan los sensores y se realiza una captura de todas las magnitudes (imagen, sonido, temperatura y humedad). Estos datos son almacenados en la memoria SD, y, transcurrido un tiempo configurable desde el último volcado de datos en la nube, o si se ha superado un determinado número de archivos almacenados, se comienza a transferir estos archivos por WIFI a un servidor FTP.
- **Modo Time-Lapse:** En este modo se realiza una captura continua, separando las capturas un tiempo configurable a partir de un minuto. Los datos se almacenan en la memoria SD y a continuación se transfieren al servidor FTP antes de la siguiente muestra.

En el siguiente apartado se muestra un diagrama de flujo, seguido de su explicación, donde ser representa recorrido de todo el programa principal.

2.4. DIAGRAMA DE FLUJO DE LA APLICACIÓN PRINCIPAL

En este apartado se pretende realizar una explicación detallada del diagrama de flujo de la aplicación principal representado en la *Figura 19.- Diagrama de flujo de la aplicación principal,* el cual sigue la siguiente secuencia:

Al inicio del programa, bien sea por que se acaba de alimentar el dispositivo o porque se realiza un reseteo, se procede a realizar la configuración inicial del hardware.

Una vez que se ha realizado toda la configuración inicial, o se ha comprobado que el hadware ya está configurado mediante la lectura del archivo de configuración correspondiente contenido en la tarjeta de memoria, se comprueba cuál es el modo de funcionamiento configurado, y dependiendo de este se sigue un camino u otro. A partir de aquí se presentan dos caminos claramente diferenciados:

- Si se ha seleccionado el modo 0 (Disparo por detección):

  En este estado se comprueba si hay presencia de personas en el entorno (PIR). Si el resultado es positivo se realiza una captura de todos los sensores y se almacena en la memoria SD.

  También se comprueba si el número de archivos sin transferir al FTP es mayor que el máximo permitido o si el tiempo desde la última subida de archivos es mayor que el máximo configurado. Si alguna de estas dos condiciones se cumple se procede a realizar el envío de los archivos al servidor FTP.

  Realizados estos pasos se regresa a la comprobación del modo de funcionamiento.

- Si se ha seleccionado el modo 1 (Time-Lapse):

  Se comprueba si el tiempo transcurrido desde la última captura es mayor al tiempo de Time-Lapse programado. Si se cumple esta condición se procede a realizar una captura de todos los sensores y almacenar los datos en la memoria SD. Una vez almacenados los datos en la memoria SD, se procede al envío de estos al servidor FTP.

Llegados a este punto se regresa a comprobar el estado en el que está configurado el dispositivo.

El paso del modo 0 al modo 1 y viceversa se realiza por interrupción, mediante el accionamiento del pulsador USER del dispositivo. Al accionar este pulsador, hace saltar al programa a una interrupción externa y cambia el valor de la variable modo.

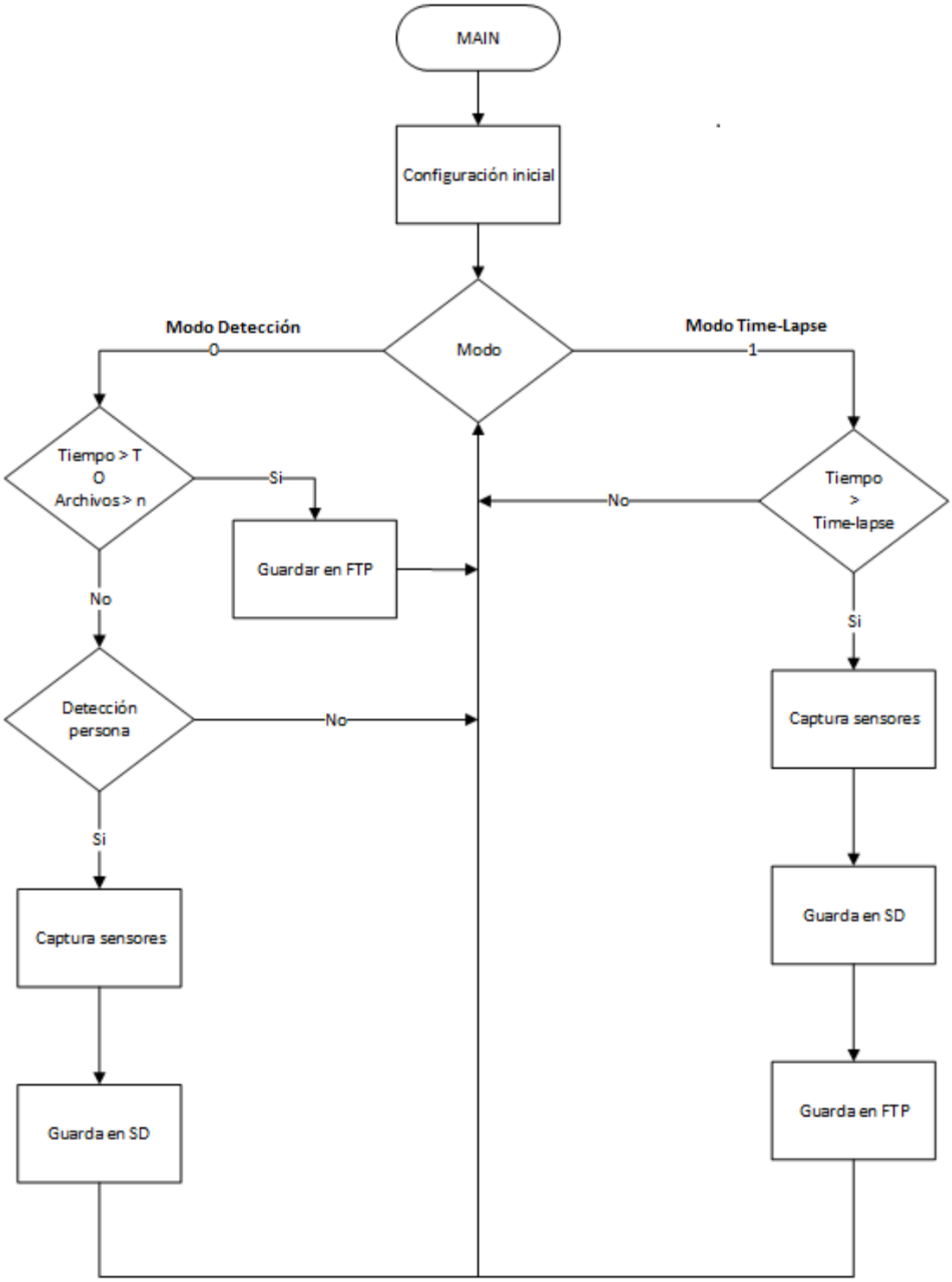


Figura 19.- Diagrama de flujo de la aplicación principal

2.5. PROCESOS FUNCIONALES PRINCIPALES

2.5.1 OBTENER EL NOMBRE DE CADA ARCHIVO

Con el objetivo de identificar fácilmente la proveniencia de los archivos en el servidor FTP, así como ordenar dichos archivos cronológicamente, el nombre de cada uno de los archivos está formado por la dirección MAC del dispositivo que ha capturado los datos seguido del último dato de TimeStamp obtenido desde el servidor SNTP configurado. Al final del nombre del archivo, antes de la extensión, se incluye un valor numérico de cuatro cifras que diferencia los archivos que contenga un mismo TimeStamp, es decir, que se hayan obtenido entre conexión y conexión a internet. Seguido a este número, se incluyen 3 caracteres que diferencian el tipo de datos que contiene el archivo. Por último la extensión del tipo de archivo en la que se han guardado los datos.

MAC												TIMESTAMP										CONT.				TIPO			EXTE.			
X	X	X	X	X	X	X	X	X	X	X	X	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Z	Z	Z	Z	T	T	T	.	E	X	T

Figura 20 Composición del nombre de los archivos.

Los campos que componen el nombre del archivo y su significado son los siguientes:

- **MAC:** dirección única de cada nodo sensor de 12 cifras hexadecimales.
- **TIMESTAMP:** 10 cifras decimales con el número de segundos desde el 1 de enero de 1970.
- **CONTADOR:** 4 cifras decimales, las cuales indican el número de archivos guardados desde la última actualización de TIMESTAMP.
- **TIPO:** 3 caracteres que indican de datos almacenados en ese archivo:
    - PIC: foto tomada por la cámara.
    - AUD: audio grabado con el micrófono.
    - THP: temperatura, humedad y estado del PIR.
- **EXTENSIÓN:** extensión del tipo de archivo en el que son guardados los datos.

2.5.2 CAPTURAR UNA IMAGEN Y ALMACENARLA EN LA MEMORIA SD

Para realizar la captura de una imagen es necesario enviar una serie de comandos serie a la cámara, donde además de indicar que se realice la captura, se indique también de qué manera se quieren recibir los datos a través de la UART. La secuencia completa de este proceso se muestra en el flujograma representado en la ***¡Error! No se encuentra el origen de la referencia.***. A continuación se comentan diferentes detalles sobre este flujograma:

La cámara JPEG LinkSprite cuenta con una memoria RAM donde almacena la última fotografía capturada. Mientras la cámara permanezca alimentada estos datos permanecen retenidos en la memoria. Debido al gran tamaño de datos de la fotografía es necesario copiar los datos que ser recibe por la UART directamente en la memoria SD. Para el prototipo se ha determinado que las fotos tengan una resolución de 640x480 píxeles, ya que este sería un tamaño adecuado para la aplicación final. Se podría utilizar tamaños mayores pero esto se traduciría en un mayor número de datos y por tanto un mayor consumo energético.

Teniendo en cuenta estas consideraciones, una vez capturada la foto, se continua por abrir un archivo con extensión ".jpg" en la memoria SD en el cual van a ser copiados los datos. Una vez abierto el archivo se comienza a recibir datos y a transferirlos directamente a la memoria SD. El final del archivo viene marcado por el dato "*0xFFD9*", de tal forma que continuamente se comprueba que los datos recibidos no coinciden con este comando para seguir recibiendo datos. En el momento en el que se detecta el comando de fin de archivo, este se copia en la memoria SD y se cierra el archivo, completando así el proceso.

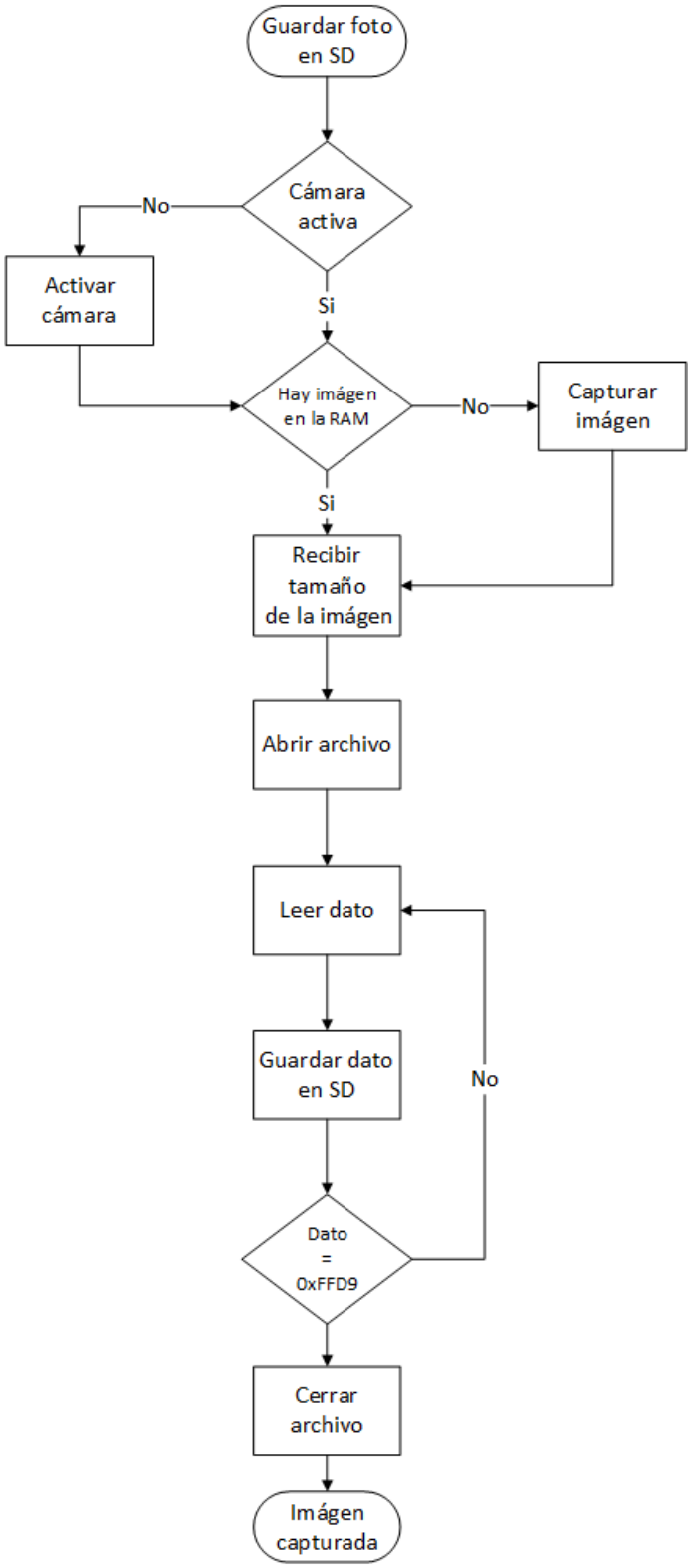


Figura 21.- Diagrama de flujo captura de imagen

2.5.3 CAPTURAR UN CLIP DE AUDIO Y ALMACENARLO EN LA MEMORIA SD

Para capturar un clip de audio debe ponerse el conversor analógico-digital a funcionar de una manera continua, de forma que se obtenga un dato de conversión a una frecuencia constante. El espectro audible por un humano sano va desde los 20Hz hasta los 20KHz [38]. Como ya se ha comentado en el apartado de diseño del hardware, para capturar este espectro de frecuencias debe satisfacerse la el criterio de Nyquist0, por lo que la frecuencia de muestreo deberá ser al menos de 2 veces la frecuencia máxima que se deseé capturar. La frecuencia mínima configurable en el modo continuo del conversor es 75KHz.

El tiempo necesario para almacenar un dato en SD es superior a la frecuencia necesaria para poder capturar el sonido por lo que estos datos deben ser almacenados en una memoria de acceso rápido como es la RAM del propio microcontrolador. Esto supone una limitación importante en el tamaño de datos que se pueden capturar, ya que la memoria SRAM del microcontrolador es de 32kB. Con este dato se decide reservar aproximadamente 20kB para el archivo de audio, de manera que se pueda almacenar un clip e audio compuesto por 20.000 muestras del conversor.

Teniendo en cuenta el tamaño de memoria disponible, se hace totalmente inviable el capturar a frecuencia necesaria para capturar señales de 20 KHz ($f_m > 40KHz$), ya que la duración del clip de audio sería muy pequeña. Para solventar este problema, se descartan el rango de frecuencias superiores a 4.096 Hz, ya que en ese rango el oído humano tiene menor sensibilidad y corresponde con señales de sonido menos frecuentes.

Con estas consideraciones se decide establecer la frecuencia de muestreo en 12,5 KHz, lo que supone con una resolución de 8 bits del conversor una tasa de datos de 100Kbps. Teniendo en cuenta el tamaño de los datos del conversor y la tasa de datos, la duración del clip de sonido viene determinada por la siguiente formula:

$$T(s) = \frac{Memoria(bytes)}{F_m(Hz)} = \frac{20.000}{12.500} = 1,6s$$

Para conseguir dicha frecuencia de muestreo, y dado que la frecuencia mínima de muestreo en modo continuo es muy superior, se decide almacenar en un buffer 6 captaciones del conversor a una frecuencia de muestreo de 75 KHz, y una vez se tienen los 6 datos realizar la media de estos para almacenar el resultado como un valor del archivo de audio. De esta forma se almacenan los datos con una frecuencia de 12,5 KHz.

Una vez capturados los datos y guardados en la memoria RAM se procede a transferir estos a un archivo en la memoria SD. Este proceso se refleja en el siguiente diagrama de flujo:

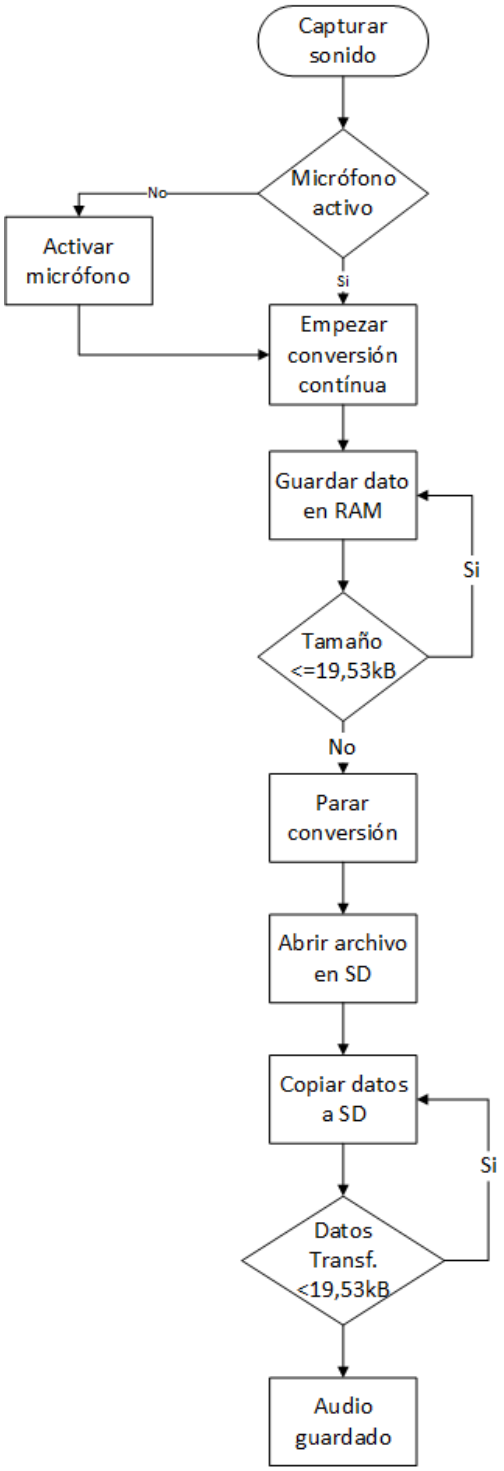


Figura 22.- Diagrama de flujo captura de sonido

2.5.4 TRANSFERIR ARCHIVOS AL SERVIDOR FTP

Para transferir los archivos al servidor FTP, en cualquiera de los modos de funcionamiento, estos archivos deben estar almacenados en la memoria SD. El inicio de esta transferencia depende del modo de funcionamiento. En el modo disparo por detección, puede ser o bien por superar un determinado número de archivos sin transferir, o por transcurrir un tiempo máximo sin transferir archivos, mientras que en el modo "Time-Lapse" la transferencia se realiza justo después de capturar los datos. Para comprender mejor este proceso se ha incluido el diagrama de flujo representado en la *Figura 22.- Diagrama de flujo captura de sonido.*

Al iniciar el proceso de transferencia se comprueba que el archivo que se pretende trasferir existe en la memoria SD, si este punto se confirma se continua por realizar una petición para crear un archivo en el servidor FTP. Una vez confirmada la conexión con el servidor se espera a que el módulo RN171 envíe el comando que indica que se ha abierto el archivo en el servidor. Una vez abierto el archivo, el módulo RN171 comienza a copiar todo lo que recibe a través de la UART en el archivo abierto hasta que transcurra un tiempo determinado en la configuración sin recibir ningún dato. Cuando transcurre este tiempo, el módulo RN171 envía un comando que indica que se ha cerrado el archivo, y vuelve a su estado de recepción de comandos.

En cuanto se recibe el comando de apertura de archivo en el servidor FTP se comienza a leer los datos contenidos en el archivo de la memoria SD y al envío de estos datos a través de la UART.

Una vez que se detecta el final de archivo, bien sea por el tamaño del archivo, o, como en el caso de las imágenes JPEG por la detección de un determinado dato, se espera a confirmar que el archivo ha sido cerrado para evitar continuar grabando en el archivo datos no deseados.

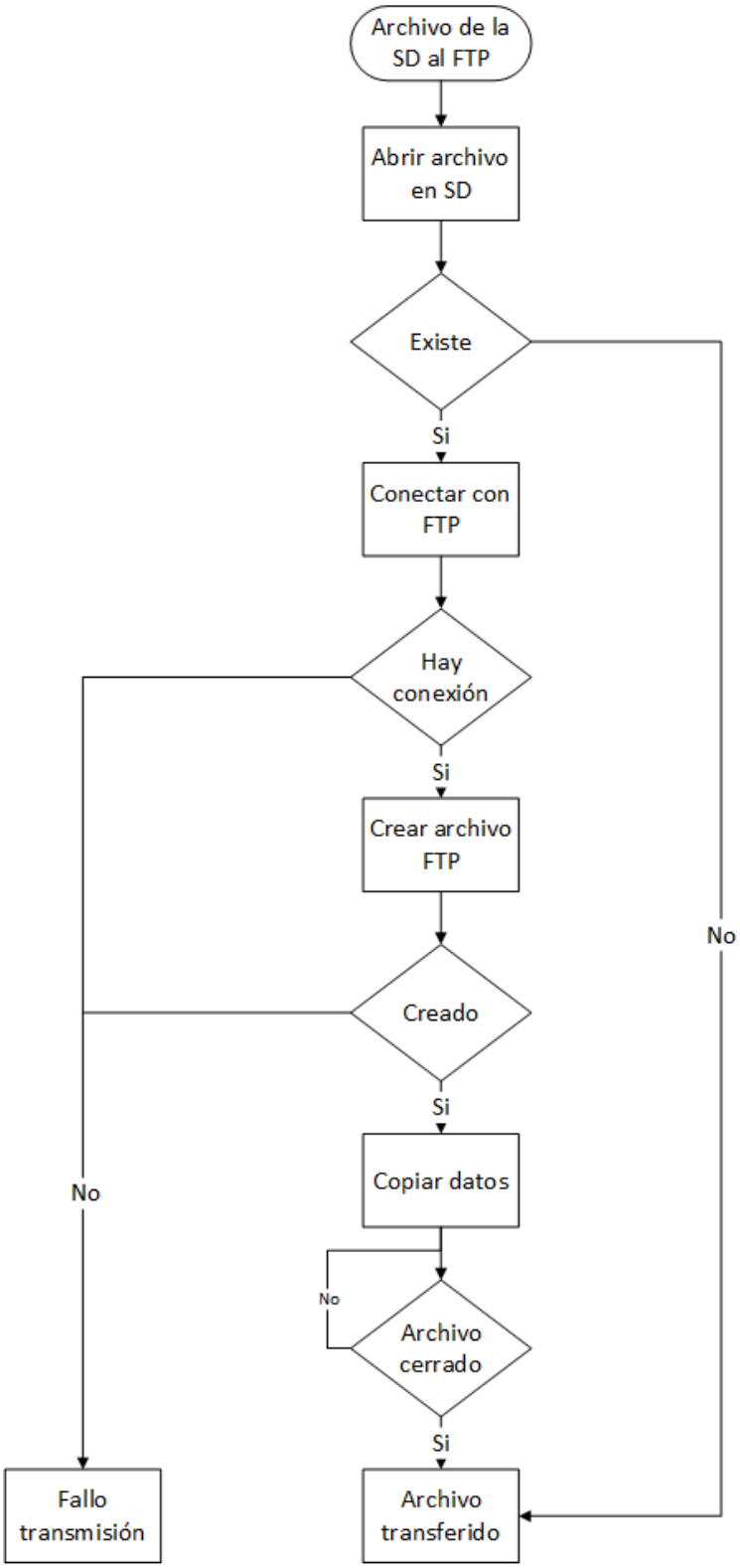


Figura 23 Diagrama de flujo transferir datos al servidor FTP

2.5.5 CONTROL DE CONSUMO

Como algoritmo de bajo consumo del prototipo se ha estipulado que los sensores y demás partes del hardware sobre las que se tenga control de su alimentación deben permanecer alimentados solamente el tiempo necesario preciso para completar las acciones necesarias.

Para conseguir esto de la manera más eficiente, evitando en todo momento que el microcontrolador quede en un punto del programa esperando a completar un proceso que no precisa de su intervención, se han utilizado, en los casos en los que ha sido posible, las interrupciones propias de cada periférico para controlar el flujo de datos. Esto permite que el microcontrolador siga con la ejecución del resto del programa y así optimizar el tiempo de funcionamiento.

Siguiendo esta filosofía, es necesario implementar una función periódica que compruebe que partes del hardware necesitan estar activas y a cuales pueden retirarse la alimentación. Esta función se ejecuta siempre dentro de la interrupción periódica Systick().

Para realizar la comprobación se ha creado una variable de tipo “bool” por cada una de las partes hardware, la cual indica si esa parte concreta del hardware ha terminado todas las tareas y puede ser desconectada. También se ha creado otra variable del mismo tipo que indica si además de no estar en uso, se ha realizado una petición de desconectar esa parte del hardware. Esta petición es necesaria ya que hay situaciones en las que si el hardware va a volver a ser utilizado, aunque en ese instante no esté en uso, será más eficiente que permanezca alimentado.

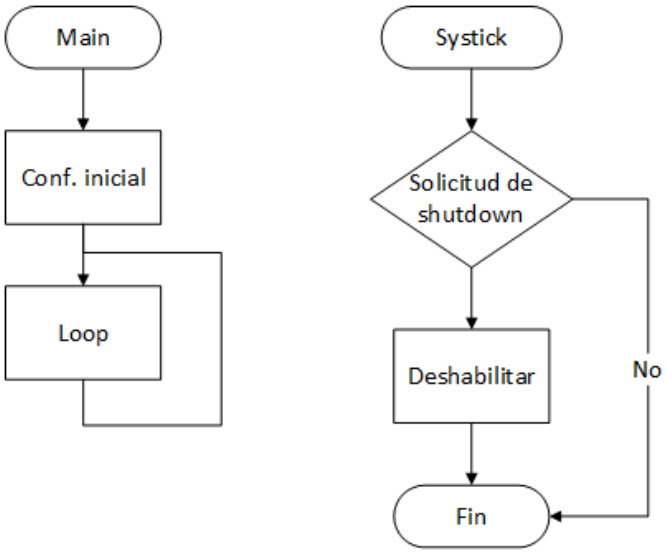


Figura 24.- Diagrama de flujo control de consumo

Capítulo 4.- Validación

Con el objeto de conocer algunos datos relevantes del prototipo implementado, se han llevado a cabo una serie de pruebas y validaciones. En el presente capítulo se pretende presentar los resultados más significativos de estas, acompañados de un breve análisis de los mismos. La descripción y resultados completos de estas pruebas pueden consultarse en el *ANEXO 3.- Experimentaciones.*

1. CONSUMO ENERGÉTICO

Para intentar estimar el consumo energético del prototipo, y del dispositivo final, se ha utilizado un multímetro de precisión configurado como amperímetro para medir la corriente de alimentación que se extraería de la batería en cada momento.

Como resultado de estas medidas se han obtenido los siguientes consumos estimados para el prototipo:

Tabla 12.- Resumen consumo energético del prototipo

Modo	Configuración	Consumo diario
Time-Lapse	Captura cada 1 minuto.	$4,201 Wh / día$
Detección	1 hora de detección al día.	$866,7 mWh / día$

En el dispositivo final, se podrán utilizar los modos de bajo consumo del microcontrolador. Con el microcontrolador configurado en modo BackUp y usando el dispositivo en el modo de detección, cuando no se detecte la presencia, se reducirá el consumo con respecto al prototipo. El resultado del cálculo estimativo de este consumo es el siguiente:

Tabla 13.- Consumo en modo BackUp

Modo	Configuración	Consumo diario
Detección (modo BackUp)	1 hora de detección al día.	$580,31 mWh / día$

Conociendo estos consumos, se puede determinar la vida útil del nodo sensor, teniendo en cuenta la energía almacenada en la batería, según la siguiente fórmula:

$$T(dias) = \frac{Energía(mWh)}{Consumo(mWh/día)}$$

En la siguiente tabla puede verse la vida útil estimada con dos baterías comerciales disponibles en el mercado:

Tabla 14.- Estimación vida útil con baterías comerciales

Batería	Energía (mWh)	Tiempo Time-Lapse (Días)	Tiempo Detección (Días)	Tiempo BackUp (Días)
Ansmann 2347-3003	9620	2,28	11,07	16,577
Ansmann 2447-3006	28860	6,86	33,29	49,73

2. COMUNICACIÓN INHALAMBRICA

Se ha creído conveniente realizar dos pruebas diferenciadas relacionadas con la comunicación a través de WiFi.

En la primera de ellas, el objetivo es comprobar la calidad de la señal WiFi a través de la antena incluida en la PCB. Para ello se han recogido datos RSSI de la señal del módulo RN171 conectado a un punto de acceso doméstico situado en una habitación de un hogar. Se han tomado datos a distintas distancias, y recorriendo varias estancias del hogar, como puede verse en la siguiente figura:

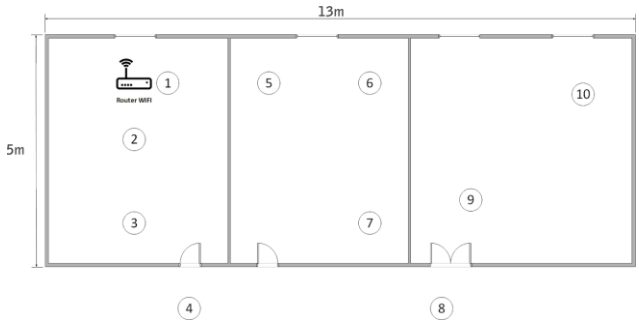


Figura 25.- Posiciones de medida de señal RSSI

El valor máximo registrado en todas las medidas realizadas en los distintos puntos numerados del 1 al 10, ha sido de -54 dbm. Pueden consultarse los resultados de todas las medidas en el *ANEXO 3.- Experimentaciones.*

Los resultados obtenidos confirman que es viable utilizar esta antena, al menos para distancias y atenuaciones similares a las dadas en el ensayo, ya que la señal RSSI no ha sobrepasado en ningún momento los -60 dBm que es el límite habitualmente utilizado para hablar de una señal estable[42].

En la segunda de las pruebas realizadas sobre comunicación inalámbrica, se ha pretendido comprobar cuál es la tasa de error que se produce al subir un archivo al servidor FTP, así como las conexiones fallidas que se producen. Esta prueba está motivada por los continuos fallos de conexión detectados en la etapa de desarrollo del firmware.

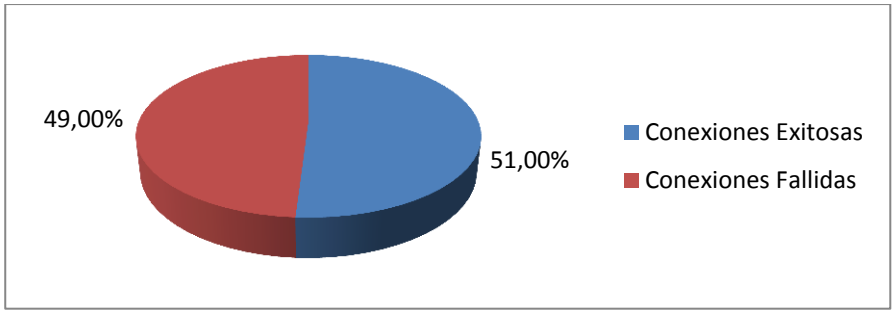


Figura 26.- Porcentaje de conexiones fallidas servidor FTP

Según las pruebas realizadas el 100% de los archivos se transfieren al servidor FTP, pero como se observa en la *Figura 26.- Porcentaje de conexiones fallidas servidor FTP* con una tasa de error en el conexionado de casi el 50%. Esto es posible debido a que el firmware del dispositivo realiza una serie de reintentos cuando detecta que la conexión no se ha realizado correctamente. La configuración de este ensayo puede consultarse en el *ANEXO 3.- Experimentaciones.*

3. AUDIO

Debido al ruido detectado en la señal de audio capturada por el conversor ADC, se plantean una serie de pruebas para intentar determinar la relación señal-ruido en la entrada del conversor y confirmar el correcto funcionamiento de la etapa pre amplificadora.

Durante el transcurso de estas pruebas se utiliza un osciloscopio con función FFT que permite ver la composición en el dominio de la frecuencia de la señal de salida de la etapa preamplificadora.

Cuando se mide en una estancia completamente en silencio se observa como aparece un ruido en la señal de unos -34,5 dB a 1,6 KHz.

A continuación se utiliza un altavoz por el que se emite un tono puro 1KHz. En la FFT de la señal se puede observar cómo se distingue perfectamente la señal de 1KHz, así como sus armónicos. Entonces aparece también la señal de ruido a 1,6KHz pero con un valor inferior de -58,1 dB. Cogiendo los valores de pico máximo la relación señal-ruido es de 32dB. En la siguiente figura se muestran la captura del osciloscopio durante la prueba:

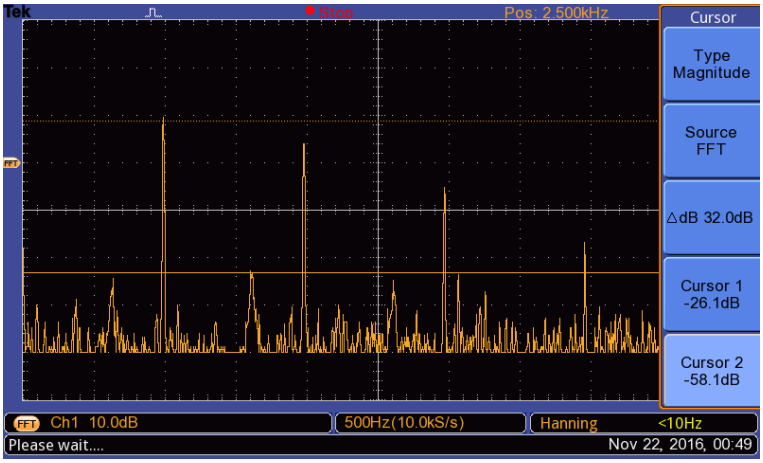


Figura 27.- FFT señal etapa pre amplificadora con señal acústica de 1KHz

Se repite esta misma prueba con señales acústicas de distintas frecuencias, comprobando que se repite la respuesta de la señal. Pueden consultarse las capturas de osciloscopio en el *ANEXO 3.- Experimentaciones*

El ruido obtenido en la etapa amplificadora es aceptable, pero podría mejorarse utilizando una alimentación más estable libre de ruido eléctrico.

Capítulo 5.- CONCLUSIONES

Como principal resultado de este T.F.M. se ha obtenido un prototipo plenamente funcional capaz de monitorizar la actividad de las personas en el hogar. Es dispositivo incorpora una comunicación inalámbrica a través de WiFi, una cámara con compresión JPEG, un micrófono omnidireccional con su etapa acondicionadora, un sensor de presencia PIR, un sensor de temperatura y humedad y una memoria SD. Todo lo anterior conectado a un microcontrolador de bajo consumo con una capacidad de procesamiento de 60 DMIPS. Además, cuenta con un cargador de baterías litio y litio-polímero, indicado para su uso en aplicaciones de “energy harvesting” con células solares.

Para lograr este objetivo se ha realizado un estado del arte acerca de la monitorización de personas en el hogar, principalmente mediante el empleo de redes de sensores inalámbricas de imagen. En base a este estado del arte, y a las necesidades planteadas en el proyecto *MEMORY LANE,* se ha decidido realizar el diseño, desarrollo e implementación de un prototipo que cumpla los objetivos planteados al inicio de este trabajo.

Una vez planteado el prototipo, se ha realizado la búsqueda tecnológica de los componentes electrónicos y el diseño para su integración en una placa de circuito impreso. Diseñada la placa de circuito impreso, se ha fabricado y montado los componentes seleccionados para tener una plataforma hardware sobre la que desarrollar el firmware. Antes de comenzar a programar se ha realizado la puesta en funcionamiento del hardware, comprobando que todas las partes a priori funcionan correctamente.

Con el hardware funcionando, se ha comenzado a programar el firmware, empezando por desarrollar las librerías de más bajo nivel que controlan todos los periféricos unidos al microcontrolador en el diseño. Una vez probadas y validadas estas librerías, se ha continuado por programar la siguiente capa de abstracción, denominada aplicaciones secundarias. Esta capa lleva a cabo todas las funcionalidades posibles del dispositivo ejecutando una función distinta para cada una de ellas. Para acabar con el desarrollo del firmware, se ha programado la aplicación principal, la cual utiliza las funciones de las aplicaciones secundarias para establecer el funcionamiento final del prototipo.

Como resultado de este desarrollo tanto hardware como firmware, se ha conseguido capturar los siguientes datos:

- Imágenes en formato JPEG de 640x480 píxeles
- Clips de audio con una calidad de 100kbps y 8 bits de resolución de 2 segundos de duración

- Medida de temperatura con una precisión de ±0.3ºC desde +5 a +60ººC
- Medida de humedad con una precisión de ±2% desde 20% a 80% RH
- Detección de personas con un alcance de 5 metros

Todos estos datos han podido ser almacenados en una memoria no volátil extraíble, y enviados a la nube con un tiempo máximo de 40 segundos por ciclo de captura.

En cuanto a consumo energético, se ha conseguido utilizar métodos que permiten la conexión y desconexión de las distintas partes del dispositivo, consiguiendo así reducir drásticamente el consumo energético en los tiempos muertos del nodo sensor.

Consecuentemente, se puede afirmar que se ha conseguido cumplir satisfactoriamente todos los objetivos marcados al inicio del trabajo, comprobando que es viable la utilización de este dispositivo para la aplicación final propuesta.

Como líneas de trabajo futuras se plantea mejorar la eficiencia del dispositivo utilizando los modos de bajo consumo del microcontrolador. También utilizando fuentes de alimentación más eficientes, y mejorando el algoritmo de detección.

Adicionalmente, podría aumentarse las prestaciones del dispositivo añadiendo una memoria, volátil o no, de acceso rápido que permita almacenar clips de audio de mayor duración, o intentar depurar el firmware del dispositivo para conseguir escribir en tiempo real en la memoria SD mientras se captura el sonido. En lo referente a la captura de audio, podría mejorarse la relación señal ruido del mismo utilizando una fuente de alimentación más estable para la parte analógica, separándola del ruido eléctrico de la parte digital.

Otra línea de trabajo futura es tratar de integrar mayor inteligencia en el sensor, para que este sea capaz de detectar ciertos hábitos, reduciendo así el consumo y minimizando las trasmisiones.

Bibliografía

[1] Hampapur, A., Brown, L. , Connell, J. , Ekin, A. , Lu, M. and Pankanti, S. (2005). Smart video surveillance: exploring the concept of multiscale spatiotemporal tracking. *IEEE Signal Processing Magazine*, 22 (2) pp 38-51.

[2] Paek J, Hicks J, Coe S and Govindan R. (2014). Image-based environmental monitoring sensor application using an embedded wireless sensor network. *Open Access Sensors.* 14 (9)

[3] Yunxia Chen and Qing Zhao.(2005). On the lifetime of wireless sensor networks. *IEEE Communications Letters*. 9 (11) pp 976 - 978

[4] Gomez Cid-Fuentes R., Cabellos-Aparicio A., and Alarcón. E. (2014). Energy Buffer Dimensioning Through Energy-Erlangs in Spatio-Temporal-Correlated Energy-Harvesting-Enabled Wireless Sensor Networks. *IEEE Journal on Emerging and Selected Topics in Circuits and systems*. 4 (3) pp 301 – 312.

[5] Imran, M., Shahzad, K., Ahmad, N., O'Nils, M., Lawal, N., and Oelmann, B. (2014). Energy-Efficient SRAM FPGA-Based Wireless Vision Sensor Node: SENTIOF-CAM. *IEEE Transactions on Circuits and Systems for Video Technology*. 24 (12) pp 2132 – 2143.

[6] Gasparini, L., Manduchi, R., Gottardi, M., and Petri, D.(2011). An Ultralow-Power Wireless Camera Node: Development and Performance Analysis. *IEEE Instrumentation and Measurement Society*. 60 (12) pp 3824 - 3832

[7] Hengstler, S., Prashanth, D., Sufen Fong, and Aghajan, H. (2007). MeshEye: A Hybrid-Resolution Smart Camera Mote for Applications in Distributed Intelligent Surveillance. *6th International Symposium on Information Processing in Sensor Networks.* pp360 - 369

[8] Rowe, A. G., Goode, A., Goel, D., and Nourbakhsh, I. (2007). CMUcam3: An Open Programmable Embedded Vision Sensor. [pdf] Pittsburgh, Pennsylvania: Robotics Institute Carnegie Mellon University. Available at: http://repository.cmu.edu/cgi/viewcontent.cgi?article=1849&context=robotics [Accessed 23 Nov 2016]

[9] Hengstler, S., and Aghajan, H. (2006), Application Development in Vision-Enabled Wireless Sensor Networks. *International Conference on Systems and Networks Communications,* pp 30 - 30

[10] Rahim, M. , Baer , R. , Iroezi , O. I., Garcia, J. C., Warrior, J., Estrin, D., and Srivastava, M. (2005) Cyclops: In situ image sensing and interpretation in wireless sensor networks. In: *3rd international conference on Embedded networked sensor systems.* San Diego: ACM pp192-204

[11] Bakkali, M., Carmona-Galan, R. and Rodriguez-Vazquez, A. (2015). A prototype node for wireless vision sensor network applications development. *International Symposium on Communications and Mobile Network.* pp1 - 4

[12] Ferrigno L., Marano S., Paciello V., and Pietrosanto A. (2005). Balancing computational and transmission power consumption in wireless image sensor networks. *IEEE International Conference on Virtual Environments, Human-Computer Interfaces and Measurement Systems.* pp6

[13] Rowe, A. and Rajkumar, R. (2007). FireFly Mosaic: A Vision-Enabled Wireless Sensor Networking System. *28th IEEE International Real-Time Systems Symposium.* pp 459 - 468

[14] Teixeira, T., Culurciello, E., Park, J., Lymberopoulos, D., Barton-Sweeney, A. and Savvides, A. (2006). Address-Event Imagers for Sensor Networks: Evaluation and Modeling. *5th International Conference on Information Processing in Sensor Networks.*

[15] LinkSprite, (n.d.). *LinkSprite Official Website*.[online] Available at: http://www.linksprite.com/ [2016]

[16] ST Microelectronics , "Tutorial for MEMS microphones", *Application Note AN4426*

[17] MP33AB01H,(n.d.).*ST Official Website*. [online] Available at: http://www.st.com/en/audio-ics/mp33ab01h.html [Accessed 23 Nov 2016]

[18] Teorema de Nyquist,(2016). *Wikipedia.* [online] Available at:

https://es.wikipedia.org/wiki/Teorema_de_muestreo_de_Nyquist-Shannon [Accessed 23 Nov 2016]

[19] SSM2167 Datasheet, (2011). *Analog Devices Official Website*. [online] Available at: http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2167.pdf [Accessed 23 Nov. 2016]

[20] Lewis,J., (2012-2013). *Op Amps for MEMS Microphone Preamp Circuits, AN-1165 APPLICATION NOTE Analog Devices*. Rev. A. [pdf] Norwood: Analog Devices, 8. Available at: http://www.analog.com/media/en/technical-documentation/application-notes/AN-1165.pdf [Accessed 23 Nov 2016]

[21] EKMC1601111, (n.d.). *Panasonic Electronic Components Official Website*. [online] Available at: https://na.industrial.panasonic.com/products/sensors/sensors-automotive-industrial-applications/pir-motion-sensor-papirs/series/ekmc-vz-series/2481/model/EKMC1601111 [Accessed 23 Nov. 2016]

[22] SHT31-DIS Datasheet, (2016). *Sensirion Official Website*. [online] Available at: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/Humidity_Sensors/Sensirion_Humidity_Sensors_SHT3x_Datasheet_Protective_Cover.pdf [Accessed 23 Nov. 2016]

[23] C. A. Trasviña-Moreno, A. Asensio, R. Casas, R. Blasco, A. Marco. WiFi Sensor Networks: A study of energy consumption, *International Multi-Conference on Systems, Signals and Devices, 2014.*

[24] NXP Semiconductors, (2013). *AN10911 SD(HC)-memory card and MMC interface conditioning*. Rev. 2. [pdf] NXP Semiconductors, 27. Available at: http://www.nxp.com/documents/application_note/AN10911.pdf [Accessed 23 Nov. 2016]

[25] F. Foust: "Secure Digital Card Interface for the MSP430", *Dept. of Electrical and Computer Engineering Michigan State University,* 2004

[26] SD Group (Panasonic, SanDisk, Toshiba), Technical Committee and SD Card Association, (2013). *SD Specifications Part 1. Physical Layer Simplified Specification*. Version 4.10. [pdf] n.d.: SD Group and SD Card Association, 202. Available at: https://members.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf [Accessed 23 Nov. 2016]

[27] REG 103 Datasheet, (2005). *Texas Instruments Official Website.* [online] Available at: http://www.ti.com.cn/cn/lit/ds/symlink/reg103.pdf

[28] REG 102 Datasheet, (2005). *Texas Instruments Official Website.* [online] Available at: http://www.ti.com.cn/cn/lit/ds/symlink/reg102.pdf

[29] J. Celani : Solar Battery Charger Maintains High Efficiency in Low Light, *LT Journal of Analog Innovation,* Oct 2013

[30] JPEG Camera, (n.d.). *LinkSprite Wiki*. [online] Available at: http://linksprite.com/wiki/index.php5?title=JPEG_Color_Camera_Serial_UART_Interface_(TTL_level)

[31] ATSAM4LC4C, (n.d.). *Atmel Official Website.* [online] Available at: http://www.atmel.com/devices/ATSAM4LC4C.aspx?tab=overview [Accessed 23 Nov. 2016]

[32] RN171, (1998-2016). *Microchip Official Website*. [online] Available at: http://www.microchip.com/wwwproducts/en/RN171 [Accessed 23 Nov. 2016]

[33] Batería de ion de litio, (13 nov 2016). *Wikipedia.* [online] Available at: https://es.wikipedia.org/wiki/Bater%C3%ADa_de_ion_de_litio [Accessed 23 Nov. 2016]

[34] Lithium polymer battery, (November 2016). *Wikipedia.* [online] Available at: https://en.wikipedia.org/wiki/Lithium_polymer_battery [Accessed 23 Nov. 2016]

[35] LTC4121/LTC4121-4.2 - 40V 400mA Synchronous Step-Down Battery Charger, (n.d.). *LT Official Website.* [online] Available at: http://www.linear.com/product/LTC4121 [Accessed 23 Nov. 2016]

[36] Células solares de interior, (2007). *IXYS Official Website*. [online] Available at: http://ixapps.ixys.com/ [Accessed 23 Nov. 2016]

[37] Conector miniaturizado debug, (2016). *Tag-Connect Official Website.* [online] Available at: http://www.tag-connect.com/[Accessed 23 Nov. 2016]

[38] Espectro audible, (27 ago 2016). *Wikipedia.* [online] Available at: https://es.wikipedia.org/wiki/Espectro_audible[Accessed 23 Nov. 2016]

[39] 9V 150MA SOLAR CELL, (2016). *Sundance Solar.* [online] Available at: http://store.sundancesolar.com/9v-150ma-solar-cell/

[40] *http://literature.cdn.keysight.com/litweb/pdf/34460-90901.pdf?id=2345839*

[41] Sleep manager, (2016). *Atmel Sofwate Framework Website.* [online] Available at: http://asf.atmel.com/docs/latest/common.services.basic.sleepmgr.example.sam4l_xplained_pro/html/group__sleepmgr__group.html [Accessed 23 Nov. 2016]

[42] Indicador de fuerza de la señal recibida, (2016). *Wikipedia.* [online] Available at: https://es.wikipedia.org/wiki/Indicador_de_fuerza_de_la_se%C3%B1al_recibida [Accessed 23 Nov. 2016]

[43] XAMPP, (2016). *Apache Friends Official Website.* [online] Available at: https://www.apachefriends.org/es/index.html [Accessed 23 Nov. 2016]

[44] FileZilla, (n.d.). *FileZilla Official Website.* [online] Available at: https://filezilla-project.org/ [Accessed 23 Nov. 2016]

[45] Atmel Software Framework, (2016). *Atmel Software Framework Official Website.* [online] Available at: http://asf.atmel.com/docs/latest/ [Accessed 23 Nov. 2016]

Glosario de términos

ASF Atmel Software Framework

ASIC Es un circuito integrado hecho a la medida para un uso en particular, en vez de ser concebido para propósitos de uso general.

CPLD Del acrónimo inglés *Complex Programmable Logic Device*

DSP Sigla en inglés de *Digital Signal Processor,* es un sistema basado en un procesador o microprocesador que posee un conjunto de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad.

EFICIENCIA ENERGÉTICA La eficiencia energética es el uso eficiente de la energía, de esta manera optimizar los procesos y el empleo de la energía utilizando lo mismo o menos para producir más.

ENERGY HARVESTING Es el proceso a través del cual se extrae energía de fuentes externas como puede ser la energía solar, se almacena, y es usada para alimentar dispositivos electrónicos de bajo consumo

FPGA Del inglés *Field Programmable Gate Array*, es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada 'in situ' mediante un lenguaje de descripción especializado.

FRAMES Término inglés que se traduce por fotograma, es decir, cada una de las imágenes instantáneas en las que se divide una película de cine que dan sensación de movimiento al ser proyectadas secuencialmente

FTP Siglas en inglés de *File Transfer Protocol*, Protocolo de Transferencia de Archivos, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en la arquitectura cliente-servidor.

FIRMWARE Es un programa que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

HAL	Del inglés *Hardware Abstraction Layer*, es la capa de firmware que abstrae al programador de los registros del microcontrolador mediante el uso de funciones.
LIFE-BLOG	Recopilación de imágenes y sonidos que percibe una persona a lo largo del día, que son almacenados cronológicamente para su posterior consulta y visualización.
MEMS	Se refieren a la tecnología electromecánica, micrométrica y sus productos, y a escalas relativamente más pequeñas (escala nanométrica).
MIPS	Millones de instrucciones por segundo o MIPS (del inglés M*illions of Instructions Per Second*) es una forma de medir la potencia de los microprocesadores.
PCB	Del inglés *Printed Circuit Board,* es la superficie constituida por caminos, pistas o buses de material conductor laminadas sobre una base no conductora.
PIR	Sensor infrarrojo pasivo. Es un sensor electrónico que mide la luz infrarroja (IR) radiada de los objetos situados en su campo de visión. Se utilizan principalmente en los detectores de movimiento
RSSI	Indicador de fuerza de la señal recibida (RSSI por las siglas del inglés Received Signal Strength Indicator)
SNTP	Servidor de protocolo NTP. Protocolo de Internet para sincronizar los relojes de los sistemas informáticos a través del enrutamiento de paquetes en redes con latencia variable
SPI	Del inglés *Serial Peripheral Interface* es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
SYSTICK	Temporizador de sistema propio de arquitectura ARM. Genera una interrupción cíclica de frecuencia configurable.
TFM	Trabajo Final de Master

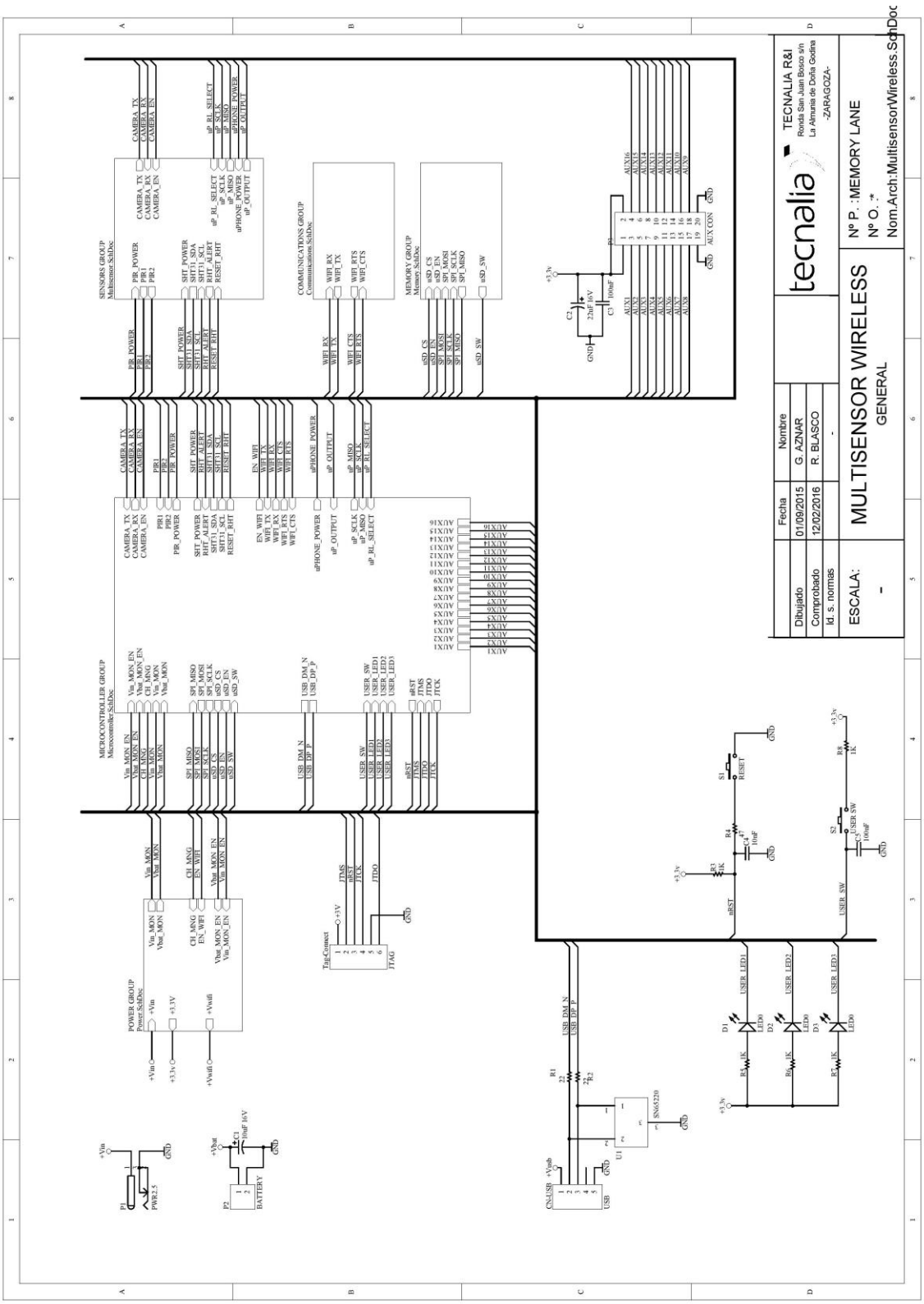
TIMESTAMP Conocida también como registro de tiempo, sello de tiempo o timestamp, es una secuencia de caracteres que denotan la hora y fecha (o alguna de ellas) en la/s que ocurrió determinado evento.

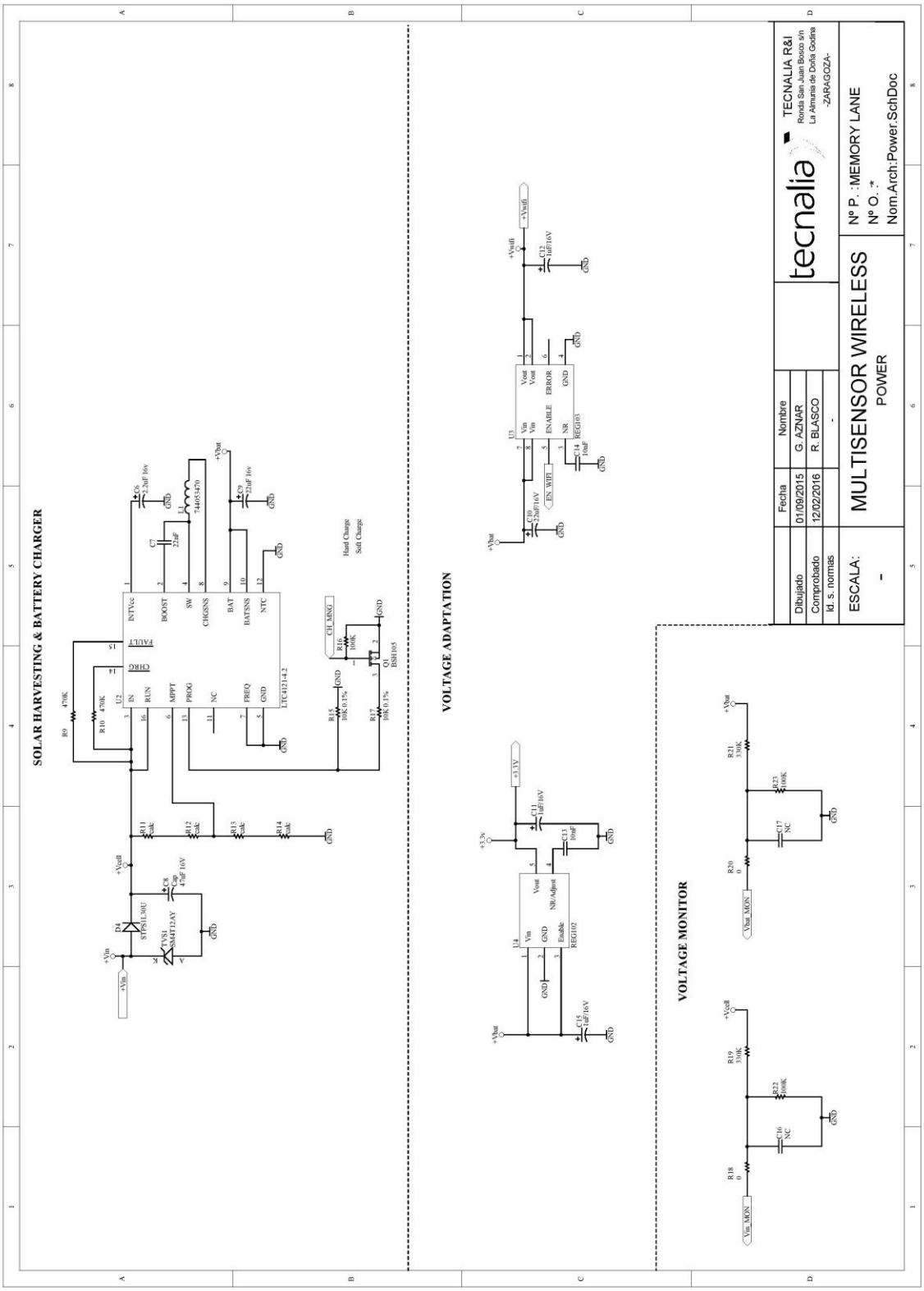
TVS Del inglés transient-voltage-suppression, es un dispositivo de protección para circuitos electrónicos que neutraliza los picos de tensión de corta duración que pueden producirse en distintas partes del circuito.

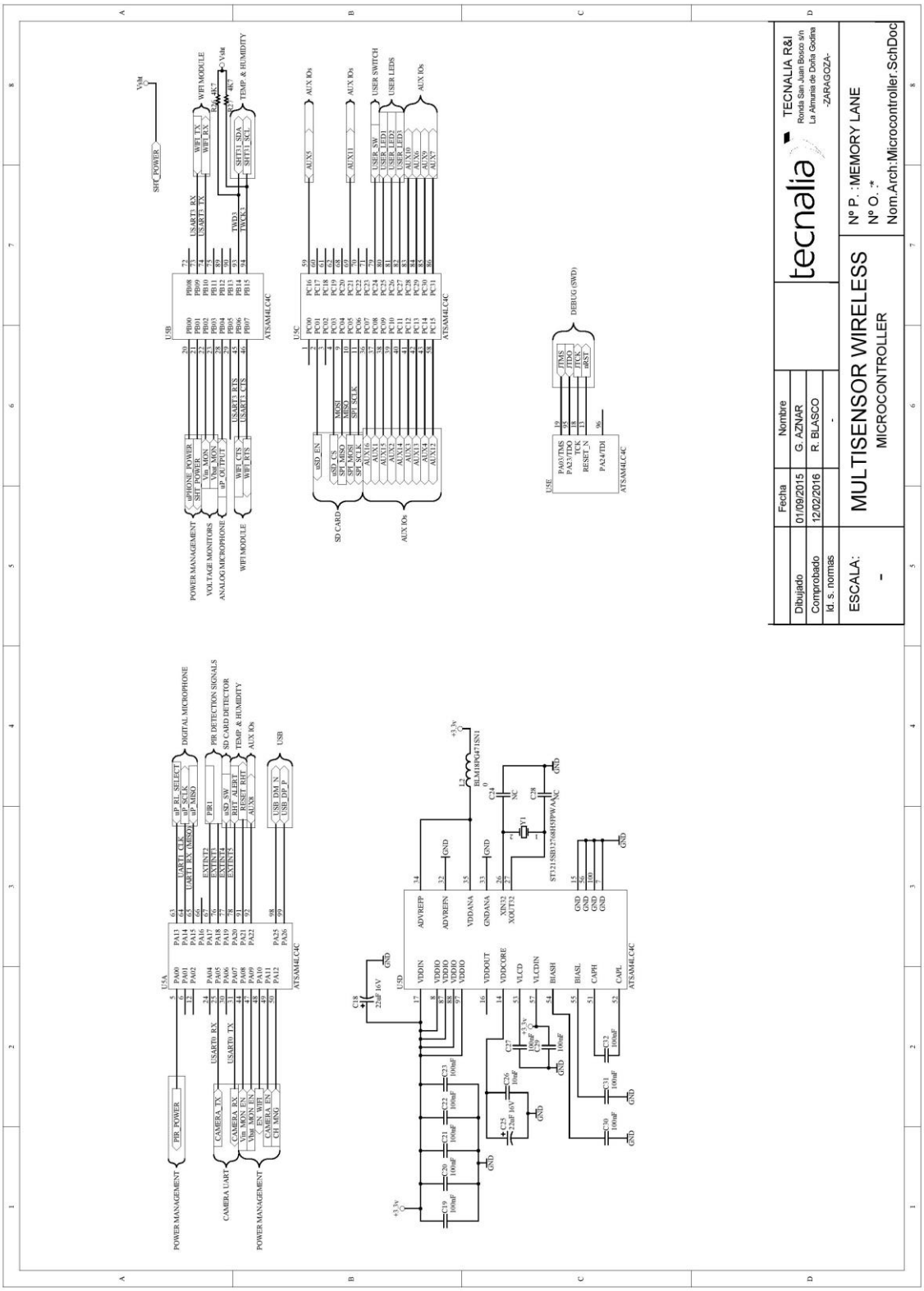
UART Siglas en inglés de *Universal Asynchronous Receiver-Transmitter*, es el dispositivo que controla los puertos y dispositivos serie.

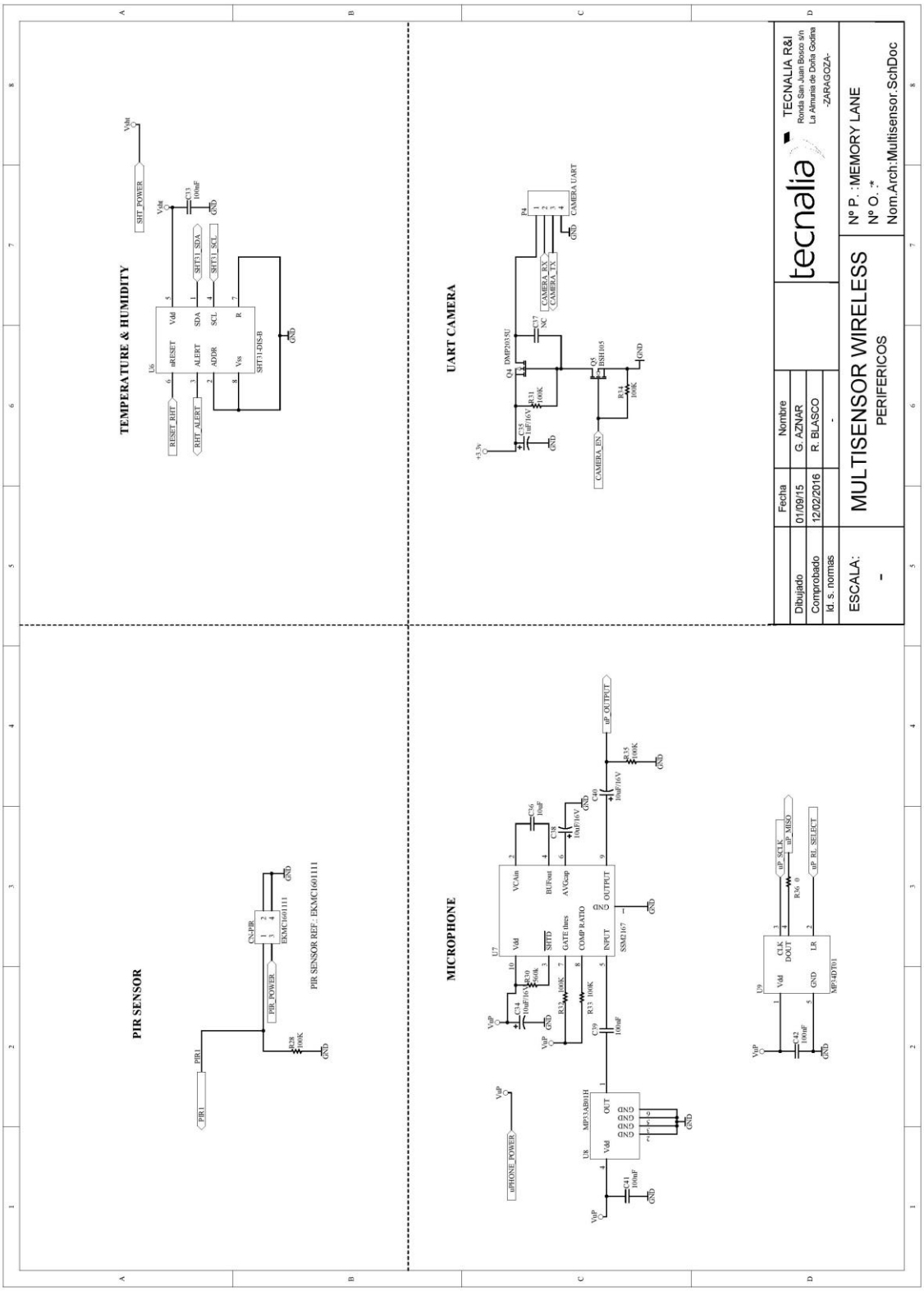
WLAN Del inglés *Wireless Local Area Network*, red de área local inalámbrica

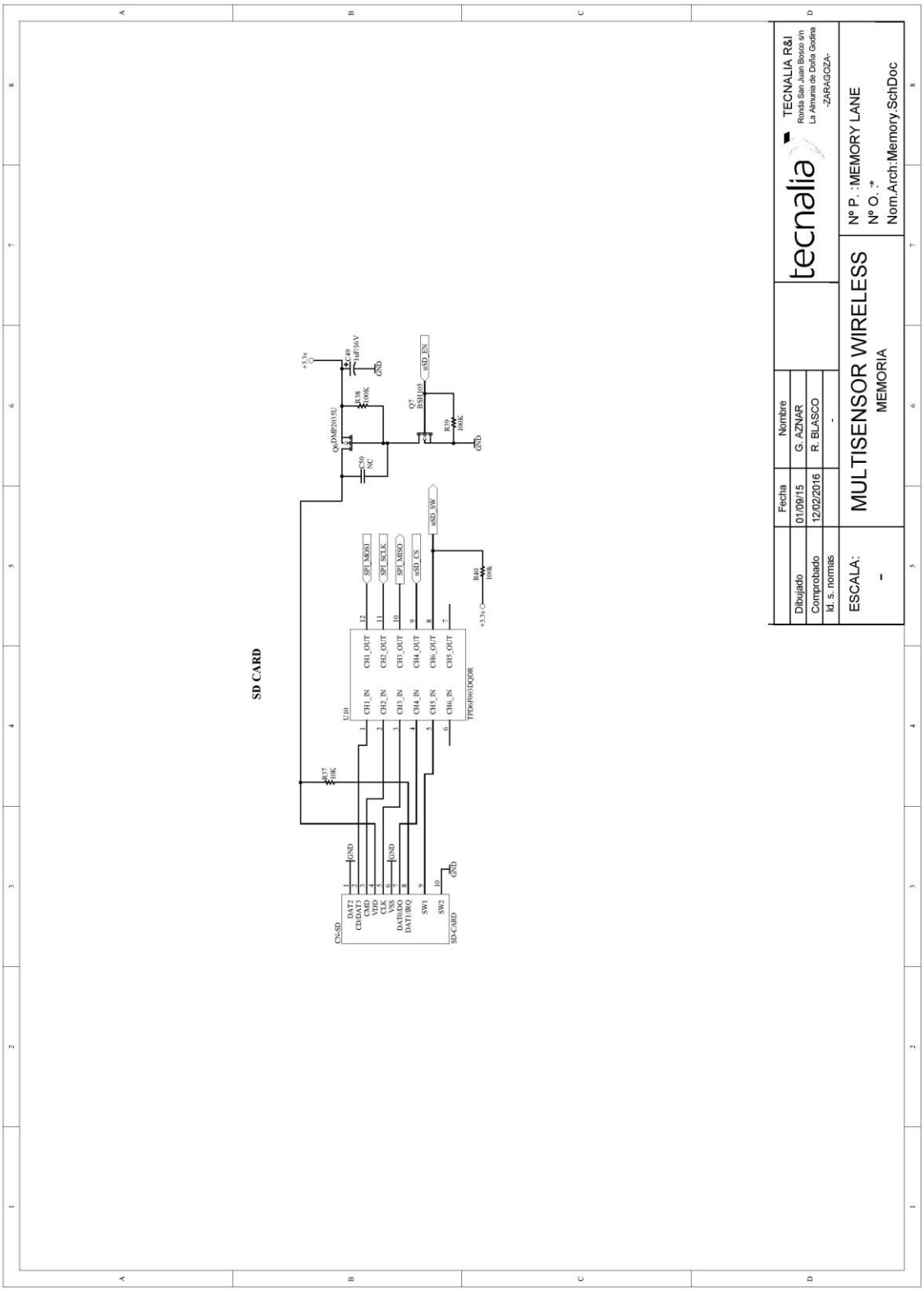
ANEXO 1: Esquemas PCB

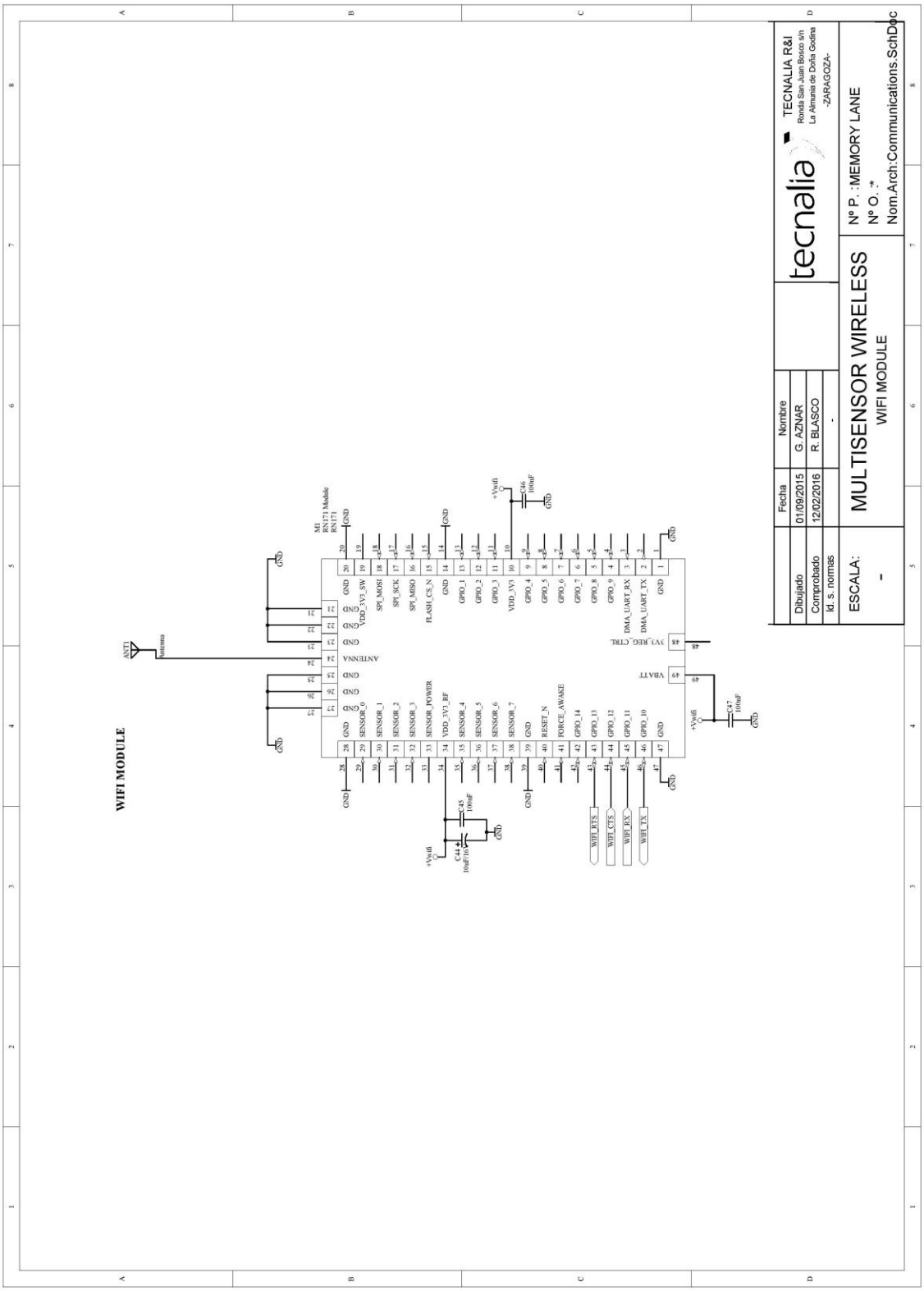

tecnalia
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
Dibujado 01/09/2015 G. AZNAR
Comprobado 12/02/2016 R. BLASCO
Id. s. normas -
ESCALA: -
MULTISENSOR WIRELESS
GENERAL
Nº P. : MEMORY LANE
Nº O. :*
Nom:Arch:MultisensorWireless.SchDoc
SENSORS GROUP
Multisensor.SchDoc
COMMUNICATIONS GROUP
Communications.SchDoc
MEMORY GROUP
Memory.SchDoc
MICROCONTROLLER GROUP
Microcontroller.SchDoc
POWER GROUP
Power.SchDoc
TagConnect
JTAG
BATTERY
USB
RESET
USER_SW
LED0
LED1
LED2
LED3
AUX CON


SOLAR HARVESTING & BATTERY CHARGER
VOLTAGE ADAPTATION
VOLTAGE MONITOR
LTC4121-4.2
REG103
REG102
Hard Charge
Soft Charge
tecnalia
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
Dibujado
Comprobado
Id. s. normas
Fecha
01/09/2015
12/02/2016
Nombre
G. AZNAR
R. BLASCO
ESCALA:
-
MULTISENSOR WIRELESS
POWER
Nº P. :MEMORY LANE
Nº O. :*
Nom.Arch:Power.SchDoc


tecnalia
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
Dibujado 01/09/2015 G. AZNAR
Comprobado 12/02/2016 R. BLASCO
Id. s. normas -
ESCALA: -
MULTISENSOR WIRELESS
MICROCONTROLLER
Nº P. : MEMORY LANE
Nº O. :*
Nom:Arch:Microcontroller.SchDoc
POWER MANAGEMENT
VOLTAGE MONITORS
ANALOG MICROPHONE
WIFI MODULE
SD CARD
AUX IOs
DEBUG (SWD)
DIGITAL MICROPHONE
PIR DETECTION SIGNALS
SD CARD DETECTOR
TEMP. & HUMIDITY
CAMERA UART
ATSAM4LC4C


TEMPERATURE & HUMIDITY
UART CAMERA
PIR SENSOR
MICROPHONE
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
tecnalia
Dibujado 01/09/15 G. AZNAR
Comprobado 12/02/2016 R. BLASCO
Id. s. normas -
ESCALA: -
MULTISENSOR WIRELESS
PERIFERICOS
Nº P. :MEMORY LANE
Nº O. :*
Nom.Arch:Multisensor.SchDoc


SD CARD
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
tecnalia
Fecha
Nombre
Dibujado
01/09/15
G. AZNAR
Comprobado
12/02/2016
R. BLASCO
Id. s. normas
-
ESCALA:
-
MULTISENSOR WIRELESS
MEMORIA
Nº P. : MEMORY LANE
Nº O. :*
Nom:Arch:Memory.SchDoc


WIFI MODULE
M1
RN171 Module
RN171
ANT1
Antenna
ANTENNA
SPI_MOSI
SPI_SCK
SPI_MISO
FLASH_CS_N
GPIO_1
GPIO_2
GPIO_3
VDD_3V1
GPIO_4
GPIO_5
GPIO_6
GPIO_7
GPIO_8
GPIO_9
DMA_UART_RX
DMA_UART_TX
3V3_REG_CTRL
VBATT
SENSOR_1
SENSOR_2
SENSOR_3
SENSOR_POWER
VDD_3V3_RF
SENSOR_4
SENSOR_5
SENSOR_6
SENSOR_7
RESET_N
FORCE_AWAKE
GPIO_14
GPIO_13
GPIO_12
GPIO_11
GPIO_10
WIFI_RTS
WIFI_CTS
WIFI_RX
WIFI_TX
+Vwifi
C46 100nF
C47 100nF
C45 100nF
C44 10uF/16V
TECNALIA R&I
Ronda San Juan Bosco s/n
La Almunia de Doña Godina
-ZARAGOZA-
tecnalia
Nombre
Fecha
Dibujado
01/09/2015
G. AZNAR
Comprobado
12/02/2016
R. BLASCO
Id. s. normas
-
ESCALA:
-
MULTISENSOR WIRELESS
WIFI MODULE
Nº P. :MEMORY LANE
Nº O. :*
Nom.Arch:Communications.SchDoc

1. MODIFICACIONES HARDWARE NECESARIAS

- Huella SHT31
- Pad alimentación Tag-connect (SWD)
- Footprint TVS uSD.
- Pinout uSD.
- Huella BSH105 (SOT23-3) (correspondencia pines)
- Arranque del regulador lineal interno. (PA02 pulldown + VDDOUT a VDDCORE)
- Arranque en modo NO LCD. (VLCD a 3.3v, y no es necesario montar C30 C31 y C32)

ANEXO 2: Resumen de las funciones implementadas en cada librería.

1. Contenido del anexo

Este anexo pretende ser un resumen donde se reúnan todas las funciones disponibles en cada una de las librerías implementadas junto con una breve explicación de su funcionamiento.

2. Funciones aplicación principal

Las funciones de la aplicación principal permiten, con la ejecución de una sola función, el realizar las acciones necesarias para conseguir la funcionalidad del dispositivo sin necesidad de conocer las acciones que llevan a cabo estas funciones internamente. Estas funciones se encuentran descritas y definidas en los archivos *main.c* y *main.h.*

- `void Set_FileName (void)`: Configura el nombre con el que van a ser guardados los archivos en la siguiente captura de sensores.
- `void UploadStoringData (void)`: Copia los archivos que solamente están guardados en local en la SD al servidor FTP.
- `void CaptureSensors (void)`: Función privada de la librería main.c, realiza una captura de los datos de todos los sensores y los almacena en la memoria SD.
- `void Wait (volatile uint32_t ul_ms)`: Función exportable de la librería main.c, realiza una espera del tiempo marcado en la variable *ul_ms*. Para utilizarla en otra librería incluir el archivo main.h.

3. Funciones aplicaciones secundarias

3.1. Aplicación secundaria Módulo WIFI

Esta aplicación secundaria está contenida en los archivos *ML_WiFi.c* y *ML_WiFi.h* y pretende dotar de las funciones necesarias para realizar acciones necesarias relacionadas con la conectividad a la red WIFI y el envío/recepción de datos desde internet.

Para ello se han desarrollado una serie de funciones exportables a otras librerías incluyendo el archivo *ML_WiFi.h* en la cabecera de la librería. Estas funciones son las siguientes:

- `void WiFi_StartUp_config (void)`: Función exportable incluyendo la librería *ML_WiFi.h* que realiza las siguientes acciones:

  - Configura el hardware necesario para la comunicación con el módulo WIFI.
  - Realiza la configuración de parámetros del módulo WIFI (WLAN, FTP, SNTP).
  - Comprueba la conexión a la red WLAN y sincroniza el tiempo con el servidor SNTP establecido.
  - Obtiene la dirección MAC y la almacena en el buffer *MAC_ADD[12]* de la librería *RN171.c.*

- `void JoinWlan (void)`: abre una conexión con la red WLAN establecida.
- `void RefreshTimeStamp (void)`: Obtiene el dato "time stamping" actualizado desde el servidor SNTP configurado, y lo guarda en el buffer *RTC_VALUE[10]* de la librería *RN171.c*.
- `void SDfileToFTP (char *file_to_transf, FILE_TYPE file_type_selected)`: Comprueba si el fichero indicado existe en la memoria SD, y si existe lo copia al servidor FTP configurado.

3.2. Aplicación secundaria cámara JPEG

- `void PictureShooting (void)`: Dispara una foto en el momento que se ejecuta la función y la guarda en la memoria RAM de la cámara hasta que esta sea leída o se retire la alimentación.
- `void PictureClear (void)`: Limpia la memoria RAM de la cámara, eliminando la foto que esté retenida en dicha memoria.
- `void PictureToSD (char *file_name)`: Guarda en la memoria SD la foto contenida en la memoria RAM de la cámara con el nombre indicado en el puntero **file_name*. Si no hay foto en la RAM de la cámara dispara una en el momento que ejecuta esta función.

3.3. Aplicación secundaria micrófono

- `void RecordClip (void)`: Graba un clip de audio y lo almacena en el buffer de la memoria RAM del microcontrolador.
- `void ClearClip (void)`: Elimina el clip de audio guardado en la memoria RAM del microcontrolador.
- `void AudioClipToSD (char *file_name)`: Guarda en la memoria SD el clip de audio contenido en el buffer de la memoria RAM del microcontrolador con el nombre indicado en

el puntero *file_name. Si no hay un clip de audio almacenado en la RAM graba uno en el momento en el que se ejecuta la función.

3.4. Aplicación secundaria PIR

- `void PIR_Detection (bool STATE)`: Activa o desactiva la detección del PIR en función del valor de la variable *STATE* de tipo bool que se introduzca.

3.5. Aplicación secundaria sensor temperatura y humedad

- `void Get_Temp_Hum (SHT31_Measurement *Measurement)`: Realiza una captura de los datos de temperatura y humedad relativa, y los guarda en la estructura de datos a la que apunta el puntero **Measurement*. Una vez completada la captura, desactiva la alimentación del sensor.

3.6. Aplicación secundaria de gestión de energía

- `void PowerMgt_Init (void)`: Inicia y configura el hardware necesario para realizar las funciones descritas en la librería.
- `void Get_Vin_Value (void)`: actualiza el dato del voltaje de entrada.
- `void Get_Vbat_Value (void)`: actualiza el dato del voltaje de batería.
- `void Check_Shutdown_Rqst (void)`: Comprueba si hay alguna solicitud de apagado de hardware que se puede llevar a cabo.
- `void Camera_Sutdown_Rqst (void)`: Solicita un apagado del hardware de la cámara en cuanto sea posible.
- `void WiFi_Sutdown_Rqst (void)`: Solicita un apagado del hardware del WiFi en cuanto sea posible.
- `void uPhone_Sutdown_Rqst (void)`: Solicita un apagado del hardware del micrófono en cuanto sea posible.
- `void SHT_Sutdown_Rqst (void)`: Solicita un apagado del hardware del sensor de temperatura y humedad en cuanto sea posible.

ANEXO 3.- Experimentaciones

1. MEDIDAS DE CONSUMO ENERGÉTICO

Para comprobar el consumo real de las distintas partes del circuito electrónico, se han realizado una serie de medidas en la entrada de alimentación del dispositivo. Para ello se ha utilizado un multímetro de precisión KEYSIGHT 34461A, configurado como amperímetro y colocado entre la salida de una fuente de alimentación METRIX AX 503 y la entrada de alimentación para batería del dispositivo. Para más detalle sobre la configuración del multímetro consultar el manual de usuario [40].

Figura 28 Detalle multimetro KEYSIGHT 34461A. Fuente: KEYSIGHT TECHNOLOGY

Figura 29.- Detalle fuente de alimentación METRIX AX503. Fuente: METRIX

Con esta configuración, además de medir el consumo propio de cada parte de la electrónica, se incluye también la potencia disipada en las distintas adaptaciones de voltaje. Aunque se comete un cierto error, ya que la eficiencia de estas etapas de adaptación depende de la potencia total de salida, la medida puede servir para hacer una estimación bastante acertada del consumo que se va a tener en cada momento.

El voltaje de alimentación utilizado para todas las medidas ha sido de 3,6 v.

1.1. CONSUMO SAM4LC4

El consumo del SAM4LC4 varía dependiendo de la configuración de reloj del sistema, de las partes activas que se tengan en cada momento, y del modo de bajo consumo configurado.

En el firmware del prototipo se ha utilizado el oscilador interno configurado a una frecuencia de 8MHz. Para este firmware no se han utilizado los modos de bajo consumo del microcontrolador, por lo que este permanece siempre en el modo activo.

Se deciden realizar dos medidas de consumo del microcontrolador, para determinar el consumo propio de la CPU. En una se utiliza el modo de funcionamiento activo y en la otra se utiliza el modo de más bajo consumo "*BackUp Mode*"[41]. Para realizar ambas medidas, se inicializa el microcontrolador con la configuración propia del firmware del prototipo. A continuación se realizan las siguientes acciones dependiendo del ensayo:

- En el caso del modo de bajo consumo, justo a continuación de la configuración se pasa ejecuta la función que hace entrar al microcontrolador en este modo, y una vez estabilizada la corriente se comienza a medir.
- Para la medida en el modo activo, se realiza la configuración, y a continuación se ejecuta un bucle infinito. Una vez en el bucle infinito se comienza a medir.

Los resultados obtenidos son los siguientes:

Tabla 15.- Consumo del microcontrolador en los diferentes modos.

Modo del microcontrolador	Intensidad mínima	Intensidad media	Intensidad máxima
Activo	4,153 mA	**4,158 mA**	4,161 mA
BackUp	883,8 µA	**884,25 µA**	884,7 µA

$$P_{MODO\_ACTIVO} = 4,158mA \cdot 3,6v = 14,96mW$$

$$P_{MODO\_BACKUP} = 884,25\mu A \cdot 3,6v = 3,1833mW$$

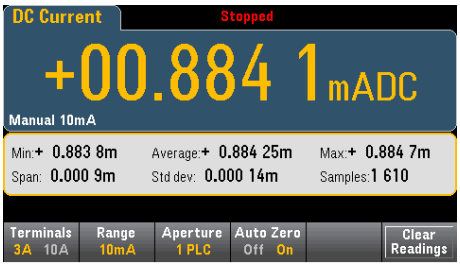


Figura 30.- Captura de pantalla intensidad SAM4LC4 modo bajo consumo

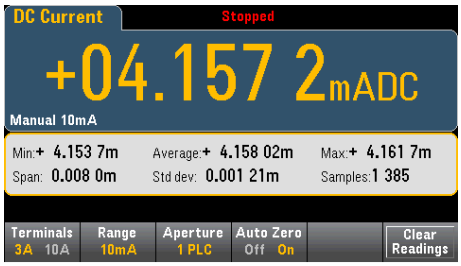


Figura 31.- Captura de pantalla intensidad SAM4LC4 modo activo

1.1. CONSUMO MEMORIA SD

Con el fin de comprobar el consumo de la memoria SD, se monitoriza la corriente consumida por el dispositivo en un ensayo con la siguiente secuencia:

- Configuración y alimentación de la memoria.
- Apertura de archivo en la memoria.
- Copia de 20.000 bytes dentro del archivo.
- Cierre del archivo.

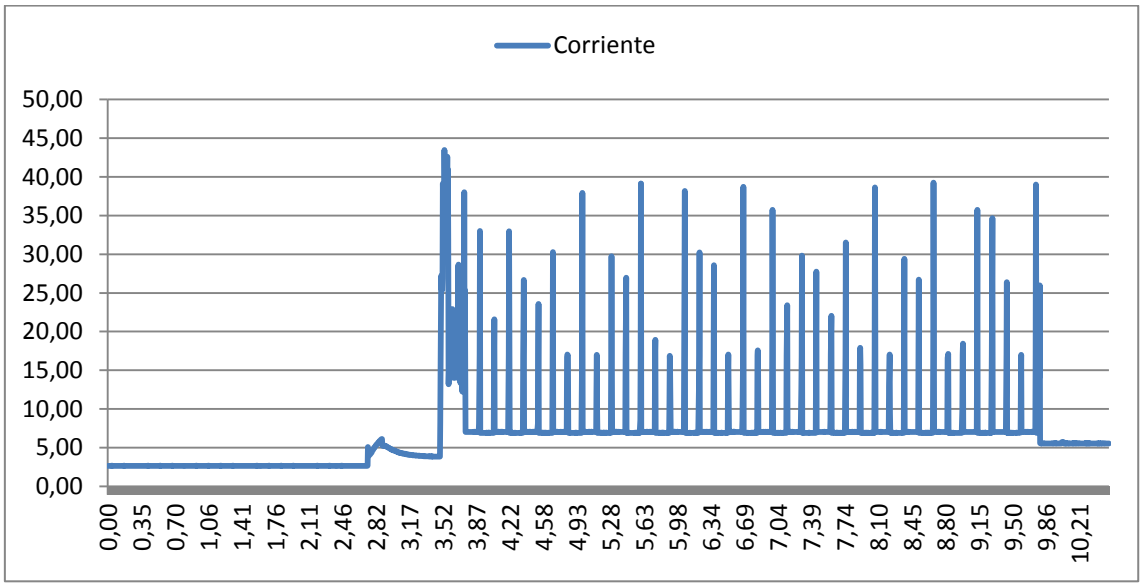


Figura 32.- Consumo de corriente de la memoria durante el guardado de un archivo

Como puede observarse en parte final de la *Figura 32.- Consumo de corriente de la memoria durante el guardado de un archivo*, el consumo de la memoria alimentada con un

archivo cerrado se sitúa entrono a los 2,5 mA. Cuando el archivo permanece abierto, la memoria incrementa su consumo llegando hasta un total de 5 mA aproximadamente.

A parte del consumo continuo que tiene la memoria, se observan claramente los picos de corriente pertenecientes a los ciclos de escritura de la memoria, que ascienden hasta 35 mA aproximadamente.

Realizando una captura de la corriente media medida durante distintos ciclos de escritura, se observa que la corriente media durante la escritura de un archivo es de 8,13 mA.

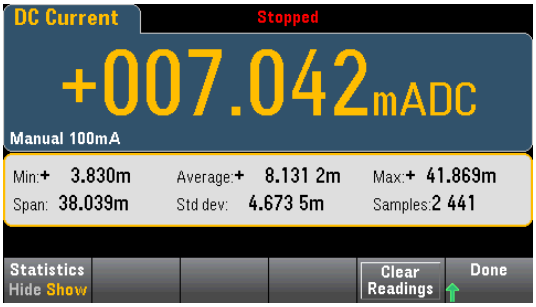


Figura 33.- Corriente media consumida durante la escritura de un archivo en la SD

Con esta corriente podemos estimar la potencia media consumida durante la escritura del archivo a través de la siguiente fórmula:

$$P_m = 8,13mA \cdot 3,6v = 29,26mW$$

Conociendo que para copiar 20.000 bytes de información se ha tardado aproximadamente 6 segundos en escribir los datos en la memoria, podemos estimar que la tasa de transferencia de datos es de aproximadamente 3.333 bytes/segundo.

La energía estimada por byte copiado en la memoria, excluyendo el consumo de apertura del archivo, será de:

$$E_{1Byte} = \frac{29,26mW \cdot \frac{1}{3333}s \cdot 1h}{3600s} = 2,43nWh$$

1.2. CONSUMO RN171

El módulo de comunicaciones RN171 según los datos recabados en el apartado de diseño del hardware es la parte del circuito que mayor consumo presenta. Para estimar el consumo de este módulo en cada uno de los estados que pueden presentarse, se decide realizar una serie de medidas:

1. Consumo en la configuración del RN171.
2. Consumo conectado a una WLAN.
3. Consumo transmitiendo datos.

1.2.1 CONSUMO EN LA CONFIGURACIÓN DEL RN171

El módulo RN171 requiere de una serie de configuraciones iniciales, las cuales deben realizarse cada vez que quieran modificarse alguno de los parámetros de configuración de WLAN, servidor FTP etc.

Para estimar el consumo energético de este proceso se realiza un ensayo con la siguiente secuencia:

- Configuración del hardware necesario
- Habilitar la alimentación del RN171.
- Envío y recepción de comandos.
- Deshabilitar la alimentación del RN171.

Como resultado de este ensayo se obtiene la siguiente gráfica:

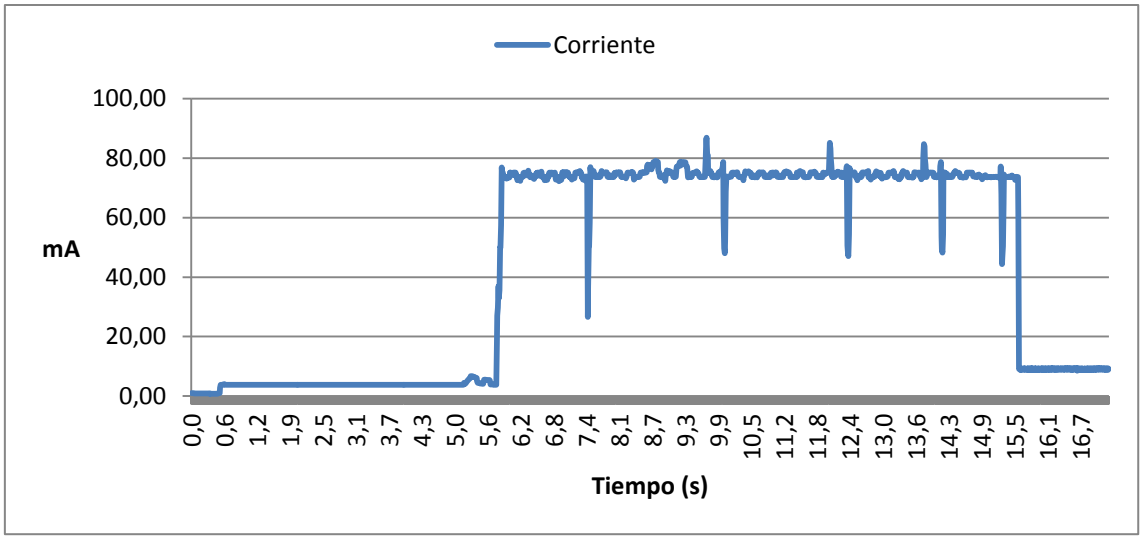


Figura 34.- Consumo corriente configuración inicial RN171

Como se puede observar en la *Figura 34.- Consumo corriente configuración inicial RN171*, el consumo energético se dispara cuando se activa la alimentación del RN171. El consumo total con el módulo WIFI activo, sin conectar a una WLAN, se sitúa en torno a los 75 mA con picos de corriente que pueden superar los 85 mA. También puede observarse como la configuración tiene una duración en torno a 10 segundos.

La energía consumida estimada en este proceso es:

$$E = \frac{75mA \cdot 3,6v \cdot 10s \cdot 1h}{3600s} = 0,625mWh$$

1.2.2 CONSUMO CONECTADO A UNA WLAN

Al conectar el dispositivo a una WLAN, el módulo RN171 comienza a transmitir a través de WIFI. Se presupone que esta transmisión aumentará considerablemente el consumo por lo que se plantea estimar el consumo del módulo en esta situación.

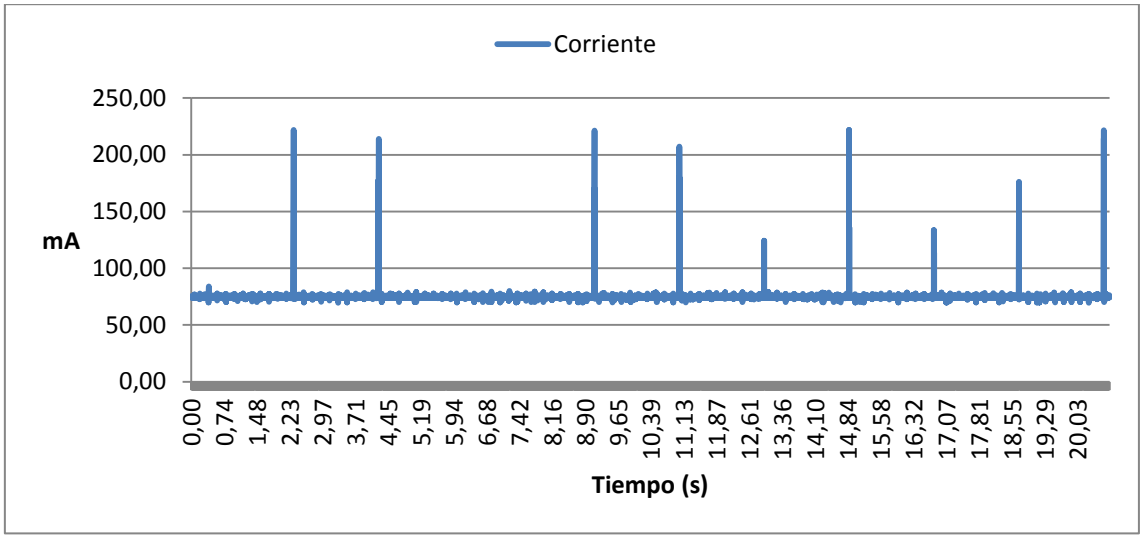


Figura 35.- Consumo corriente RN171 conectado a WLAN

Como se puede observar en la *Figura 35.- Consumo corriente RN171 conectado a WLAN*, el consumo de corriente durante la mayor parte del tiempo se sitúa en torno a los 80 mA, pero en este caso se observan claros picos de corriente de más de 200 mA correspondientes a la transmisión de datos.

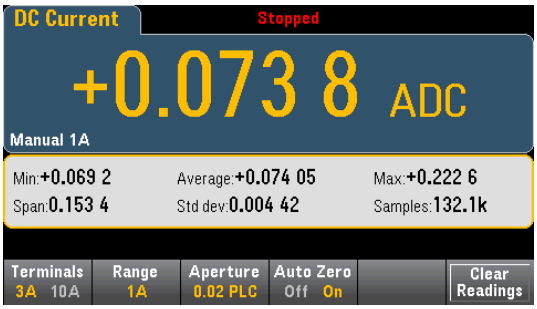


Figura 36.- Captura consumo corriente media RN171 conectado a WLAN

El consumo de corriente medio del RN171 conectado a una WLAN, y sin enviar ni recibir ningún dato más que los imprescindibles para mantenerse conectado, se sitúa aproximadamente en 74 mA, lo que se traduce en una potencia consumida de:

$$P_m = 74mA \cdot 3,6v = 266,4mW$$

1.2.3 CONSUMO TRANSMITIENDO DATOS A FTP

Como se ha visto en el ensayo anterior, el consumo aumenta con la transmisión de datos por lo que se plantea el estimar un consumo medio del módulo RN171 en la transmisión de un archivo a un servidor FTP. Para ello se realiza un muestreo de la corriente durante el envío de un archivo de imagen utilizando el firmware del prototipo.

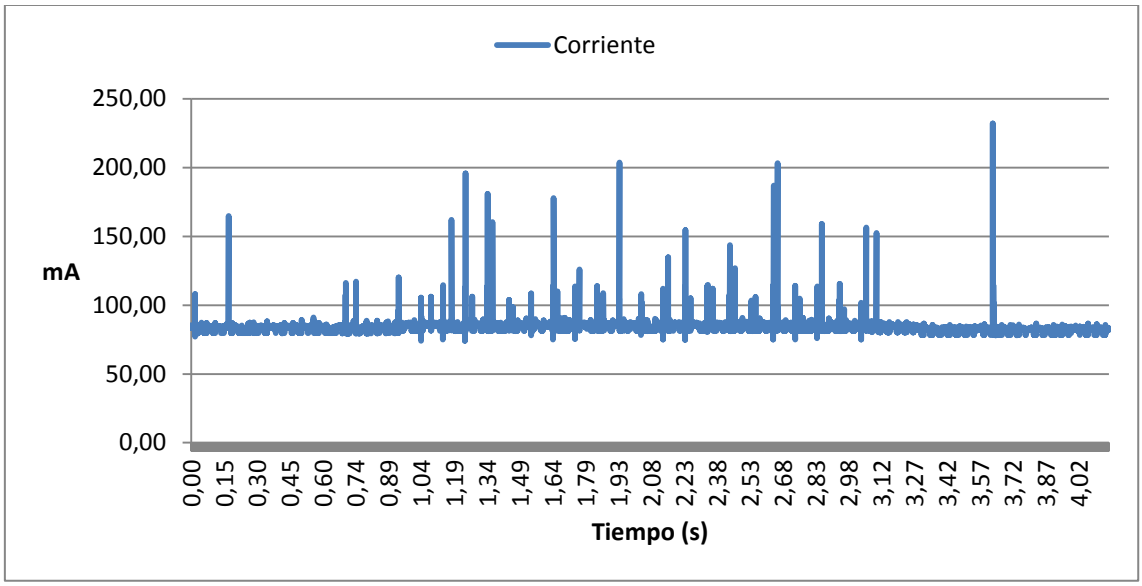


Figura 37.- Consumo corriente RN171 transmitiendo datos

Como se puede observar en la *Figura 37.- Consumo corriente RN171 transmitiendo datos*, vuelven a aparecer los picos de corriente correspondientes a la transmisión de los paquetes de datos. En este caso los picos son más frecuentes, y cómo podemos observar en el multímetro, esto se traduce en un aumento del consumo medio de unos 10 mA, situándose en aproximadamente 83 mA.

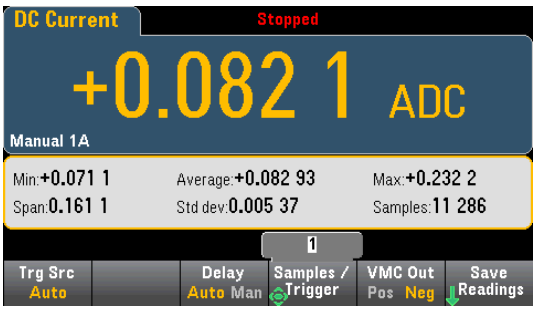


Figura 38.- Captura consumo corriente media RN171 enviando datos

La velocidad de transmisión viene marcada por la UART del módulo. Para este ensayo se ha utilizado la configuración del firmware del prototipo, con una tasa de transferencia de 115200.

1.3. CONSUMO CAMARA JPEG LINKSPRITE

La cámara JPEG es la otra parte del circuito, junto con el módulo WIFI que más corriente consume. Se realiza una medida con la siguiente secuencia:

- Configuración de la cámara
- Tomar fotografía.
- Copiar fotografía en la memoria SD
- Apagado de la cámara.

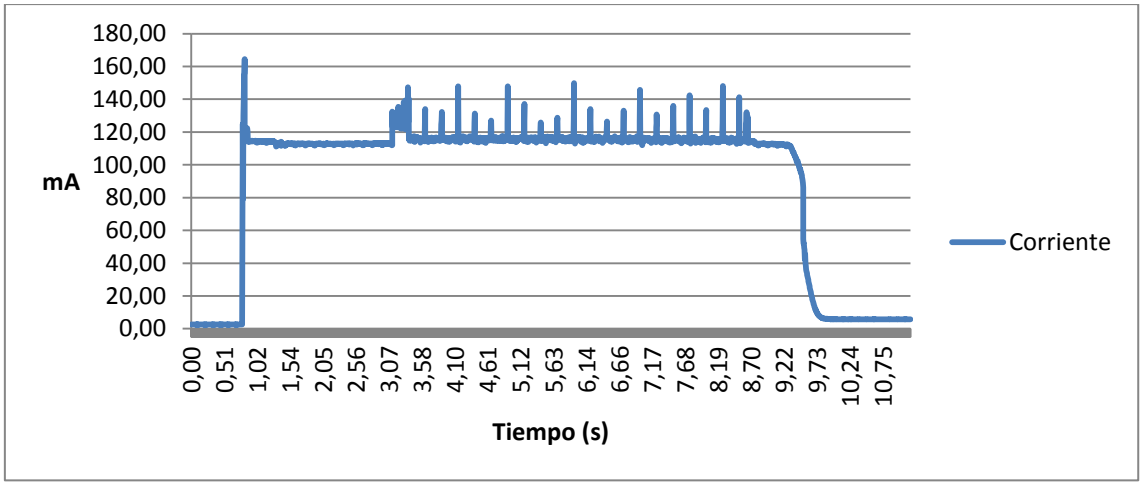


Figura 39.- Consumo corriente cámara JPEG

Como puede observarse en la figura anterior Figura 39.- Consumo corriente cámara JPEG, una vez alimentada la cámara se tiene un consumo de aproximadamente 120 mA. A este consumo se le suma además picos de corriente que llegan hasta los 155 mA correspondientes a la transferencia de los datos de la memoria RAM incluida en la cámara a la memoria SD.

También se puede observar el momento justo de la captación de foto, entorno a los 3 segundos del comienzo de la captación de datos. En este instante se produce un aumento de unos 10 mA durante aproximadamente 40 ms.

El tiempo total que la cámara permanece alimentada es de aproximadamente 9 segundos. Tomando este dato como referencia y estimando un consumo medio durante este tiempo de 120 mA, podemos estimar que la energía necesaria para tomar una fotografía y guardarla en la memoria SD es:

$$E = \frac{120mA \cdot 3,6v \cdot 9s \cdot 1h}{3600s} = 1,08mWh$$

1.4. CONSUMO CAPTURA DE AUDIO

Se plantea un ensayo en el que se mida el consumo de corriente durante la grabación de un clip de audio. Para ello se programa la siguiente secuencia:

- Configuración del conversor ADC y alimentación del hardware de sonido
- Adquisición de 20.000 bytes de datos.
- Desconexión de la alimentación del hardware.
- Copia de los datos de la RAM a la tarjeta SD.

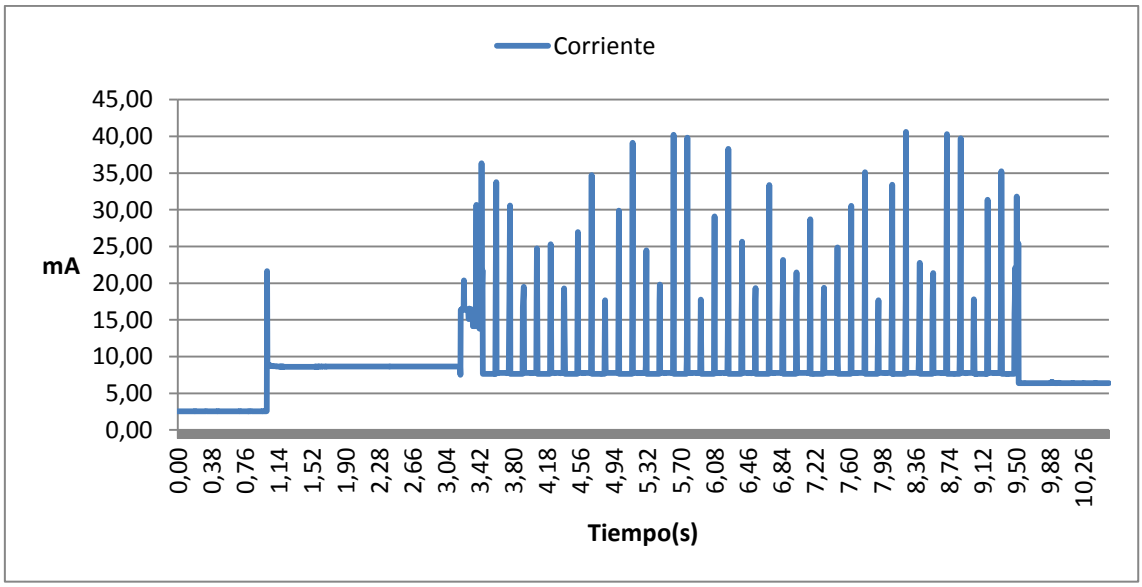


Figura 40.- Consumo de corriente durante la captura de un clip de audio

Como puede observarse en la anterior gráfica, la activación de todo el hardware de sonido, y del conversor ADC del microcontrolador supone un incremento en el consumo de corriente de aproximadamente 6,5 mA, situándose en 9,5 mA en total. Este consumo se mantiene durante los 2 segundos en los que se están capturando datos del conversor.

Llegados a este punto se retira la alimentación del hardware de sonido, pero esta desconexión queda tapada por el incremento de consumo correspondiente a la apertura de un archivo en la SD como se ha mostrado anteriormente. Tras el ciclo de escritura de los datos en la memoria SD, se observa como el consumo desciende hasta 6 mA aproximadamente. La diferencia con los 9,5 mA que se tenían anteriormente corresponde con el consumo del hardware de sonido, y los 3,5 mA restantes al conversor ADC.

El consumo medio aproximado es de unos 10mA, y el proceso completo tiene una duración de unos 9,5 segundos con lo que la energía consumida aproximadamente será:

$$E = \frac{10mA \cdot 3,6v \cdot 9,5s \cdot 1h}{3600s} = 95\mu Wh$$

1.5. CONSUMO PIR

El PIR es el sensor utilizado en el firmware del prototipo para determinar cuándo despertar la electrónica en su modo de disparo por detección. Este elemento debe estar permanentemente alimentado, por lo que su consumo impacta en gran parte en consumo total dispositivo.

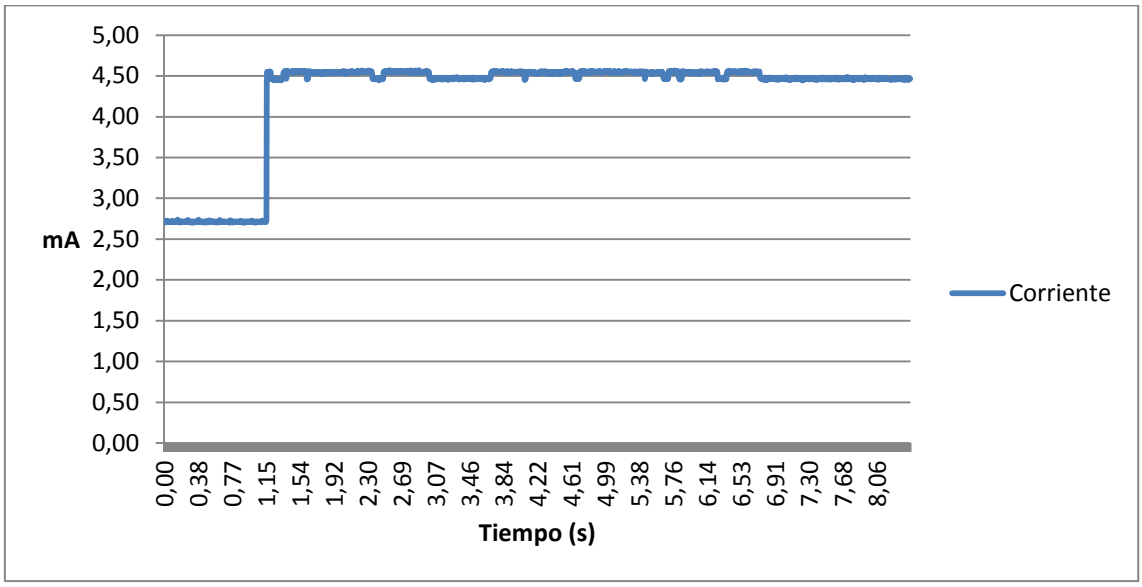


Figura 41.- Consumo de corriente del PIR

Como puede apreciarse en la Figura 41.- Consumo de corriente del PIR, al alimentar el PIR y configurar el periférico de interrupción del microcontrolador, el consumo de corriente aumenta en casi 2mA. Una vez configurado el PIR se puede observar variaciones mínimas de corriente correspondientes a las distintas detecciones de presencia.

1.6. CONSUMO SENSOR TEMPERATURA Y HUMEDAD

Para completar la serie de medidas, y tener así una estimación del consumo energético de cada una de las partes del dispositivo, se realiza una medida del consumo del sensor de temperatura y humedad. La secuencia llevada a cabo es la siguiente:

- Configuración del bus I2C.
- Alimentación del sensor.

- Configuraciones de la medida.
- Medida de temperatura y humedad
- Apagado del sensor.

El sensor cuenta con varios modos de funcionamiento en los que se puede programar una medida continua o medida única. En el modo de medida continua, el sensor va adquiriendo los datos de temperatura y humedad a una frecuencia determinada y permite la consulta del último dato tomado. En el modo de medida única, realiza una única medida y esta es enviada en la consulta posterior.

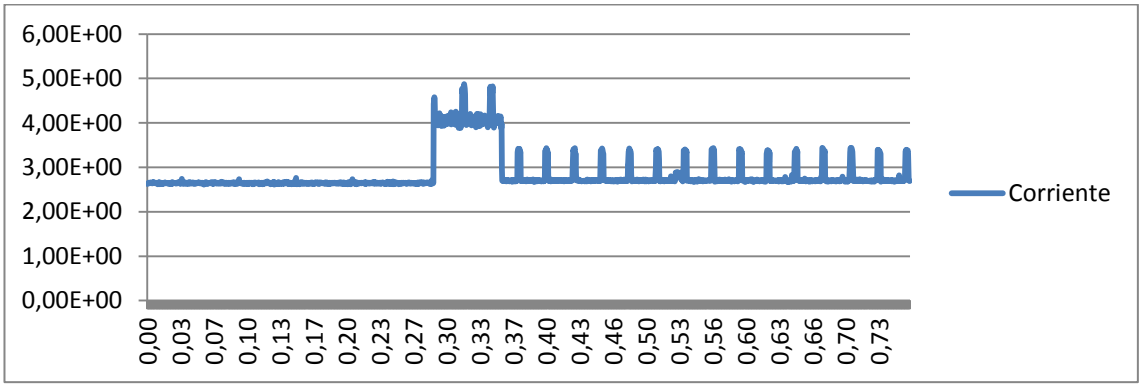


Figura 42.- Consumo de corriente SHT31 modo continuo

Como puede observarse en la figura anterior, el sensor solamente consume unos microamperios, por lo que apenas se aprecia el momento en el que se alimenta el sensor. A continuación se incrementa el consumo de corriente en algo más de 1 mA con motivo del intercambio de datos a través del bus I2C. Durante el intercambio de datos se observa como el sensor comienza ya ha realizar medidas. Estas se reflejan en los picos que suben el consumo total hasta casi 5 mA. Estos picos se repiten con una frecuencia constante debido a que el sensor está configurado en el modo continuo.

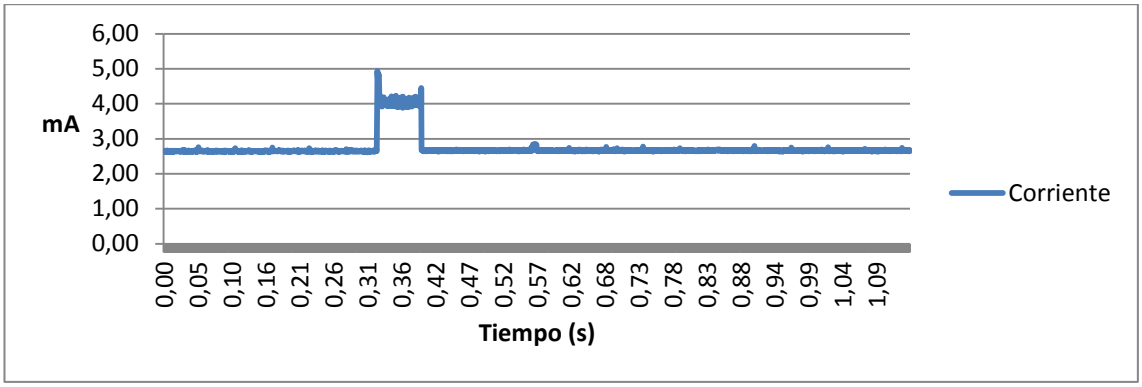


Figura 43.- Consumo de corriente SHT31 modo único

Si se observa la gráfica del modo único, se puede identificar los mismos consumos que en el modo continuo, exceptuando los picos correspondientes a las medidas de temperatura y humedad cuando se dejan de intercambiar datos.

En el modo de disparo único, realizando todas las configuraciones, la medida tarda tan solo 30 ms en completarse. Si estimamos que el cosumo de corriente medio está en torno a 1,2 mA, se producirá un consumo energético total de:

$$E = \frac{1,2mA \cdot 3,6v \cdot 0,3s \cdot 1h}{3600s} = 36nWh$$

1.7. Consumo estimado por captura de datos

Teniendo en cuenta los resultados obtenidos en el apartado anterior se puede hacer una estimación de la energía consumida en la captura total de los datos y su guardado en la tarjeta SD. Para realizar esa estimación se suman los consumos por separado de cada uno de los procesos.

Tabla 16.- Consumo de los distintos proceso

Proceso	Energía necesaria
Tomar una fotografía y almacenarla en la SD	$1,08mWh$
Grabar un clip de audio y almacenado en la SD	$95\mu Wh$
Dato de temperatura y humedad y almacenado en la SD	$36nWh$

TOTAL	$1{,}175 mWh$

1.8. CONSUMO DEL PROTOTIPO EN LOS DISTINTOS MODOS DE FUNCIONAMIENTO

Para la estimación del consumo total del dispositivo, se distinguen dos situaciones claras cuando se detecta presencia y cuando no. Estas dos situaciones solo afectan al modo de funcionamiento por disparo, ya que en el caso de Time-Lapse, el dispositivo siempre permanece activo.

1.8.1 CONSUMO DEL PROTOTIPO EN MODO TIME-LAPSE

En este modo se realizan capturas periódicas de todos los sensores, y estos datos son enviados al servidor FTP nada más acabar de ser capturados. El tiempo configurado entre captura y captura en el prototipo es 1 minuto.

El consumo energético de la captura completa de datos y su guardado en la memoria SD es de $1{,}175mWh$ según lo estimado en el apartado *1.7*. .Este proceso dura aproximadamente 20 segundos.

El tiempo medio de transmisión de todos los datos al servidor FTP es también de 20 segundos. Teniendo en cuenta la estimación del consumo del apartado *1.2.3 CONSUMO TRANSMITIENDO DATOS A FTP* el consumo de corriente medio subiendo los archivos al servidor FTP es de 83mA. Por tanto la energía necesaria estimada para subir un conjunto de datos al servidor será de:

$$E = \frac{83mA \cdot 3,6v \cdot 20s \cdot 1h}{3600s} = 1,66mWh$$

Completados estos dos procesos quedan 20 segundos de espera hasta la siguiente captura, en este tiempo se desactiva todo el hardware, y solo permanece activo el microcontrolador cuyo consumo medio medido es de 4,158 mA según el apartado CONSUMO SAM4LC4*1.1 CONSUMO SAM4LC4.* El consumo en este periodo será de:

$$E = \frac{4,158mA \cdot 3,6v \cdot 20 \cdot 1h}{3600s} = 0,083mWh$$

Si se realiza la suma de estos tres procesos se obtiene el consumo energético por ciclo:

$$E = 1,175mWh + 1,66mWh + 0,083mWh = 2,918mWh / ciclo$$

El consumo en 1 hora será de:

$$E_{TIME\_LAPSE} = 2,918mWh / ciclo \cdot 60ciclos = 175,08mWh$$

Y el consumo en un día seria de:

$$4,201Wh/día$$

1.8.2 CONSUMO DEL PROTOTIPO EN MODO DETECCIÓN

En este modo las únicas partes activas, mientras no se detecte presencia, son el microcontrolador y el PIR que es utilizado como elemento de detección. El consumo medio medido en este modo es de 4,5 mA según lo expuesto en el apartado *1.5 CONSUMO PIR*.

El consumo energético en este modo y mientras no se detecte presencia será de:

$$E = 4,5mA \cdot 3,6v \cdot 1h = 16,2mWh$$

En este modo, cuando se detecta presencia en la estancia el prototipo realiza una captura de todos los sensores. Estos archivos son transferidos al servidor FTP en grupos de 250 archivos máximo.

Estimando un tiempo de detección de 1 horas al día, con un tiempo de 20 segundos por captura, se generan 180 capturas al día. Los archivos de estas capturas serán subidos en 1 vez al servidor FTP, con una duración de 20 segundos por captura.

Si consideramos los mismos consumos que en el apartado anterior la energía consumida en este modo en un día sería la siguiente:

$$E_{MODO\_DETECCION} = ((1,175 + 1,66mWh) \cdot 180) + (16,2mWh \cdot 22h) = 866,7mWh/día$$

Este consumo puede reducirse más en la aplicación final utilizando el modo de bajo consumo “BackUp” del microcontrolador. En este modo la corriente consumida cuando no hay detección se reduce hasta los 884,25 µA. Si se utilizase este modo, la energía consumida por el dispositivo en un día seria de:

$$E_{MODO\_BACKUP} = ((1,175 + 1,66mWh) \cdot 180) + (884,25\mu A \cdot 3,6v \cdot 22h) = 580,31mWh/día$$

2. WIFI

2.1. ANTENA PCB

A fin de comprobar el alcance de la cobertura WIFI con la antena implementada en el ruteo de la PCB, se plantea realizar una serie de medidas de la señal RSSI con el módulo

conectado a un punto de acceso WIFI. Para realizar estas medidas se utiliza un conversor USB a UART, para conectar la UART del módulo RN171 a un PC y lanzar los comandos a través de un terminal serie.

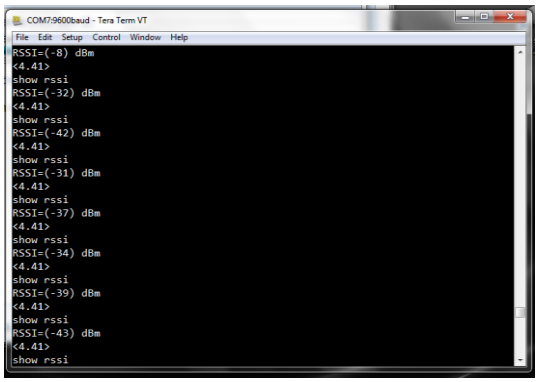


Figura 44.- Terminal serie captando señales RSSI

Figura 45.- RN171 con TTL-232R-3V3

Como punto de acceso se utiliza un router ZyXEL MOD. P660HW-B1A, instalado dentro de un hogar. Los distintos puntos de medida se han representado en la *Figura 46.- Posiciones de medida de señal RSSI.* A través del terminal serie se van midiendo los distintos valores de señal RSSI en la misma habitación e instancias contiguas de la casa, obteniendo los siguientes resultados:

Tabla 17.- Medidas señal RSSI

Punto de medida	Distancia y situación	RSSI (dBm)
1	Justo al lado de la antena del router	-7
2	A 1 metro del router, dentro de la misma habitación	-34
3	A 2 metros del router, dentro de la misma habitación	-39
4	A 5 metros del router, fuera de la habitación	-47
5	A 2 metros del router, en la habitación contigua	-43

6	A 4 metros del router, en la habitación contigua	-45
7	A 5 metros del router en la habitación contigua	-47
8	A 7 metros del router, con dos paredes de por medio	-51
9	A 7 metros del router, con dos paredes y una habitación de por medio	-54
10	A 10 metros del router, con dos paredes y una habitación de por medio	-54

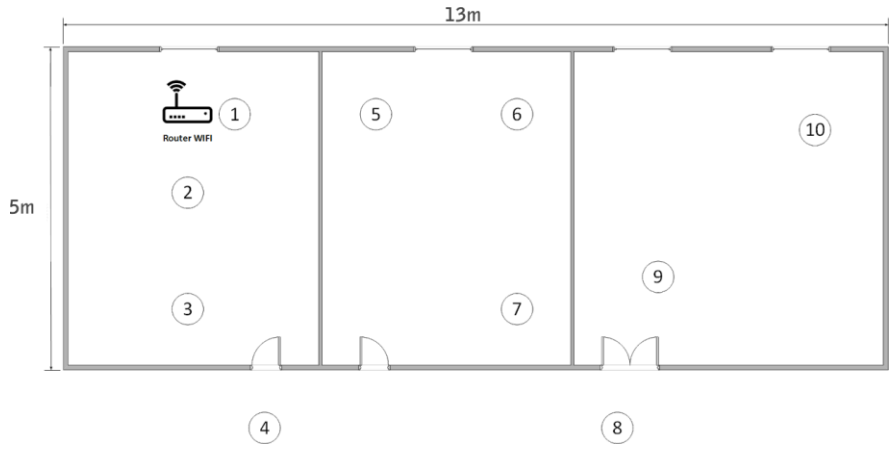


Figura 46.- Posiciones de medida de señal RSSI

Los resultados obtenidos confirman que es viable utilizar esta antena, al menos para distancias y atenuaciones similares a las dadas en el ensayo, ya que la señal RSSI no ha sobrepasado en ningún momento los -60 dBm que es el límite habitualmente utilizado para hablar de una señal estable 52[42].

2.2. CONEXIONES FALLIDAS AL SERVIDOR FTP

Durante el desarrollo del firmware se ha detectado que el módulo RN171 falla bastante al intentar abrir un archivo en un servidor FTP. Par solucionar esta problemática en el firmware se ha implementado que si la apertura del archivo falla, se vuelva a reintentar hasta en 4 ocasiones.

Para cuantificar en qué porcentaje se producen estos fallos, en el firmware hay una serie de variables con contabilizan los intentos de conexión y los archivos transferidos correctamente.

En este ensayo se ha configurado el prototipo en modo Time-Lapse y se ha puesto capturar datos y enviarlos al servidor FTP durante aproximadamente 3 horas, con una configuración de 1 captura por minuto.

El servidor FTP ha sido simulado en un PC mediante el software XAMPP [43]. Este PC se ha conectado mediante un cable Ethernet a un router ZyXEL MOD. P660HW-B1A, el cual hace las veces de punto de acceso WIFI al que se conecta el prototipo.

En la configuración del router, al PC que simula el servidor FTP se le ha asignado dirección IP fija “192.168.1.36”, la cual ha sido usada como dirección del servidor FTP en la configuración del módulo RN171. Para poder comprobar que los datos se han transferido correctamente a este servidor simulado se ha utilizado el software gratuito FileZilla [44].

Después de cada serie de intentos de subir un archivo al servidor, se ha creado un archivo .txt en la memoria SD con el que poder obtener de manera rápida los datos de las estadísticas.

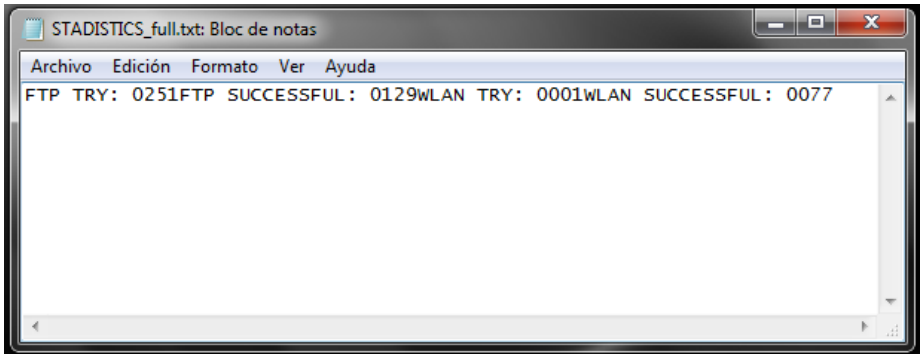


Figura 47.- Captura de pantalla archivo estadísticas servidor FTP

Los datos de estas estadísticas expresados en tanto por ciento son los siguientes:

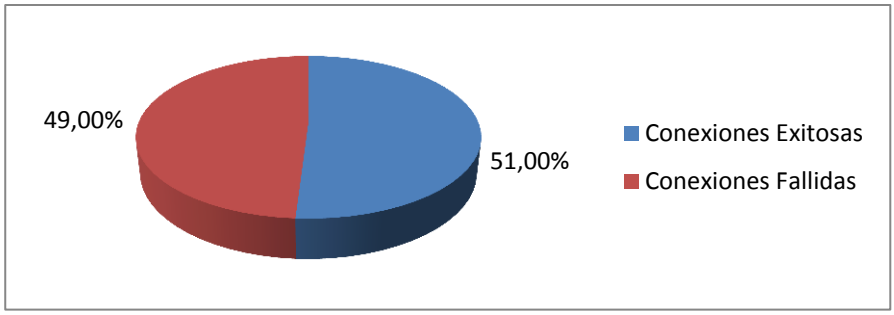


Figura 48.- Porcentaje de conexiones fallidas servidor FTP

Como se ve en el gráfico anterior, según los datos descargados del prototipo, en un total de 251 intentos de guardado de archivo en el servidor FTP el 49% de las conexiones han sido erróneas. Con los reintentos implementados en el firmware se han conseguido guardar todos los archivos que se pretendía ya que tras los fallos producidos, se ha vuelto a intentar enviar el mismo archivo hasta un número máximo de 4 veces.

Algunas de las fotos subidas al servidor FTP se muestran en la *Figura 49.- Collage de algunas fotografías subidas al servidor FTP.*

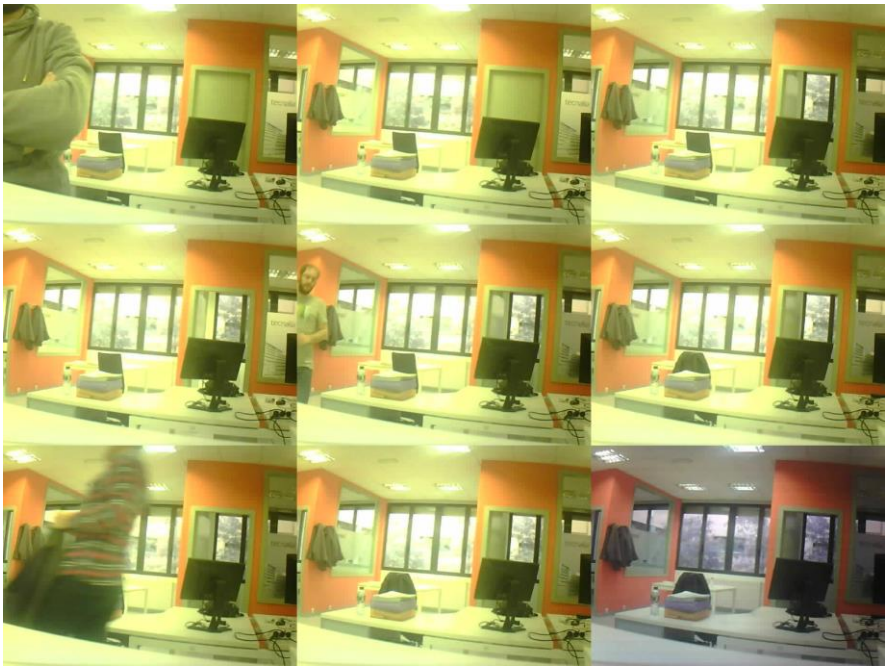


Figura 49.- Collage de algunas fotografías subidas al servidor FTP

3. MICRÓFONO

Para validar el funcionamiento del micrófono y la etapa preamplificadora posterior, se plantea un ensayo que consista en emitir tonos puros a través de un altavoz para captar este sonido mediante el micrófono y poder ver la señal obtenida al final de la etapa de pre amplificación. Esta señal es la que le llega al conversor analógico-digital del microcontrolador.

Para medir estas señales se utiliza un osciloscopio TEKTRONIX TBS1052B. Este osciloscopio permite realizar una FFT en tiempo real de las señales capturadas en uno de los canales.

Se comienza por tomar una captura el ruido existente a la salida de la etapa sin que exista ningún sonido ambiente.

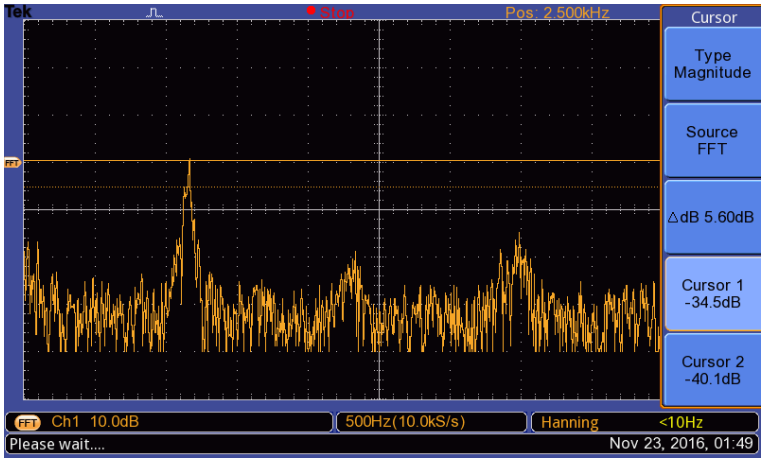


Figura 50.- Ruido a la salida de la etapa pre amplificadora de audio

Como puede observarse en la gráfica, con silencio absoluto aparece el ruido con su pico máximo a una frecuencia de unos 1,6 KHz y -34,5 dB.

A continuación a través de un altavoz situado a unos 20 centímetros se reproduce una onda sonora de 1KHz y se vuelve a repetir la medida.

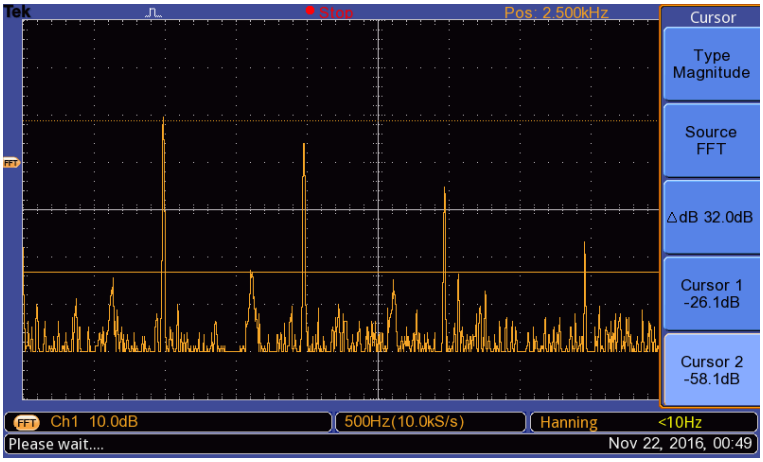


Figura 51.- FFT señal etapa pre amplificadora con señal acústica de 1KHz

En la FFT de la señal se puede observar cómo se distingue perfectamente la señal de 1KHz, así como sus armónicos. En esta gráfica aparece también la señal de ruido con un valor de -58,1 dB. Cogiendo el pico máximo que vuelve a situarse en torno a los 1,6KHz, la relación señal-ruido es de 32dB.

Se puede observar como el ruido ha disminuido debido a la ganancia adaptativa de la etapa pre amplificadora. Para comprobar este comportamiento de la etapa, se va reduciendo el volumen de la señal emitida por el altavoz y se comprueba como la señal en la salida de la etapa pre amplificadora se mantiene.

Se repiten estas pruebas para una señal de 2 KHz y se obtienen los mismos resultados.

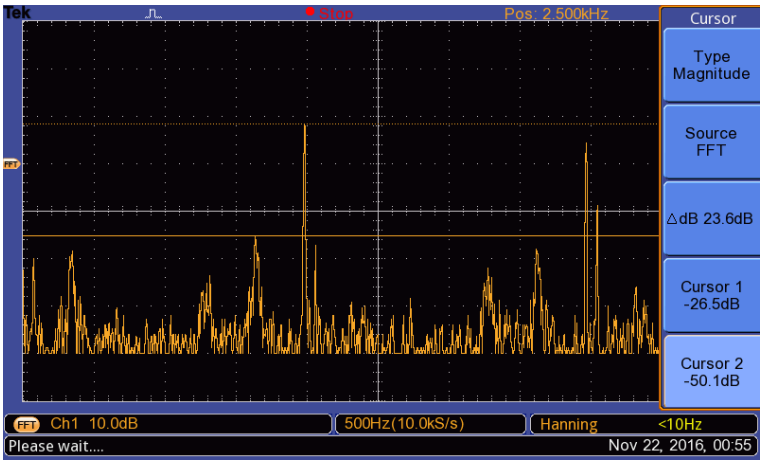


Figura 52.- FFT señal etapa pre amplificadora con señal acústica de 2 KHz

Llegado este punto se comprueba que el micrófono es capaz de captar la voz humana, con un volumen de voz hablada normal de conversación y a una distancia entre 1 y 4 metros.

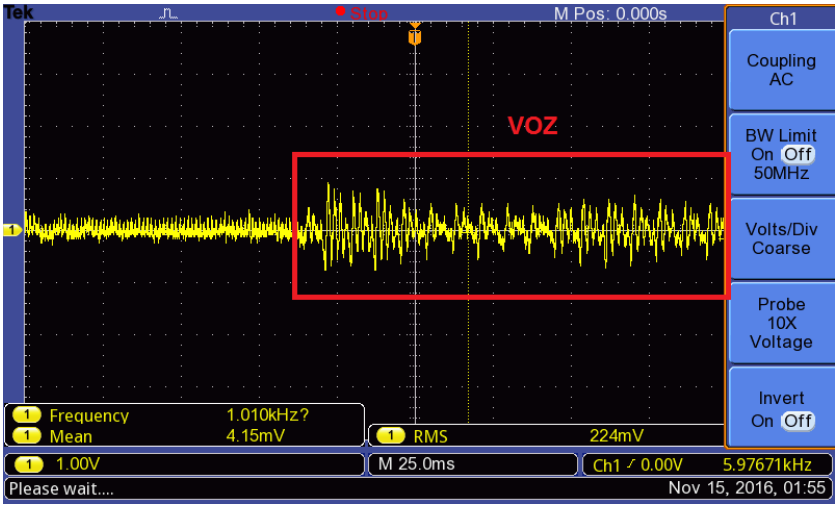


Figura 53.- Señal etapa pre amplificadora de audio con voz hablada

Como puede observarse en la gráfica el micrófono es capaz de captar la voz hablada del sonido ambiente.

ANEXO 4.- Código fuente

1. MAIN

```
/**
 * \file
 *
 * \brief Multisensor Wireless
 *
 */



/*
 * Include header files for all drivers that have been imported from
 * Atmel Software Framework (ASF).
 */
/*
 * Support and FAQ: visit <a href="http://www.atmel.com/design-support/">Atmel Support</a>
 */
#include <asf.h>

#include "main.h"

#include "HMI/HMI.h"
#include "SHT/SHT31.h"
#include "ENERGY MGT/ML_PowerMgt.h"
#include "AUDIO/ML_uPhone.h"
#include "SHT/SHT31.h"
#include "SD MEMORY/ML_SDcard.h"
#include "WIFI/ML_Wifi.h"
#include "WIFI/RN171.h"
#include "CAMERA/ML_Camera.h"


#include <string.h>

extern bool RN171_WAIT_RESPONSE;
extern bool Camera_Busy;
extern bool Wifi_Busy;
extern bool NET_JOINED;
extern uint8_t MAC_ADD[12];
extern uint8_t RTC_VALUE[10];

//// Private functions
void Set_FileName (void);
void UploadStoringData (void);
void CaptureSensors (void);
void IncreaseRTC (void);



uint32_t get_tick_count(void);

/// DELATE, ONLY FOR DEBUG
void PrintStadistic(void); ////////////////////////////DELATE
 uint32_t FTP_TRY_STADISTIC=0;
 uint32_t FTP_SUCCESFUL_STADISTIC=0;

 uint32_t WLAN_TRY_STADISTIC=0;
 uint32_t WLAN_SUCCESFUL_STADISTIC=0;
///

uint32_t second_count=0;
uint8_t  ticks_to_second=0;

uint8_t nFilesDemostrator=0;


CAPTURE_MODE ML_MODE=MODE_CONTINUOUS;

/** Tick Counter in unit of ms. */
uint32_t g_ul_tick_count;
/** System tick frequency */
#define SYS_TICK_FREQ        4

uint16_t Vbat;
uint16_t Vin;
uint16_t register_value;
```

```
uint8_t nFiles=0;
char new_file_name[32];

bool PIR_EVENT = false;


/**
 *  \brief Handler for System Tick interrupt.
 *
 *  Process System Tick Event.
 *  Increment the ul_ms_ticks counter.
 */
void SysTick_Handler(void)
{
	g_ul_tick_count++;
	ticks_to_second++;
	if (ticks_to_second>5)
	{
		second_count++;
		ticks_to_second=0;
	}


		Blink_Leds (); // Blink slected LEDs.


	Check_Shutdown_Rqst ();
}



int main (void)
{



	board_init();

	// Insert system clock initialization code here (sysclk_init()).
	sysclk_init();
	configure_systick();
	Init_HMI ();
	Led_blink(RED_LED_PIN,ENABLE);
	////// INIT ////////////////////////////////////////////////////////////////

	SDcard_Init();

	Wifi_StartUp_config ();

	Led_reset(RED_LED_PIN);


	JoinWlan ();
	RefreshTimeStamp ();
	Wifi_Sutdown_Rqst ();

	Set_FileName ();



	while(1){

		switch(ML_MODE){
			case MODE_SINGLE:
				if(PIR_EVENT){
					CaptureSensors();
					PIR_EVENT=false;
				}
				if((second_count>=(UPLOAD_TIME*3600)) || (nFiles>=MAX_FILES_TO_ULOAD)){
					JoinWlan ();

					RefreshTimeStamp ();
					UploadStoringData();

					Wifi_Sutdown_Rqst ();
					second_count=0;
				}
```

```
                                        break;

                                case MODE_CONTINUOUS:
                                        if(second_count>=(LAPSE_TIME-CAPTURE_TIME)){
                                                CaptureSensors();
                                                JoinWlan ();
                                                if(NET_JOINED){
                                                        UploadStoringData();


                                                        RefreshTimeStamp ();

                                                }
                                                Wifi_Sutdown_Rqst ();
                                                second_count=0;
                                        }
                                        break;

                        }


                }

        }

void Set_FileName (void){
        uint8_t i=0;

        for(i=0; i<12; i++){
                new_file_name[i]= MAC_ADD[i];
        }
        for(i=12; i<22;i++){
                new_file_name[i]= RTC_VALUE[i-12];
        }
        uint8_t nFiles_CEN, nFiles_DEC, nFiles_UNI;
        nFiles_CEN = nFiles/100;
        nFiles_DEC = (nFiles- (nFiles_CEN*100))/10;
        nFiles_UNI = (nFiles- (nFiles_CEN*100) - (nFiles_DEC*10));

        new_file_name[22] = nFiles_CEN+48;
        new_file_name[23] = nFiles_DEC+48;
        new_file_name[24] = nFiles_UNI+48;

        nFiles++;
}

void UploadStoringData (void){

        Wifi_Busy=true;
        while (nFiles>0){

                nFiles=nFiles-1;
                uint8_t nFiles_CEN, nFiles_DEC, nFiles_UNI;
                nFiles_CEN = nFiles/100;
                nFiles_DEC = (nFiles- (nFiles_CEN*100))/10;
                nFiles_UNI = (nFiles- (nFiles_CEN*100) - (nFiles_DEC*10));

                new_file_name[22] = nFiles_CEN+48;
                new_file_name[23] = nFiles_DEC+48;
                new_file_name[24] = nFiles_UNI+48;

                new_file_name[25]='P';
                new_file_name[26]='I';
                new_file_name[27]='C';
                new_file_name[28]='.';
                new_file_name[29]='j';
                new_file_name[30]='p';
                new_file_name[31]='g';
                SDfileToFTP (&new_file_name,  PICTURE);

                new_file_name[25]='A';
                new_file_name[26]='U';
                new_file_name[27]='D';
                new_file_name[28]='.';
                new_file_name[29]='t';
                new_file_name[30]='x';
                new_file_name[31]='t';
                SDfileToFTP (&new_file_name,  AUDIO);
```

```
                    new_file_name[25]='R';
                    new_file_name[26]='H';
                    new_file_name[27]='T';
                    new_file_name[28]='.';
                    new_file_name[29]='t';
                    new_file_name[30]='x';
                    new_file_name[31]='t';
                    SDfileToFTP (&new_file_name,  TRHPIR);

          }
          Wifi_Busy=false;
}
void CaptureSensors (void){
          Set_FileName();

          PictureToSD(&new_file_name);

          AudioClipToSD (&new_file_name);

           Tem_Hum_ToSD (&new_file_name);

}

/**
 *  Configure system tick to generate an interrupt every 1s.
 */
static void configure_systick(void)
{
          uint32_t ul_flag;

          ul_flag = SysTick_Config(sysclk_get_cpu_hz() / SYS_TICK_FREQ );
          if (ul_flag) {
                    while (1) {
                    }
          }

          NVIC_SetPriority(SysTick_IRQn , 5);
}

void wait(volatile uint32_t ul_ms)
{
          uint32_t ul_start;
          uint32_t ul_current;

          ul_start = g_ul_tick_count;
          do {
                    ul_current = g_ul_tick_count;
          } while (ul_current - ul_start < ul_ms);
}
/**
 *  \brief Get the tick count value.
 *
 */
uint32_t get_tick_count(void)
{
          return g_ul_tick_count;
}
```

```
/*
 * main.h
 *
 * Created: 19/10/2016 21:07:29
 *  Author: 108778
 */


#ifndef MAIN_H_
#define MAIN_H_


#define DEMOSTRATOR

#ifdef DEMOSTRATOR
                #warning ("Warning: DEMOSTRATOR MODE define in main.h")
#endif

#define UPLOAD_TIME                                     12 //// Hours between files uploads

#define MAX_FILES_TO_ULOAD          1 /// Max. number of files to be upload/connection

#define LAPSE_TIME                          60

#define CAPTURE_TIME                        30 /// SET THIS VALUE TESTING THE CAPTURE PROCESS
 /// Lapse time in seconds for continuous mode capture

typedef enum{
        MODE_SINGLE=0,
        MODE_CONTINUOUS,
}CAPTURE_MODE;

void wait(volatile uint32_t ul_ms);

#endif /* MAIN_H_ */
```

2. WIFI

```
/*
 * ML_Wifi.c
 *
 * Created: 01/11/2016 12:31:29
 *  Author: 108778
 */
#include "asf.h"
#include "compiler.h"
#include "ML_Wifi.h"
#include "RN171.h"
#include "main.h"
#include "ENERGY MGT/ML_PowerMgt.h"
#include "HMI/HMI.h"
#include "WIFI_TIMERS.h"

/// External variables


extern bool Wifi_Busy;
extern MODULE_STATE RN171_STATE;
extern uint32_t Wifi_UART_Baudrate;
extern UART_CONFIG Initial_UART_Config;

extern bool NET_JOINED;
extern bool FTP_FILE_OPEN;
extern bool RN171_TIMEOUT;

WIFI_CONFIG WIFI_STATE = WIFI_UNCONFIGURED;

void Wifi_StartUp_config (void){
        Wifi_Busy=true;
        Wifi_UART_Baudrate = WIFI_UART_DEFAULT_BAUDRATE;
        Initial_UART_Config = DEFAULT_CONFIG;
        RN171_Init();
        if (RN171_STATE== MODULE_RUN)
```

```
        {
                RN171_ChangeBaudRate (BR115200);
                //RN171_ChangeParity ();
        }
        Wifi_UART_Baudrate = WIFI_UART_HS_BAUDRATE;
        Initial_UART_Config = ML_CONFIG;
        RN171_Power(DISABLE);
        wait(1);
        RN171_Init();
        RN171_NetParameters_config (&WLAN_default_ssid, &WLAN_default_pass);
        RN171_FTP_Config (&FTP_default_address, &FTP_default_folder, &FTP_default_user, &FTP_default_pass);
        RN171_SNTP_Config (&SNTP_default_address, SNTP_default_zone);
        RN171_GetMac ();

        Wifi_Busy=false;

        WIFI_STATE = WIFI_CONFIGURED;
        //Wifi_Sutdown_Rqst ();
}

void JoinWlan (void){
        if(RN171_STATE!=MODULE_RUN && RN171_STATE!=MODULE_CMD){
                RN171_Init();
        }
        uint8_t nRetry=0;

        while(!NET_JOINED && (nRetry<4)){
        RN171_Join_Wlan ();
        nRetry++;
        }
        if (NET_JOINED)
        {
        }
}

void RefreshTimeStamp (void){
        if (NET_JOINED){
        RN171_GetTimeStamp();
        }
}

void SDfileToFTP (char *file_to_transf,  FILE_TYPE file_type_selected){

        bool File_exist=false;
        static uint16_t SD_data=0;
        char SD_data_array[2];
        uint8_t nRetry=0;
        uint32_t nData=0, File_Size=0;

        File_exist = SD_OpenFile (file_to_transf, NO_OVERWRITE);

        if(File_exist){

                //// ADD IF NO JOINED
        nRetry=0;
        while(!FTP_FILE_OPEN && nRetry<3){
                RN171_FTP_NewFile (file_to_transf);
                nRetry++;

        }

                if (FTP_FILE_OPEN){
                switch (file_type_selected){
                        case PICTURE:
                                SD_data  = 0;
                                Led_set(GREEN_LED_PIN);
                                while((FTP_FILE_OPEN) && (SD_data!=0xffd9)){

                                        SD_data_array[0]=SD_data_array[1];

                                        SD_data_array[1]= SD_ReadFromFile ();
                                        SD_data=SD_data<<8|SD_data_array[1];

                                        RN171_FTP_FileWrite (SD_data_array[1]);


                                }
                                Led_reset(GREEN_LED_PIN);
                                RN171_StartTimeOut();
```

```
                                        while(FTP_FILE_OPEN && !RN171_TIMEOUT);
                                        if(RN171_TIMEOUT){
                                                FTP_FILE_OPEN=false;
                                                Led_reset(BLUE_LED_PIN);
                                        }
                                break;

                                case AUDIO:
                                        Led_set(GREEN_LED_PIN);
                                        File_Size=20000;
                                        while(FTP_FILE_OPEN && (nData<File_Size)){


                                                SD_data_array[1]= SD_ReadFromFile ();

                                                RN171_FTP_FileWrite (SD_data_array[1]);

                                                nData++;
                                        }
                                        Led_reset(GREEN_LED_PIN);
                                        while(FTP_FILE_OPEN && !RN171_TIMEOUT);
                                        if(RN171_TIMEOUT){
                                                FTP_FILE_OPEN=false;
                                                Led_reset(BLUE_LED_PIN);
                                        }
                                        break;

                                case TRHPIR:
                                Led_set(GREEN_LED_PIN);
                                File_Size=10;
                                while(FTP_FILE_OPEN && (nData<File_Size)){

                                        SD_data_array[1]= SD_ReadFromFile ();

                                        RN171_FTP_FileWrite (SD_data_array[1]);

                                        nData++;
                                }
                                Led_reset(GREEN_LED_PIN);
                                while(FTP_FILE_OPEN && !RN171_TIMEOUT);
                                if(RN171_TIMEOUT){
                                        FTP_FILE_OPEN=false;
                                        Led_reset(BLUE_LED_PIN);
                                }
                                break;

                        }
                        //Delay_to_ToR();
                        /// return TRANS_OK
                        }
                        if (!FTP_FILE_OPEN)
                        {
                                /// return ERR FTP
                        }


                }

}

/*
 * ML_Wifi.h
 *
 * Created: 01/11/2016 12:31:48
 *  Author: 108778
 */


#ifndef ML_WIFI_H_
#define ML_WIFI_H_


typedef enum{
         WIFI_CONFIGURED=0,
         WIFI_UNCONFIGURED,
} WIFI_CONFIG;

typedef enum{
         PICTURE=0,
```

```
        AUDIO,
        TRHPIR,
} FILE_TYPE;



/******************** Exported Functions **********************/
void Wifi_StartUp_config (void);

void JoinWlan (void);

void RefreshTimeStamp (void);

void SDfileToFTP (char *file_to_transf,  FILE_TYPE file_type_selected);

#endif /* ML_WIFI_H_ */

/**
 * \file
 *
 * \brief RN171 Driver
 *
 * Copyright (c) 2015 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */

#include "asf.h"
#include "compiler.h"
#include "RN171.h"
#include "WIFI/WIFI_TIMERS.h"
#include "HMI/HMI.h"
#include "main.h"
#include <stdio.h>
#include <string.h>

#ifdef DEBUGMODE
        #include <stdio_serial.h>
        #include "MLSerialApp.h"
#endif


/** Private Functions **/

void RN171_CheckMsg(void);
void Ready_to_reception (uint8_t nBytes, ANSWER_TYPE ASW);
void Find_RTC_Value (void);
void RN171_CMD_MODE (void);
void RN171_SaveConfig (void);
void RN171_Reboot(void);

uint8_t MAC_ADD[12]="000000000000";
uint8_t IP_ADD[12]= "000000000000";
uint8_t RTC_VALUE[10]="0000000000";

bool MAC_ADD_READED=false;
/** Modes of RN171**/


NET_STATE WLAN_STATE = NET_UNCONNECTED;
ANSWER_TYPE COMMAND_ANSWER = ANSWER_SPECIAL;
MODULE_STATE RN171_STATE = MODULE_POWERUP;


bool RN171_ERR = false;
bool RN171_CMD_OK = false;

bool NET_JOINED = false;
bool FTP_FILE_OPEN = false;

bool RN171_TIMEOUT = false;
bool RN171_DELAY = false;
```

```
/** Defines for RX BUFFER **/

uint8_t WIFI_RX_INDEX=0;
uint8_t WIFI_RX_BYTE_TO_TRANSFER=0;

#define WIFI_MAX_BITS_TO_RECIVE 100

uint32_t WIFI_RX_BUFFER[WIFI_MAX_BITS_TO_RECIVE];

uint32_t *WIFI_RX_BUFFER_POINTER= &WIFI_RX_BUFFER[0];

uint32_t Wifi_UART_Baudrate = WIFI_UART_DEFAULT_BAUDRATE;
UART_CONFIG Initial_UART_Config = DEFAULT_CONFIG;
/** State of reading or writing. */

USART_STATE wifi_g_uc_state = STATE_EMPTY;

 sam_usart_opt_t usart_wifi_settings;

/**  Net states **/

CONFIG_STATE wifi_net_configuration = UNCONFIGURED;
NET_STATE wifi_net_connection = NET_UNCONNECTED;

/** FTP Server state **/

CONFIG_STATE ftp_configuration = UNCONFIGURED;

CONFIG_STATE sntp_configuration = UNCONFIGURED;

/** RN171 Power State **/
extern bool RN171_PS;



/**
 * \brief WIFI_UART_Handler
 *
 * \Interrupt handler for WIFI USART. Recive & Transmit buffer management.
 *
 * \param none
 *
 * \return none
 */

void WIFI_UART_Handler(void){

	ioport_set_pin_level(GREEN_LED_PIN, IOPORT_PIN_LEVEL_LOW);

	uint32_t wifi_ul_status;

	/* Read USART Status. */
	wifi_ul_status = usart_get_status(WIFI_UART);

	/* Receive buffer is full. */
	if ((wifi_ul_status & US_IER_USART_RXRDY)) {
		#if WIFI_MAX_BITS_TO_RECIVE< WIFI_RX_INDEX
		#error "El número de bits a recivir es mayor que el del buffer"
		#endif
		usart_read(WIFI_UART, WIFI_RX_BUFFER_POINTER+WIFI_RX_INDEX);

		if(WIFI_RX_INDEX==(WIFI_RX_BYTE_TO_TRANSFER-1)){
					RN171_StopTimeOut();
					 /* Disable all the interrupts. */
					usart_disable_interrupt(WIFI_UART, ALL_INTERRUPT_MASK);
					RN171_CheckMsg();
					wifi_g_uc_state = STATE_EMPTY;
		}else{
					wifi_g_uc_state = STATE_READ;
					WIFI_RX_INDEX++;
		}

	}

	/*TX buffer is ready */
	if ((wifi_ul_status & US_CSR_TXRDY)) {
```

```
		}
		ioport_set_pin_level(GREEN_LED_PIN, IOPORT_PIN_LEVEL_HIGH);
}



/**
 * \brief Init_RN171
 *
 * \note Initialize RN171 driver
 *
 * \param none
 *
 * \return none
 */
void RN171_Init (void){

		/* Configure ENABLE pin */

		ioport_set_pin_dir(WIFI_EN_PIN, IOPORT_DIR_OUTPUT);
		ioport_set_pin_mode(WIFI_EN_PIN, IOPORT_MODE_PULLUP);

		 /* Configure UART pins */

		ioport_set_pin_mode(PIN_PB10A_USART3_TXD,MUX_PB10A_USART3_TXD);
		ioport_disable_pin(PIN_PB10A_USART3_TXD);

		ioport_set_pin_mode(PIN_PB09A_USART3_RXD,MUX_PB09A_USART3_RXD);
		ioport_disable_pin(PIN_PB09A_USART3_RXD);
		/*
		if (Wifi_UART_Baudrate == WIFI_UART_HS_BAUDRATE)
		{
		ioport_set_pin_mode(PIN_PB06A_USART3_RTS,MUX_PB06A_USART3_RTS);
		ioport_disable_pin(PIN_PB06A_USART3_RTS);

		ioport_set_pin_mode(PIN_PB07A_USART3_CTS,MUX_PB07A_USART3_CTS);
		ioport_disable_pin(PIN_PB07A_USART3_CTS);
		}*/


		 /* Enable the peripheral clock in the PMC. */
		 sysclk_enable_peripheral_clock(WIFI_UART);

		if (Initial_UART_Config == ML_CONFIG)
		{

				/*Configure UART settings */

					usart_wifi_settings.baudrate= WIFI_UART_HS_BAUDRATE;
					usart_wifi_settings.char_length= WIFI_UART_CHAR_LENGTH;
					usart_wifi_settings.parity_type= WIFI_UART_PARITY;
					//usart_wifi_settings.parity_type= US_MR_PAR_ODD;
					usart_wifi_settings.stop_bits= WIFI_UART_STOP_BITS;
					usart_wifi_settings.channel_mode= US_MR_CHMODE_NORMAL;
					usart_wifi_settings.irda_filter=0;
			/*
			usart_init_hw_handshaking(WIFI_UART,
			&usart_wifi_settings, sysclk_get_peripheral_bus_hz(WIFI_UART));*/

			usart_init_rs232(WIFI_UART,
			&usart_wifi_settings, sysclk_get_peripheral_bus_hz(WIFI_UART));

		}
		if (Initial_UART_Config == DEFAULT_CONFIG)
		{
			/*Configure UART settings */

				usart_wifi_settings.baudrate= WIFI_UART_DEFAULT_BAUDRATE;
				usart_wifi_settings.char_length= WIFI_UART_CHAR_LENGTH;
				usart_wifi_settings.parity_type= WIFI_UART_PARITY;
				usart_wifi_settings.stop_bits= WIFI_UART_STOP_BITS;
				usart_wifi_settings.channel_mode= US_MR_CHMODE_NORMAL;
				usart_wifi_settings.irda_filter=0;

			usart_init_rs232(WIFI_UART,
			&usart_wifi_settings, sysclk_get_peripheral_bus_hz(WIFI_UART));
		}
```

```
		/* Disable all the interrupts. */
		usart_disable_interrupt(WIFI_UART, ALL_INTERRUPT_MASK);

		usart_enable_interrupt(WIFI_UART, US_IER_USART_RXRDY);

			/* Configure and enable interrupt of USART. */
		NVIC_EnableIRQ(WIFI_UART_IRQn);

		/* Enable the receiver and transmitter. */
		usart_enable_tx(WIFI_UART);
		usart_enable_rx(WIFI_UART);

		RN171_TimeOut_Init();  //CONFIGURE  TIMER

		Ready_to_reception (100, ANSWER_SPECIAL);

		usart_drive_RTS_pin_high(WIFI_UART);

		RN171_TIMEOUT=false;
			RN171_Power (ENABLE); //  Power_RN171 (ENABLE)
			Delay_to_ToR();

			RN171_StartTimeOut();

			while ((RN171_TIMEOUT==false) && (RN171_STATE != MODULE_RUN) && (!RN171_ERR)){

			}


		Delay_to_ToR (); //All header

}



/**
 * \brief Power_RN171
 *
 * \note: Turn on or turn off the power supply off RN171 Module.
 *
 * \param STATE: ENABLE or DISABLE
 *
 * \return none
 */
void RN171_Power (bool STATE){
  if(STATE == ENABLE){
  RN171_STATE=MODULE_POWERUP;
  ioport_set_pin_level(WIFI_EN_PIN, HIGH);
  }
  else{
		RN171_Reboot();
		RN171_STATE=MODULE_SHUTDOWN;
		NET_JOINED=false;
  ioport_set_pin_level(WIFI_EN_PIN, LOW);
  }
}


/**
 * \brief RN171_NetParameters_config
 *
 * \note:  Configure the access point for RN171 module
 *
 * \param
 *			- name : SSID of wireless access point
 *			- password: Password of wireless acces point
 *
 * \return none
 */
void RN171_NetParameters_config ( const char *name,  const char *password){

		wifi_net_configuration = UNCONFIGURED;

		if(RN171_STATE!= MODULE_CMD){
			RN171_CMD_MODE();
		 }
		if(RN171_STATE == MODULE_CMD){
```

```
	uint8_t nBytes_lenght;

		/*Sets the RNMODULE to automatically associate with the specified network upon boot-up*/

		usart_write_line(WIFI_UART, "set wlan ssid ");

		Delay_to_ToR ();
		nBytes_lenght= strlen(name)+16;
		Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

		usart_write_line(WIFI_UART, name);
		usart_write_line(WIFI_UART, "\r");
		Delay_to_ToR ();
		RN171_StartTimeOut();  //START TIMEOUT
		while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

			}

		if (RN171_ERR){
			wifi_net_configuration = CONFIGURE_FAIL;
			goto finish_net_config;
		}

		/*Provides the password to connect to the specified network*/
		usart_write_line(WIFI_UART, "set wlan pass ");

		Delay_to_ToR ();
		nBytes_lenght= strlen(password)+16;
		Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

		usart_write_line(WIFI_UART, password);
		usart_write_line(WIFI_UART, "\r");
		Delay_to_ToR();
		RN171_StartTimeOut();  //START TIMEOUT
		while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

			}

			if (RN171_ERR){
			wifi_net_configuration = CONFIGURE_FAIL;
			goto finish_net_config;
			}


		RN171_SaveConfig();

		if (RN171_ERR){
			wifi_net_configuration = CONFIGURE_FAIL;
			goto finish_net_config;
		}

		/*Force a reboot*/
		RN171_Reboot();
			finish_net_config:
		if(wifi_net_configuration != CONFIGURE_FAIL){
				wifi_net_configuration = CONFIGURED;
			}
			else{
				wifi_net_configuration = UNCONFIGURED;
		};
	}


}


/**
 * \brief RN171_Join_Wlan
 *
 * \note   Join Wlan configurated.
 *
 * \param  none
 *
 * \return none
 */
void RN171_Join_Wlan (void){
```

```
        if (wifi_net_configuration==CONFIGURED){

                if(RN171_STATE!= MODULE_CMD){
                        RN171_CMD_MODE();
                }
                if(RN171_STATE == MODULE_CMD){
                        //Delay_to_ToR ();
                        uint16_t XI;
                        Ready_to_reception(64, ANSWER_SPECIAL);

                        usart_write_line(WIFI_UART, "join\r");


                        for(XI=0; XI<100;XI++){
                        RN171_StartTimeOut();

                                while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                                }
                                usart_enable_interrupt(WIFI_UART, US_IER_USART_RXRDY);
                                if (RN171_ERR || RN171_CMD_OK){
                                                break;
                                        }
                        }
                        //Delay_to_ToR ();
                        Led_blink(GREEN_LED_PIN,ENABLE);
                        if ((!RN171_TIMEOUT) && (!RN171_ERR))
                        {

                                Ready_to_reception(90, ANSWER_SPECIAL);
                                for(XI=0; XI<0xFFFF;XI++){

                                        RN171_StartTimeOut();
                                        while ((!RN171_TIMEOUT) && (!RN171_ERR)){

                                        }
                                        usart_enable_interrupt(WIFI_UART, US_IER_USART_RXRDY);
                                        if (RN171_ERR||RN171_CMD_OK){
                                                NET_JOINED = true;
                                                usart_disable_interrupt(WIFI_UART, US_IDR_RXRDY);
                                                break;
                                        }

                                }

                        }

                        Led_reset(GREEN_LED_PIN);
                        Delay_to_ToR();
                }

        }
        if (wifi_net_configuration!=CONFIGURED){

        }
}

void RN171_GetMac (void){
        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }
        wait(1);
        if(RN171_STATE == MODULE_CMD){

                Ready_to_reception(46, ANSWER_MAC);

                usart_write_line(WIFI_UART, "get mac\r");
                RN171_StartTimeOut();

                while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                }

        }
}

void RN171_GetIp (void){
if (NET_JOINED){
```

```
		if(RN171_STATE!= MODULE_CMD){
			RN171_CMD_MODE();
		}
		if(RN171_STATE == MODULE_CMD){

			Ready_to_reception(90, ANSWER_IP);

			usart_write_line(WIFI_UART, "get ip\r");
			RN171_StartTimeOut();

			while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

			}

		}
}

}


/**
 * \brief RN171_FTP_config
 *
 * \note:  Configure the parameters of FTP Server
 *
 * \param
 *			- address : IP address of the FTP server
 *			- folder: The directory of the FTP server
 *			- user: user name
 *			- password: password
 *
 * \return none
 */
void RN171_FTP_Config (const char *address, const char *folder, const char *user, const char *password){

		if(RN171_STATE!= MODULE_CMD){
			RN171_CMD_MODE();
		}
		if(RN171_STATE == MODULE_CMD){

			uint8_t nBytes_lenght;

			/*Sets the FTP address*/
			usart_write_line(WIFI_UART, "set ftp address ");

			Delay_to_ToR ();
			nBytes_lenght= strlen(address)+16;
			Ready_to_reception(nBytes_lenght, ANSWER_AOK0);

			usart_write_line(WIFI_UART, address);
			usart_write_line(WIFI_UART, "\r");

			RN171_StartTimeOut();  //START TIMEOUT
			while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

				}

			if (RN171_ERR){

				ftp_configuration = CONFIGURE_FAIL;
				goto finish_ftp_config;
			}



			/*Sets the FTP address*/
			usart_write_line(WIFI_UART, "set ftp dir ");

			Delay_to_ToR ();
			nBytes_lenght= strlen(folder)+16;
			Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

			usart_write_line(WIFI_UART, folder);
			usart_write_line(WIFI_UART, "\r");

			RN171_StartTimeOut();  //START TIMEOUT
			while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

			}
```

```
                if (RN171_ERR){

                        ftp_configuration = CONFIGURE_FAIL;
                        goto finish_ftp_config;
                }

                /*Sets the FTP user*/

                usart_write_line(WIFI_UART, "set ftp user ");

                Delay_to_ToR ();
                nBytes_lenght= strlen(user)+16;
                Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

                usart_write_line(WIFI_UART, user);
                usart_write_line(WIFI_UART, "\r");

                RN171_StartTimeOut();  //START TIMEOUT
                while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

                }
                if (RN171_ERR){

                        ftp_configuration = CONFIGURE_FAIL;
                        goto finish_ftp_config;
                }

                /*Sets the FTP pass*/
                usart_write_line(WIFI_UART, "set ftp pass ");

                Delay_to_ToR ();
                nBytes_lenght= strlen(password)+16;
                Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

                usart_write_line(WIFI_UART, password);
                usart_write_line(WIFI_UART, "\r");

                RN171_StartTimeOut();  //START TIMEOUT
                while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

                }
                if (RN171_ERR == true){

                        ftp_configuration = CONFIGURE_FAIL;
                        goto finish_ftp_config;
                }


                RN171_SaveConfig();

                if (RN171_ERR){
                        ftp_configuration = CONFIGURE_FAIL;
                        goto finish_ftp_config;
                }

                RN171_Reboot();

        }

        finish_ftp_config:
        if(ftp_configuration != CONFIGURE_FAIL){
                ftp_configuration = CONFIGURED;
        }
        else{
                ftp_configuration = UNCONFIGURED;
        };

}

/**
 * \brief RN171_SNTP_config
 *
 * \note:  Configure the parameters of SNTP Server
 *
 * \param
 *                - address : IP address of the SNTP server
 *                - time_zone: UTC hour zone
 *
```

```
 * \return none
 */
void RN171_SNTP_Config (const char *address, int8_t time_zone){
        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }
        if(RN171_STATE == MODULE_CMD){
                uint8_t nBytes_lenght;
                uint8_t time_zone_dec,time_zone_uni;
                time_zone=time_zone+24;
                time_zone_dec=(time_zone/10);
                time_zone_uni=(time_zone-(time_zone_dec*10))+48;
                time_zone_dec=time_zone_dec+48;

                /*Sets the SNTP address */
                usart_write_line(WIFI_UART, "set time address ");
                Delay_to_ToR ();
                nBytes_lenght= strlen(address)+16;
                Ready_to_reception(nBytes_lenght,ANSWER_AOK0);

                usart_write_line(WIFI_UART, address);
                usart_write_line(WIFI_UART, "\r");

                RN171_StartTimeOut();  //START TIMEOUT
                while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

                }

                /*Sets the UTC Zone */
                usart_write_line(WIFI_UART, "set time zone ");

                Delay_to_ToR ();

                        if (time_zone_dec !='0')
                        {
                                usart_write_line(WIFI_UART, &time_zone_dec);
                        }

                usart_write_line(WIFI_UART, &time_zone_uni);
                Ready_to_reception(18,ANSWER_AOK0);
                usart_write_line(WIFI_UART, "\r");

                RN171_StartTimeOut();  //START TIMEOUT
                while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

                }

                RN171_SaveConfig();
                RN171_Reboot();

                sntp_configuration = CONFIGURED;
        }
}

/**
 * \brief RN171_SNTP_config
 *
 * \note:  Get time stap from SNTP server
 *
 * \param none
 *
 * \return none
 */
void RN171_GetTimeStamp (void){
        if (NET_JOINED){

                if(RN171_STATE!= MODULE_CMD){
                        RN171_CMD_MODE();
                }
                if(RN171_STATE == MODULE_CMD){

                        usart_write_line(WIFI_UART, "time");
                        Delay_to_ToR();
                        Ready_to_reception(16, ANSWER_AOK0);
                        usart_write_line(WIFI_UART, "\r");
                        RN171_StartTimeOut();
                        while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                        }
```

```
                        usart_write_line(WIFI_UART, "show t t");
                        Ready_to_reception(80, ANSWER_AOK0);
                        usart_write_line(WIFI_UART, "\r");
                        RN171_StartTimeOut();
                        while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                        }
                        Find_RTC_Value ();
                }
        }
}

/**
 * \brief RN171_FTP_NewFile
 *
 * \note:  Create a new archive into FTP server. This name must include the extension of archive.
 *
 * \param
 *                - file name: Name of file to be created in FTP server

 * \return none
 */
void RN171_FTP_NewFile (const char *file_name){

        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }
        if(RN171_STATE == MODULE_CMD){

                usart_write_line(WIFI_UART, "set ftp timer ");
                //Delay_to_ToR();
                usart_write_line(WIFI_UART, &FTP_TIMEOUT);
                Delay_to_ToR();
                Ready_to_reception(16, ANSWER_AOK0);
                usart_write_line(WIFI_UART, "\r");
                RN171_StartTimeOut();
                while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                }
                uint16_t XII;
                usart_write_line(WIFI_UART, "ftp put ");
                usart_write_line(WIFI_UART, file_name);

                Delay_to_ToR();
                Delay_to_ToR();
                Ready_to_reception(100,ANSWER_FILE_OPEN);
                usart_write_line(WIFI_UART, "\r");
                RN171_StartTimeOut();

                while(!(FTP_FILE_OPEN) && (!RN171_ERR) && (!RN171_CMD_OK) && (!RN171_TIMEOUT)){
                }

                Ready_to_reception(6,ANSWER_FILE_OPEN);
                for(XII=0; XII<10; XII++){
                        RN171_StartTimeOut();
                        while(!(FTP_FILE_OPEN) && (!RN171_ERR) && (!RN171_CMD_OK) && (!RN171_TIMEOUT)){
                        }
                        if (FTP_FILE_OPEN){
                                Ready_to_reception(6,ANSWER_FILE_CLOSE);
                                break;
                        }
                        if (RN171_ERR)
                        {
                                break;
                        }
                }

        }
}

/**
 * \brief RN171_FTP_FileWrite
 *
 * \note:  Write data into open file
 *
 * \param  file_data: Data to be written into file
 * \return none
```

```
 */
void RN171_FTP_FileWrite (char data){

        while (!usart_is_tx_empty(WIFI_UART));
        usart_putchar(WIFI_UART, data);
        //usart_write_line(WIFI_UART, file_data); //comprobar si esta es la funcion a usar
        //usart_write_line(WIFI_UART, string);

}


void RN171_FactorySettings (void){

        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }

        Delay_to_ToR();
        Delay_to_ToR();

        usart_write_line(WIFI_UART, "factory RESET\r");

        Delay_to_ToR();
        Delay_to_ToR();

        RN171_SaveConfig();

        Delay_to_ToR();
        Delay_to_ToR();

        RN171_Reboot();

}

void RN171_ChangeBaudRate (BAUD_RATE br_value){
        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }
        if(RN171_STATE == MODULE_CMD){
                if (br_value==BR115200)
                {
                        Wifi_UART_Baudrate = WIFI_UART_HS_BAUDRATE;
                        usart_write_line(WIFI_UART, "set uart baud 115200");
                        Ready_to_reception(16, ANSWER_AOK0);
                        usart_write_line(WIFI_UART, "\r");
                        RN171_StartTimeOut();
                        while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                        }


                }
                if (br_value==BR9600)
                {
                        Wifi_UART_Baudrate = WIFI_UART_DEFAULT_BAUDRATE;
                        usart_write_line(WIFI_UART, "set uart baud 9600");
                        Ready_to_reception(16, ANSWER_AOK0);
                        usart_write_line(WIFI_UART, "\r");
                        RN171_StartTimeOut();
                        while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

                        }

                }
                RN171_SaveConfig();
                RN171_Reboot();
        }
}

void RN171_ChangeParity (void){
        if(RN171_STATE!= MODULE_CMD){
                RN171_CMD_MODE();
        }
        if(RN171_STATE == MODULE_CMD){

        usart_write_line(WIFI_UART, "set uart flow 0x00");
        Ready_to_reception(16, ANSWER_AOK0);
        usart_write_line(WIFI_UART, "\r");
        RN171_StartTimeOut();
```

```
        while ((!RN171_TIMEOUT) && (!RN171_ERR) && (!RN171_CMD_OK)){

        }
                        RN171_SaveConfig();
                        RN171_Reboot();
        }
}

void Find_RTC_Value (void){
        uint8_t SI,SI2,SI3;
        uint8_t nStart, nStop, valuesize;

        for(SI=10;SI<80;SI++){
                if((WIFI_RX_BUFFER[SI-2]== 'R') && (WIFI_RX_BUFFER[SI-1]=='T') && (WIFI_RX_BUFFER[SI]=='C')
){
                        nStart=SI+2;
                        for(SI2=SI; SI2<SI+15 ; SI2++){
                                if (WIFI_RX_BUFFER[SI2]==13)
                                {
                                nStop=SI2;
                                break;
                                }
                        }
                break;
                }
        }
        if (nStop>nStart)
        {
    valuesize = nStop - nStart;

        for(SI3=9; SI3 >= (10-valuesize); SI3=SI3-1 ){
                nStop=nStop-1;
                RTC_VALUE[SI3] = WIFI_RX_BUFFER[nStop];

        }
        }

}


void RN171_SaveConfig (void){
                        /*Save the settings*/
                Delay_to_ToR();
                Ready_to_reception(33,ANSWER_SPECIAL);
                usart_write_line(WIFI_UART, "save");
                usart_write_line(WIFI_UART, "\r");
                RN171_StartTimeOut();
                /* Contesta con:Storing in config no con AOK*/
                while ((!RN171_TIMEOUT) && (!RN171_CMD_OK) && (!RN171_ERR)){

                        }

}

void RN171_Reboot (void){
                Delay_to_ToR();
                RN171_STATE = MODULE_REBOOT;
                Ready_to_reception(90, ANSWER_SPECIAL);
                usart_write_line(WIFI_UART, "reboot\r");
                RN171_StartTimeOut();
                 while ((RN171_TIMEOUT==false) && (RN171_STATE!= MODULE_RUN) &&(!RN171_ERR)){

                 }
                Delay_to_ToR();
}

/**
 * \brief Ready_to_reception
 *
 * \note  Set all parameters for receive command ack
 *                      This is only an internal function for this library.
 *                      Don´t call this function in others libraries.
 *
 * \param  none
 *
 * \return none
 */

void Ready_to_reception (uint8_t nBytes, ANSWER_TYPE ASW){
```

```
		COMMAND_ANSWER = ASW;
		uint8_t x;

		RN171_CMD_OK = false;
		RN171_ERR = false;

		for(x=0;x<=WIFI_RX_INDEX;x++){
			WIFI_RX_BUFFER[x] = 0;
		}
		WIFI_RX_BYTE_TO_TRANSFER=nBytes;
		WIFI_RX_INDEX=0;
		usart_enable_interrupt(WIFI_UART, US_IER_USART_RXRDY);
}

void RN171_CMD_MODE (void){

		usart_enable_tx(WIFI_UART);
		usart_enable_rx(WIFI_UART);


		Ready_to_reception(4,ANSWER_AOK0);

		usart_write_line(WIFI_UART,"$$$");

		Delay_to_ToR();
		RN171_StartTimeOut();  //START TIMEOUT
		while ((!RN171_TIMEOUT) && (RN171_STATE!= MODULE_CMD)){

		}

}

/**
 * \brief RN171_CheckMsg
 *
 * \note   Interpret the recive message trough WIFI USART
 *                          This is only an internal function for this library.
 *                          Don´t call this function in others libraries.
 *
 * \param  none
 *
 * \return none
 */

void RN171_CheckMsg (void) {

switch (RN171_STATE)
{
		case MODULE_POWERUP:
		/// POWERUP 9600
			if((WIFI_RX_BUFFER[1]=='w') && (WIFI_RX_BUFFER[2]=='i'))
			{
				usart_disable_rx(WIFI_UART);
				RN171_STATE=MODULE_RUN;
				RN171_CMD_OK = true;
				//RN171_READY = true;
			}
		/// POWERUP 115200
			if((WIFI_RX_BUFFER[0]=='w') && (WIFI_RX_BUFFER[1]=='i'))
			{
				usart_disable_rx(WIFI_UART);
				RN171_STATE=MODULE_RUN;
				RN171_CMD_OK = true;
				//RN171_READY = true;
			}
		break;

		case MODULE_REBOOT:
			if((WIFI_RX_BUFFER[18]=='w') && (WIFI_RX_BUFFER[19]=='i'))
			{
				usart_disable_rx(WIFI_UART);
				RN171_STATE = MODULE_RUN;
				RN171_CMD_OK = true;
				///RN171_READY = true;
			}
			if((WIFI_RX_BUFFER[19]=='w') && (WIFI_RX_BUFFER[20]=='i'))
			{
				usart_disable_rx(WIFI_UART);
```

```
                        RN171_STATE = MODULE_RUN;
                        RN171_CMD_OK = true;
                        ///RN171_READY = true;
                }

        break;

        case MODULE_RUN:

                if((WIFI_RX_BUFFER[0]=='C') && (WIFI_RX_BUFFER[1]=='M')){
                        RN171_CMD_OK=true;
                        RN171_STATE = MODULE_CMD;
                        ///RN171_CMD = true;
                }
                if((WIFI_RX_BUFFER[1]=='C') && (WIFI_RX_BUFFER[2]=='M')){
                        RN171_CMD_OK=true;
                        RN171_STATE = MODULE_CMD;
                        ///RN171_CMD = true;
                }
        break;

        case MODULE_CMD:
                switch (COMMAND_ANSWER)
                {
                        case ANSWER_AOK0:
                        if((WIFI_RX_BUFFER[WIFI_RX_INDEX-11]=='A') && (WIFI_RX_BUFFER[WIFI_RX_INDEX-
10]=='O')&& (WIFI_RX_BUFFER[WIFI_RX_INDEX-9]=='K')){
                                RN171_CMD_OK = true;
                        }
                        break;
                        case ANSWER_SPECIAL:
                        //Storing in config
                        if((WIFI_RX_BUFFER[7]=='S') && (WIFI_RX_BUFFER[8]=='t')){
                                RN171_CMD_OK = true;
                        }
                        ///Join wlan
                        if((WIFI_RX_BUFFER[14]=='A') && (WIFI_RX_BUFFER[15]=='u')){
                                RN171_CMD_OK = true;

                        }
                        if((WIFI_RX_BUFFER[15]=='A') && (WIFI_RX_BUFFER[16]=='u')){
                                RN171_CMD_OK = true;

                        }
                        break;
                        case ANSWER_MAC:
                                if((WIFI_RX_BUFFER[11]=='M') && (WIFI_RX_BUFFER[12]=='a')){
                                        RN171_CMD_OK = true;
                                        MAC_ADD_READED =true;
                                        uint8_t ii=20;
                                        uint8_t ib=0;
                                        for(ii=20;ii<37;ii++){
                                                if(WIFI_RX_BUFFER[ii]!=':'){
                                                        MAC_ADD[ib]=WIFI_RX_BUFFER[ii];
                                                        ib++;
                                                }

                                        }
                                }
                        break;
                        case ANSWER_IP:
                                if((WIFI_RX_BUFFER[13]=='U') && (WIFI_RX_BUFFER[14]=='P')){
                                        RN171_CMD_OK = true;
                                        uint8_t ii=0;
                                        for(ii=0;ii<12;ii++){

                                        IP_ADD[ii]=WIFI_RX_BUFFER[ii+29];

                                        }
                                }
                                break;
                        case ANSWER_FILE_OPEN:
                                if((WIFI_RX_BUFFER[13]=='P') && (WIFI_RX_BUFFER[15]=='c')){
                                        RN171_CMD_OK = true;
                                }
                                if((WIFI_RX_BUFFER[1]=='*') && (WIFI_RX_BUFFER[2]=='O') &&
(WIFI_RX_BUFFER[5]=='N')){
                                        FTP_FILE_OPEN= true;
                                        RN171_CMD_OK = true;
```

```
                                        Led_set(BLUE_LED_PIN);
                                }
                                if((WIFI_RX_BUFFER[0]=='*') && (WIFI_RX_BUFFER[1]=='O') &&
(WIFI_RX_BUFFER[4]=='N')){
                                        FTP_FILE_OPEN= true;
                                        RN171_CMD_OK = true;
                                        Led_set(BLUE_LED_PIN);
                                }
                                break;
                        case ANSWER_FILE_CLOSE:
                                        FTP_FILE_OPEN= false;
                                        RN171_CMD_OK = true;
                                        Led_reset(BLUE_LED_PIN);
                        break;
                        case ANSWER_TIME:
                                if((WIFI_RX_BUFFER[0]=='*') && (WIFI_RX_BUFFER[1]=='C') &&
(WIFI_RX_BUFFER[4]=='S')){
                                        RN171_CMD_OK = true;
                                }
                        break;
                } ///switch

        break;
        } ///switch
        if (!RN171_CMD_OK)
        {
                RN171_ERR=true;
        }
}
```

```
/**
 * \file
 *
 * \brief RN171 Driver.
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
 *
 *
 * \license_stop
 *
 */

#ifndef RN171_H
#define RN171_H

#include "compiler.h"

/// @cond 0
/**INDENT-OFF**/
#ifdef __cplusplus
extern "C" {
        #endif
        /**INDENT-ON**/
        /// @endcond


/** All interrupt mask. */
#define ALL_INTERRUPT_MASK  0xffffffff

/// PIN CONFIGURATIONS FOR RN171

#define WIFI_EN_PIN                     PIN_PA10

#define WIFI_UART_CTS_PIN               PIN_PB07A_USART3_CTS
#define WIFI_UART_RTS_PIN               PIN_PB06A_USART3_RTS
#define WIFI_UART_TXD_PIN               PIN_PB10A_USART3_TXD
#define WIFI_UART_RXD_PIN               PIN_PB09A_USART3_RXD
```

```
#define WIFI_UART_CTS_MUX                MUX_PB07A_USART3_CTS
#define WIFI_UART_RTS_MUX                MUX_PB06A_USART3_RTS
#define WIFI_UART_TXD_MUX                MUX_PB10A_USART3_TXD
#define WIFI_UART_RXD_MUX                MUX_PC11B_USART2_RXD

/** USART Interface */
#define WIFI_UART              USART3
#define WIFI_USART_USART_ID           ID_USART3
/** Baudrate setting */
#define WIFI_UART_DEFAULT_BAUDRATE     9600
#define WIFI_UART_HS_BAUDRATE     115200
/** Character length setting */
#define WIFI_UART_CHAR_LENGTH  US_MR_CHRL_8_BIT
/** Parity setting */
#define WIFI_UART_PARITY       US_MR_PAR_NO
/** Stop bits setting */
#define WIFI_UART_STOP_BITS    US_MR_NBSTOP_1_BIT

#define WIFI_UART_Handler              USART3_Handler

#define WIFI_UART_IRQn                 USART3_IRQn

typedef enum{
         MODULE_POWERUP=0,
         MODULE_RUN,
         MODULE_CMD,
         MODULE_FTP_RQT,
         MODULE_UPLOAD,
         MODULE_REBOOT,
         MODULE_SHUTDOWN,
}MODULE_STATE;

typedef enum {
         UNCONFIGURED=0,                                /* (0) */
         CONFIGURED,                                            /* (1) */
         CONFIGURE_FAIL,                                /* (2) */
} CONFIG_STATE;

typedef enum{
         DEFAULT_CONFIG=0,
         ML_CONFIG,
} UART_CONFIG;
typedef enum{
         NET_UNCONNECTED=0,
         NET_GETTING_IP,
         NET_CONNECTED,
         NET_CONNECTION_FAIL,
} NET_STATE;



typedef enum{
         STATE_READ=0,
         STATE_WRITE,
         STATE_EMPTY,
} USART_STATE;

typedef enum{
         BR115200=0,
         BR9600,
}BAUD_RATE;

typedef enum{
         ANSWER_AOK0=0,
         ANSWER_MAC,
         ANSWER_IP,
         ANSWER_TIME,
         ANSWER_FILE_OPEN,
         ANSWER_FILE_CLOSE,
         ANSWER_SPECIAL,
}ANSWER_TYPE;



/* Default net configuration parameters */
#define WLAN_default_ssid            "MULTISENSOR"
#define WLAN_default_pass            "TFMGABRIEL"
```

```
/* Default FTP Server configuration parameters */
#define FTP_default_address          "192.168.1.36"
#define FTP_default_folder           "TFM"

#define FTP_default_user             "gabriel"
#define FTP_default_pass             "gabriel"

#define FTP_TIMEOUT "010"

/*Default SNTP Server configuration parameters */
#define SNTP_default_address         "150.214.94.10"
#define SNTP_default_zone            0


/********************* Exported Functions ***********************/

void RN171_Init (void);
void RN171_Power (bool STATE);

void RN171_NetParameters_config ( const char *name,  const char *password);
void RN171_Join_Wlan (void);

void RN171_FTP_Config (const char *address, const char *folder, const char *user, const char *password);

void RN171_FTP_NewFile (const char *file_name);
void RN171_FTP_FileWrite (char data);

void RN171_FactorySettings (void);
void RN171_ChangeBaudRate (BAUD_RATE br_value);
void RN171_ChangeParity (void);

void RN171_GetMac (void);
void RN171_GetIp (void);

void RN171_SNTP_Config (const char *address, int8_t time_zone);
void RN171_GetTimeStamp (void);
#endif /* RN171_H_INCLUDED */

/*
 * WIFI_TIMERS.c
 *
 * Created: 05/09/2016 19:36:36
 *  Author: 108778
 */

#include "asf.h"
#include "compiler.h"
#include "WIFI_TIMERS.h"
#include "RN171.h"

extern bool RN171_TIMEOUT;
extern bool RN171_DELAY;


/**
* \brief Interrupt handler for TC00. Record the number of bytes received,
* and then restart a read transfer on the USART if the transfer was stopped.
*/
void TC00_Handler(void)
{
	uint32_t ul_status;

	/* Read TC0 Status. */
	ul_status = tc_get_status(TC0, 0);

	/* RC compare. */
	if ((ul_status & TC_SR_CPCS) == TC_SR_CPCS)
	{
		tc_stop(TC0,0);
		NVIC_DisableIRQ(TC00_IRQn);
		usart_disable_interrupt(WIFI_UART, US_IDR_RXRDY);

		RN171_TIMEOUT= true; //TIMEOUT
	}
}

void TC01_Handler (void){
	uint32_t ul_status2;
```

```
		/* Read TC0 Status. */
		ul_status2 = tc_get_status(TC0, 1);

		/* RC compare. */
		if ((ul_status2 & TC_SR_CPCS) == TC_SR_CPCS)
		{
			tc_stop(TC0,1);
			NVIC_DisableIRQ(TC01_IRQn);
			RN171_DELAY=true;
		}
}

/**
 * \brief RN171_TimeOut_Init
 *
 * \note   Interpret the recive message trough WIFI USART
 *                        This is only an internal function for this library.
 *                        Don´t call this function in others libraries.
 *
 * \param  none
 *
 * \return none
 */
/**
 * \brief Configure timer.
 */
void RN171_TimeOut_Init(void){


		/* Configure clock service. */
		sysclk_enable_peripheral_clock(TC0);


}



void RN171_StartTimeOut (void){

		RN171_TIMEOUT=false;

		/** Timer counter frequency in Hz. */
		#define TC_FREQ			0.1
		uint32_t ul_div;
		uint32_t ul_tcclks;
		static uint32_t ul_pbaclk;

		/* Get system clock. */
		ul_pbaclk = sysclk_get_cpu_hz(); //sysclk_get_peripheral_bus_hz(TC0);

		/* Configure TC for a 1Hz frequency and trigger on RC compare. */
		tc_find_mck_divisor(TC_FREQ, ul_pbaclk, &ul_div, &ul_tcclks, ul_pbaclk);
		tc_init(TC0, 0, ul_tcclks | TC_CMR_CPCTRG);
		tc_write_rc(TC0, 0,((ul_pbaclk / ul_div) / TC_FREQ));


		 /* Configure and enable interrupt on RC compare. */
		 tc_enable_interrupt(TC0, 0, TC_IER_CPCS);
		 tc_start(TC0,0); /// Start time-out

		// Configure TC interrupts
		NVIC_DisableIRQ(TC00_IRQn);
		NVIC_ClearPendingIRQ(TC00_IRQn);
		NVIC_SetPriority(TC00_IRQn, 4);
		NVIC_EnableIRQ(TC00_IRQn);


}

void RN171_StopTimeOut(void){
		tc_stop(TC0,0);
		NVIC_DisableIRQ(TC00_IRQn);

}
```

```
void Delay_to_ToR (void){
	RN171_DELAY=false;

	/** Timer counter frequency in Hz. */
	#define TC_FREQ			0.1
	uint32_t ul_div;
	uint32_t ul_tcclks;
	static uint32_t ul_pbaclk;

	/* Get system clock. */
	ul_pbaclk = sysclk_get_cpu_hz(); //sysclk_get_peripheral_bus_hz(TC0);

	/* Configure TC for a 1Hz frequency and trigger on RC compare. */
	tc_find_mck_divisor(TC_FREQ, ul_pbaclk, &ul_div, &ul_tcclks, ul_pbaclk);
	tc_init(TC0, 1, ul_tcclks | TC_CMR_CPCTRG);
	tc_write_rc(TC0, 1,((ul_pbaclk / ul_div) / TC_FREQ));


	/* Configure and enable interrupt on RC compare. */
	tc_enable_interrupt(TC0, 1, TC_IER_CPCS);
	tc_start(TC0,1); /// Start time-out

	// Configure TC interrupts
	NVIC_DisableIRQ(TC01_IRQn);
	NVIC_ClearPendingIRQ(TC01_IRQn);
	NVIC_SetPriority(TC01_IRQn, 4);
	NVIC_EnableIRQ(TC01_IRQn);

	while (!RN171_DELAY)
	{
	}
}

/*
 * WIFI_TIMERS.h
 *
 * Created: 05/09/2016 19:36:54
 *  Author: 108778
 */


#ifndef WIFI_TIMERS_H_
#define WIFI_TIMERS_H_
#include "compiler.h"

/* Use TC Peripheral 0. */
#define TC TC0
//#define TC_PERIPHERAL 0

/* Use TC2_Handler for TC capture interrupt. */
#define TC_Handler			TC00_Handler
#define TC_IRQn				TC00_IRQn

#define ID_TC_CAPTURE		ID_TC0

/******************** Exported Functions **********************/
void RN171_TimeOut_Init(void);
void RN171_StartTimeOut(void);
void RN171_StopTimeOut(void);
void Delay_to_ToR (void);


#endif /* WIFI_TIMERS_H_ */
```

3. SHT-31

```
/*
 * SHT31.c
 *
 * Created: 23/08/2016 20:57:55
 *  Author: 108778
 */
#include <asf.h>
#include "SHT31.h"
#include "ENERGY MGT/ML_PowerMgt.h"
#include "main.h"

extern bool SHT_Busy;

/** Private Functions **/
int8_t   SHT31_I2C_Read(uint8_t Address, uint8_t *I2C_Data, uint8_t BytesToRead);
void     SHT31_I2C_Write(uint8_t Address, uint8_t *I2C_Data, uint8_t BytesToWrite);
void    SHT31_ReadMeasurement      (uint8_t Address, SHT31_Measurement *Measurement);
void     SHT31_ReadFrame(uint8_t Address, uint16_t Command, uint8_t *Data, uint16_t BytesToRead);
void     SHT31_WriteCommand(uint8_t Address, uint16_t Command);


/** I2C RX BUFFER **/
uint8_t SHT31_I2C_ReadData[SHT31_I2C_READ_BUFFER];

/** I2C COMMAND BUFFER**/
uint8_t SHT_Cmd[2];

/** I2C TX & RX PACKETS  **/
twi_package_t packet_tx, packet_rx;

bool SHT_POWERED = false;
bool SHT_CONFIGURED = false;

/**
 * \brief Initialize SHT31 sensor.
 *
 * \return none
 */
void SHT31_Init (void){


          ioport_set_pin_mode(SHT31_SCL_PIN, SHT31_SCL_MUX);
          ioport_disable_pin(SHT31_SCL_PIN);

          ioport_set_pin_mode(SHT31_SDA_PIN, SHT31_SDA_MUX);
          ioport_disable_pin(SHT31_SDA_PIN);

          sysclk_enable_peripheral_clock(SYSCLK_TWIM3);

          /* Set TWIM options */

          struct twim_config opts = {

          .twim_clk = sysclk_get_cpu_hz(),
          .speed = TWIM_MASTER_SPEED,
          .hsmode_speed = TWIM_MASTER_SPEED,
          .data_setup_cycles = 0,
          .hsmode_data_setup_cycles = 0,
          .smbus = false,
          .clock_slew_limit = 0,
          .clock_drive_strength_low = 1,
          .data_slew_limit = 0,
          .data_drive_strength_low = 1,
          .hs_clock_slew_limit = 0,
          .hs_clock_drive_strength_high = 0,
          .hs_clock_drive_strength_low = 1,
          .hs_data_slew_limit = 0,
          .hs_data_drive_strength_low = 1,
          };


     twim_set_config(EXAMPLE_TWIM, &opts);
          /* Initialize the TWIM Module */
     twim_set_callback(EXAMPLE_TWIM, 0, twim_default_callback, 1);
          twim_enable(EXAMPLE_TWIM);
```

```
		//twim_enable_interrupt(EXAMPLE_TWIM,TWIM_IER_CRDY);

		SHT31_ReadFrame(TARGET_ADDRESS, STATUS_REGISTER, &SHT31_I2C_ReadData[0], 4);
		SHT31_WriteCommand(TARGET_ADDRESS, ONE_SHOT_HR);

		 SHT_CONFIGURED = true;
		 wait(1);

}

void SHT_Power (bool STATE){
		if(STATE == ENABLE){
			SHT_POWERED = true;
			ioport_set_pin_level(SHT31_POWER_EN, HIGH);
		}
		else{
			SHT_POWERED = false;
			ioport_set_pin_level(SHT31_POWER_EN, LOW);
		}
}


/**
 * \brief Write frame to sensor
 *
 * \note:
 *
 * \return none
 */
void SHT31_WriteCommand (uint8_t Address, uint16_t Command)
{
		//uint8_t SHT_Cmd[2];
		SHT_Cmd[0]=(uint8_t)(Command>>8);
		SHT_Cmd[1]=(uint8_t)(Command);

		SHT31_I2C_Write(Address, &SHT_Cmd[0], 2);
}

/**
 * \brief Read frame from the sensor
 *
 * \note:
 *
 * \return none
 */
void	SHT31_ReadFrame(uint8_t Address, uint16_t Command, uint8_t *Data, uint16_t BytesToRead)
{
		//uint8_t SHT_Cmd[2];
		SHT_Cmd[0]=(uint8_t)(Command>>8);
		SHT_Cmd[1]=(uint8_t)(Command);


		SHT31_I2C_Write(Address, &SHT_Cmd[0] , 2);

		SHT31_I2C_Read(Address, Data, BytesToRead);

}

/**
 * \brief Reset SHT31 sensor
 *
 * \note:
 *
 * \return none
 */
void  SHT31_Reset	(uint8_t Address)
{
		SHT31_WriteCommand(Address,SOFT_RESET);
		SHT31_Init();

}

/**
 * \brief Read last measurement.
 *
 * \note:
 *
 * \return none
```

```
 */
void SHT31_ReadMeasurement (uint8_t Address, SHT31_Measurement *Measurement)
{
	uint16_t Register;
	float Calcule;

	SHT31_ReadFrame(Address, 0xE000, &SHT31_I2C_ReadData[0], 6);
		SHT_Busy=false;
	SHT_Sutdown_Rqst ();
	/*********************************************/

	/*		TEMPERATURE			*/


	//Paso a entero
	Register=SHT31_I2C_ReadData[0];
	Register=(Register<<8)+SHT31_I2C_ReadData[1];
	//Calculo de la temperatura
	Calcule=Register;
	Calcule=Calcule/65535;
	Measurement->Temperature=-45+175*Calcule;
	/*********************************************/

	/*		HUMIDITY			*/
	Register=SHT31_I2C_ReadData[3];
	Register=(Register<<8)+SHT31_I2C_ReadData[4];
	Calcule=Register;
	Calcule=Calcule/65535;
	Measurement->Humidity=100*Calcule;
	/*********************************************/

}



/**
 * \brief Write the data pattern to the target.
 *
 *			This is only an internal function for this library.
 *			Don´t call this function in others libraries.
 *
 * \return none
 */
void SHT31_I2C_Write(uint8_t Address, uint8_t *I2C_Data, uint8_t BytesToWrite)
{
	/* TWI chip address to communicate with */
	packet_tx.chip = Address;
	/* TWI address/commands to issue to the other chip (node) */
	packet_tx.addr[0] = VIRTUALMEM_ADDR;
	/* Length of the TWI data address segment (1-3 bytes) */
	packet_tx.addr_length = TARGET_ADDR_LGT;
	/* Where to find the data to be written */
	packet_tx.buffer = I2C_Data;
	/* How many bytes do we want to write */
	packet_tx.length = BytesToWrite;
	/* Write data to TARGET */
	twi_master_write(EXAMPLE_TWIM, &packet_tx);
}


/**
 * \brief Read the data pattern to the target.
 *
 *			This is only an internal function for this library.
 *			Don´t call this function in others libraries.
 *
 * \return none
 */

int8_t	SHT31_I2C_Read (uint8_t Address, uint8_t *I2C_Data, uint8_t BytesToRead)
{
	/* TWI chip address to communicate with */
	packet_rx.chip = Address;
	/* Length of the TWI data address segment (1-3 bytes) */
	packet_rx.addr_length = TARGET_ADDR_LGT;
	/* How many bytes do we want to write */
	packet_rx.length = BytesToRead;
	/* TWI address/commands to issue to the other chip (node) */
	packet_rx.addr[0] = VIRTUALMEM_ADDR;
```

```
		/* Where to find the data to be written */
		packet_rx.buffer = I2C_Data;

		/* Read data from TARGET */
		return twi_master_read(EXAMPLE_TWIM, &packet_rx);

}


/**
 * \brief Get Temperature and Humidity data from SHT.
 *
 *
 * \return none
 */
void Get_Temp_Hum (SHT31_Measurement *Measurement){
		SHT_Busy=true;
		if (!SHT_POWERED)
		{
			SHT_Power(ENABLE);
		}
		if (!SHT_CONFIGURED)
		{
			SHT31_Init();
		}


		SHT31_ReadMeasurement (TARGET_ADDRESS, Measurement);

		SHT_Busy=false;


}

void Tem_Hum_ToSD (char *file_name){

			file_name[25]='R';
			file_name[26]='H';
			file_name[27]='T';
			file_name[28]=46;
			file_name[29]=116;
			file_name[30]=120;
			file_name[31]=116;

			SHT31_Measurement ambiente;

			Get_Temp_Hum (&ambiente);

			SD_PrintInFile (ambiente.Humidity);
			SD_PrintInFile ("\r");
			SD_PrintInFile (ambiente.Temperature);

			SD_OpenFile (file_name, OVERWRITE);

}



/**
 * \file
 *
 * \brief SHT31 Driver. Temperature & Humidity sensor
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
 *
 *
 * \license_stop
 *
 */
```

```
#ifndef SHT31_H_
#define SHT31_H_



#define SHT31_ADDRESS_DEVICE_1   0x44//<<1    //Ojo!! 0x44 es de 7 bit hay que rotar 1 bit
#define SHT31_ADDRESS_DEVICE_2   0x45//<<1    //Ojo!! 0x44 es de 7 bit hay que rotar 1 bit

/** TWIM Interrupt Number */
#define EXAMPLE_TWIM_IRQn    TWIM3_IRQn
/** TWIM Module Used */
#define EXAMPLE_TWIM         TWIM3
/** Target's TWI address */
#define TARGET_ADDRESS       SHT31_ADDRESS_DEVICE_1

/** Internal Address length */
#define TARGET_ADDR_LGT      0
/** Internal Address */
#define VIRTUALMEM_ADDR      0x0
/** Speed of TWI */
//#define TWIM_MASTER_SPEED    1
#define TWIM_MASTER_SPEED    TWI_FAST_MODE_SPEED
/** TWIM Interrupt Handler */
#define EXAMPLE_TWIM_Handler TWIM3_Handler



//COMANDOS
#define STATUS_REGISTER                         0xF32D
#define SOFT_RESET                              0x30A2
#define PERIODIC_1MPS_HR                0x2130        //1mps High Repeatibility
#define PERIODIC_10MPS_HR               0x2737        //10mps High Repeatibility
#define ONE_SHOT_HR                                 0x2C06                  //1 SHOT High
Repeatability
#define MEASUREMENT_READOUT             0xE000        //Lectura del Sensor

#define CMD_READ_SN                                     0x3780          //Serial Number Read Command


#define SHT31_COM_ERROR                         -1
#define SHT31_COM_ACK                           0

#define SHT31_I2C_READ_BUFFER           9


typedef struct {
        float      Temperature;
        float      Humidity;
} SHT31_Measurement;

#define SHT31_POWER_EN                  PIN_PB01

#define SHT31_SCL_MUX                   MUX_PB15C_TWIMS3_TWCK
#define SHT31_SCL_PIN                   PIN_PB15C_TWIMS3_TWCK

#define SHT31_SDA_MUX                   MUX_PB14C_TWIMS3_TWD
#define SHT31_SDA_PIN                   PIN_PB14C_TWIMS3_TWD



/******************** Exported Functions **********************/

void     SHT31_Init (void);
void     SHT_Power (bool STATE);

void     SHT31_Reset     (uint8_t Address);

void     Get_Temp_Hum (SHT31_Measurement *Measurement);

void    Tem_Hum_ToSD (char *file_name);


#endif /* SHT31_H_ */
```

4. MEMORIA SD

```
/*
 * ML_SDcard.c
 *
 * Created: 09/01/2016 0:36:03
 *  Author: 108778
 */
#include <asf.h>
#include "ML_SDcard.h"
#include "sd_mmc_spi.h"
#include "sd_mmc.h"
#include "ff.h"
#include <string.h>
#include "sd_mmc_mem.h"
#include "HMI/HMI.h"


sd_mmc_err_t err;
FRESULT res;
Ctrl_status status;
uint8_t slot = 0;
char test_file_name[] = "0:sd_mmc_test.txt";

static bool sd_mmc_ejected[2] = {false, false};
Ctrl_status Comprobar (uint8_t slot);

FATFS fs;
FIL file_object;

bool SD_INIT_ONGOING =false;
bool SD_DETECT = false;     /// CARD INSIDE
bool SD_ENABLE = false; /// POWER ENABLE
bool SD_READY = false;      /// CONFIGURED LOGICAL DRIVE


char SD_File_Buffer[10];

char* SD_File_Buffer_Pointer= &SD_File_Buffer;

/** SD Memory Power State **/
extern bool uSD_PS;
extern void wait (volatile uint32_t ul_ms);


struct eic_line_config eic_line_conf;

/**
 * \brief Set peripheral mode for one single IOPORT pin.
 * It will configure port mode and disable pin mode (but enable peripheral).
 * \param pin IOPORT pin to configure
 * \param mode Mode masks to configure for the specified pin (\ref ioport_modes)
 */
#define ioport_set_pin_peripheral_mode(pin, mode) \
        do {\
                ioport_set_pin_mode(pin, mode);\
                ioport_disable_pin(pin);\
        } while (0)


/**
 * \brief Interrupt handler for uSD switch interrupt.
 */
//! [set_sw_uSD_callback]
static void sw_uSD_callback(void)
{
        /* Check if EIC push button line interrupt line is pending. */
        if ((eic_line_interrupt_is_pending(EIC, SD_SW_EIC_LINE)) || SD_INIT_ONGOING) {

                eic_line_disable(EIC, SD_SW_EIC_LINE);
                eic_line_clear_interrupt(EIC, SD_SW_EIC_LINE);

                if(ioport_get_pin_level(SD_SW_PIN)){

                                #ifdef DEBUGMODE
                                SerialAux_Print ("SD MEMORY: CARD INSERTED\n");
```

```
                #endif
            SD_DETECT = true;

            /// Configure detection for card removed                ////////////////////////
            struct eic_line_config SD_eic_line_conf;
            SD_eic_line_conf.eic_mode = EIC_MODE_EDGE_TRIGGERED;
            SD_eic_line_conf.eic_edge = EIC_EDGE_FALLING_EDGE;
            SD_eic_line_conf.eic_filter = EIC_FILTER_ENABLED;
            SD_eic_line_conf.eic_async = EIC_ASYNCH_MODE;

            //! [configure_eic_mode]
            eic_line_set_config(EIC, SD_SW_EIC_LINE,      &SD_eic_line_conf);
            eic_line_enable(EIC, SD_SW_EIC_LINE);
            ////////////////////////////////////////////////////////////////////////


            SD_Power (ENABLE);
            wait(1);

            /* Initialize SD MMC stack */
            sd_mmc_init();

                #ifdef DEBUGMODE
                SerialAux_Print ("SD MEMORY: SD/MMC Stack Initialized\n");
                SerialAux_Print ("SD MEMORY: Wait until card be ready\n");
                #endif

            // Wait for a card and ready
            do {
                err = sd_mmc_check(LUN_ID_2);
                if ((SD_MMC_ERR_NO_CARD != err)&& (SD_MMC_INIT_ONGOING != err)&&
(SD_MMC_OK != err)) {

                    while (SD_MMC_ERR_NO_CARD != sd_mmc_check(LUN_ID_2)) {
                    }
                }
            } while (SD_MMC_OK != err);

            /* Wait card present and ready */
            do {
                status = Comprobar (0);
                if (CTRL_FAIL == status) {
                    while (CTRL_NO_PRESENT != sd_mmc_check(0)) {
                    }
                }
            } while (CTRL_GOOD != status);

                #ifdef DEBUGMODE
                SerialAux_Print ("SD MEMORY: Card is ready\n");
                SerialAux_Print ("SD MEMORY: Mounting logical drive\n");
                #endif

            memset(&fs, 0, sizeof(FATFS));
            res = f_mount(0, &fs);
            if (FR_INVALID_DRIVE == res) {
                #ifdef DEBUGMODE
                SerialAux_Print ("SD MEMORY: FAIL to mount logical drive\n");
                #endif
                Led_set(RED_LED_PIN);
            }
            SD_READY=true;
        }

        if(!ioport_get_pin_level(SD_SW_PIN)){
            SD_DETECT = false;
            SD_READY = false;
            SD_Power (DISABLE);

            /// Configure detection for card inserted               ////////////////////////
            struct eic_line_config SD_eic_line_conf;
            SD_eic_line_conf.eic_mode = EIC_MODE_EDGE_TRIGGERED;
            SD_eic_line_conf.eic_edge = EIC_EDGE_RISING_EDGE;
            SD_eic_line_conf.eic_filter = EIC_FILTER_ENABLED;
            SD_eic_line_conf.eic_async = EIC_ASYNCH_MODE;

            //! [configure_eic_mode]
            eic_line_set_config(EIC, SD_SW_EIC_LINE,      &SD_eic_line_conf);
            eic_line_enable(EIC, SD_SW_EIC_LINE);
            ////////////////////////////////////////////////////////////////////////
```

```
                }

        }
}
//! [set_sw_uSD_callback]


void SDcard_Init (void){

        SD_INIT_ONGOING =true;
        /** Configure SPI Hardware **/
        ioport_set_pin_mode(SPI_MISO_PIN,SPI_MISO_MUX);
        ioport_disable_pin(SPI_MISO_PIN);

        ioport_set_pin_mode(SPI_MOSI_PIN, SPI_MOSI_MUX);
        ioport_disable_pin(SPI_MOSI_PIN);

        ioport_set_pin_mode(SPI_SCK_PIN, SPI_SCK_MUX);
        ioport_disable_pin(SPI_SCK_PIN);

        ioport_set_pin_mode(SPI_NPCS0_PIN, SPI_NPCS0_MUX);
        ioport_disable_pin(SPI_NPCS0_PIN);

        /** Power uSD Memory **/
        ioport_set_pin_dir(SD_EN_PIN, IOPORT_DIR_OUTPUT);
        ioport_set_pin_mode(SD_EN_PIN, IOPORT_MODE_PULLUP);
        ioport_set_pin_level(SD_EN_PIN, IOPORT_PIN_LEVEL_LOW);

        ioport_set_pin_mode(SD_SW_EIC_PIN, SD_SW_EIC_PIN_MUX);
        ioport_disable_pin(SD_SW_EIC_PIN);

        eic_enable(EIC);

        struct eic_line_config SD_eic_line_conf;
        SD_eic_line_conf.eic_mode = EIC_MODE_EDGE_TRIGGERED;
        SD_eic_line_conf.eic_edge = EIC_EDGE_RISING_EDGE;
        SD_eic_line_conf.eic_level = EIC_LEVEL_HIGH_LEVEL;
        SD_eic_line_conf.eic_filter = EIC_FILTER_ENABLED;
        SD_eic_line_conf.eic_async = EIC_ASYNCH_MODE;

        //! [configure_eic_mode]
        eic_line_set_config(EIC, SD_SW_EIC_LINE,      &SD_eic_line_conf);
        //! [configure_eic_mode]

        //! [set_eic_callback_1]
        eic_line_set_callback(EIC, SD_SW_EIC_LINE, sw_uSD_callback,
        GPIO_SD_SW_EIC_IRQ, 1);
        //! [set_eic_callback_1]

        sw_uSD_callback();
        //! [enable_eic_line]
        //eic_line_enable(EIC, SD_SW_EIC_LINE);
        //! [enable_eic_line]
        SD_INIT_ONGOING =false;
}

/**
 * \brief SD_Power
 *
 * \note: Turn on or turn off the power supply off SD Memory.
 *
 * \param STATE: ENABLE or DISABLE
 *
 * \return none
 */
void SD_Power (bool STATE){
  if(STATE == ENABLE){
  ioport_set_pin_level(SD_EN_PIN, HIGH);
  SD_ENABLE = true;
  }
  else{
  ioport_set_pin_level(SD_EN_PIN, LOW);
  SD_ENABLE = false;
  }
}

void SD_Mount (void){
```

```
		memset(&fs, 0, sizeof(FATFS));
		res = f_mount(0, &fs);
		if (FR_INVALID_DRIVE == res) {
			Led_set(RED_LED_PIN);
		}

}

bool SD_OpenFile (char *file_name, FILE_OVW OverWrite){
		bool File_Opened=true;

		*file_name = 0 + '0';
		if (OverWrite == OVERWRITE)
		{
			res = f_open(&file_object,file_name, FA_CREATE_ALWAYS | FA_WRITE);
		}else{
			res = f_open(&file_object,file_name, /*FA_OPEN_ALWAYS|*/ FA_READ | FA_WRITE);
		}

		if (res != FR_OK) {
			File_Opened=false;
			//Led_set(RED_LED_PIN);
		}
		return File_Opened;
}

void SD_CloseFile (void){

		f_close(&file_object);

}

void SD_WriteInFile (uint16_t data_file){

		UINT nbytes= 2;
		UINT* nbytes_pointer= &nbytes;
		f_write (&file_object, &data_file,   nbytes, nbytes_pointer);


}

void SD_PrintInFile (char data_file)
{
		f_putc (data_file, &file_object);                                        /* Put a
character to the file */
}

void SD_PrintStringInFile (char* String){

		f_puts (String, &file_object);
}

void SD_ReadStringFromFile (char* PointerToSave, int nCharters){

		f_gets (PointerToSave, nCharters,  &file_object);                        /* Get a
string from the file */

}

uint8_t SD_ReadFromFile (void){

		UINT nbytes= 1;
		UINT* nbytes_pointer= &nbytes;
		FRESULT res;
		//uint8_t file_data[4];
		uint8_t Data_to_Return=0;
		while (res!= FR_OK){
			res = f_read (&file_object, &Data_to_Return, nbytes,&nbytes);
		}

		/*
		Data_to_Return=file_data[0]<<8;
		Data_to_Return=Data_to_Return|file_data[1];*/

		return Data_to_Return;
}
```

```
Ctrl_status Comprobar (uint8_t slot){

	switch (sd_mmc_check(slot))
	{
		case SD_MMC_OK:
		if (sd_mmc_ejected[slot]) {
			return CTRL_NO_PRESENT;
		}
		if (sd_mmc_get_type(slot) & (CARD_TYPE_SD | CARD_TYPE_MMC)) {
			return CTRL_GOOD;
		}
		// It is not a memory card
		return CTRL_NO_PRESENT;

		case SD_MMC_INIT_ONGOING:
		return CTRL_BUSY;

		case SD_MMC_ERR_NO_CARD:
		sd_mmc_ejected[slot] = false;
		return CTRL_NO_PRESENT;

		default:
		return CTRL_FAIL;
	}
}



/*
 * ML_SDcard.h
 *
 * Created: 09/01/2016 0:37:01
 *  Author: 108778
 */


#ifndef ML_SDCARD_H_
#define ML_SDCARD_H_


/** Enable SD MMC interface pins through SPI */
#define CONF_BOARD_SD_MMC_SPI



/** Defines required by SD MMC Stack */

#define SD_MMC_SPI_MEM_CNT		1


#define SD_MMC_0_CD_GPIO		(PIN_PA19)
#define SD_MMC_0_CD_DIR			(IOPORT_DIR_INPUT)
#define SD_MMC_0_CD_MODE		(IOPORT_MODE_PULLUP)
#define SD_MMC_0_CD_DETECT_VALUE	1			// High value with target inserted

#define SD_SW_PIN                                       PIN_PA19 /* Wrapper definition */
#define SD_SW_ACTIVE			true
#define SD_SW_INACTIVE			!USER_SW_ACTIVE
#define SD_SW_EIC_PIN			PIN_PA19C_EIC_EXTINT4
#define SD_SW_EIC_PIN_MUX		MUX_PA19C_EIC_EXTINT4
#define SD_SW_EIC_LINE			4
#define GPIO_SD_SW_EIC_IRQ                      EIC_4_IRQn



#define SD_MMC_SPI			SPI
#define SD_MMC_SPI_0_CS			0

#define SPI_NPCS0_PIN			(PIN_PC03A_SPI_NPCS0)
#define SPI_NPCS0_MUX			(MUX_PC03A_SPI_NPCS0)

#define SPI_MISO_PIN			(PIN_PC04A_SPI_MISO)
#define SPI_MISO_MUX			(MUX_PC04A_SPI_MISO)

#define SPI_MOSI_PIN			(PIN_PC05A_SPI_MOSI)
#define SPI_MOSI_MUX			(MUX_PC05A_SPI_MOSI)
```

```
#define SPI_SCK_PIN               (PIN_PC06A_SPI_SCK)
#define SPI_SCK_MUX               (MUX_PC06A_SPI_SCK)


#define SD_EN_PIN                 PIN_PC01

//#define SD_MMC_SPI_ENABLE          /// Configure FATfs access

#define SD_MMC_ENABLE
#define ACCESS_MEM_TO_RAM_ENABLED

typedef enum {
         NO_OVERWRITE=0,
         OVERWRITE
} FILE_OVW;


//#define SD_SPI_MAX_FREQ_CLOCK                        10000000


/*Exported functions */

void SDcard_Init(void);
void SD_Power (bool STATE);

// Open / Close files
bool SD_OpenFile (char *file_name, FILE_OVW OverWrite);
void SD_CloseFile (void);


// Write/Read data
void SD_WriteInFile (uint16_t data_file);
uint8_t SD_ReadFromFile (void);


// Write/Read CHAR
void SD_PrintInFile (char data_file);
void SD_PrintStringInFile (char* String);
void SD_ReadStringFromFile (char* PointerToSave, int nCharters);


#endif /* ML_SDCARD_H_ */
```

5. PIR

```
/**
 * \file
 *
 * \brief PIR Driver
 *
 * Proyect: MULTISENSOR WIRELESS - TFM UNIZAR
 *
 * Copyright (c) 2016 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */

#include "asf.h"
#include "compiler.h"
#include "PIR.h"
#include "main.h"
#include "HMI/HMI.h"

extern bool PIR_EVENT;

/**
 * \brief Interrupt handler for EIC interrupt.
 */
//! [set_eic_callback]
static void PIR_callback(void)
{

	/* Check if EIC push button line interrupt line is pending. */
	if (eic_line_interrupt_is_pending(EIC, PIR_EIC_LINE)) {
		eic_line_clear_interrupt(EIC, PIR_EIC_LINE);
		/*
		Led_set(GREEN_LED_PIN);
		//PIR_EVENT=true;
		wait(10);
		Led_reset(GREEN_LED_PIN);
		wait(10);*/

	}

}
//! [set_eic_callback]

/**
 * \brief Init_PIR
 *
 * \note
 *
 * \param none
 *
 * \return none
 */
void Init_PIR (void){

  /* Initialize USER SW */
  ioport_set_pin_dir(PIR_PIN, IOPORT_DIR_INPUT);
  ioport_set_pin_mode(PIR_PIN, IOPORT_MODE_PULLDOWN);

  ioport_set_pin_mode(PIR_EIC_PIN, IOPORT_MODE_PULLDOWN|PIR_EIC_PIN_MUX);
  ioport_disable_pin(PIR_EIC_PIN);


  eic_enable(EIC);

  struct eic_line_config eic_line_conf;
  eic_line_conf.eic_mode = EIC_MODE_EDGE_TRIGGERED;
  eic_line_conf.eic_edge = EIC_EDGE_RISING_EDGE;
  eic_line_conf.eic_level = EIC_LEVEL_HIGH_LEVEL;
  eic_line_conf.eic_filter = EIC_FILTER_DISABLED;
  eic_line_conf.eic_async = EIC_ASYNCH_MODE;
```

```
  //! [configure_eic_mode]
  eic_line_set_config(EIC, PIR_EIC_LINE,      &eic_line_conf);
  //! [configure_eic_mode]

  //! [set_eic_callback_1]
  eic_line_set_callback(EIC, PIR_EIC_LINE, PIR_callback,
  PIR_EIC_IRQ, 6);
  //! [set_eic_callback_1]

  Power_PIR (ENABLE);

  eic_line_clear_interrupt(EIC, PIR_EIC_LINE);
  //! [enable_eic_line]
  eic_line_enable(EIC, PIR_EIC_LINE);
  //! [enable_eic_line]

}


/**
 * \brief Power_PIR
 *
 * \note: Turn on or turn off the power supply off Passive Infrared Sensor
 *
 * \param STATE: ENABLE or DISABLE
 *
 * \return none
 */
void Power_PIR (bool STATE){

           //! [disable_eic_line]
           eic_line_disable(EIC, PIR_EIC_LINE);
           //! [disable_eic_line]

         if(STATE == ENABLE){
                                  //ENABLE PIR       POWER
                 ioport_set_pin_dir(PIR_EN_PIN, IOPORT_DIR_OUTPUT);
                 ioport_set_pin_mode(PIR_EN_PIN, IOPORT_MODE_DRIVE_STRENGTH);
                 ioport_set_pin_level(PIR_EN_PIN, IOPORT_PIN_LEVEL_HIGH);
         }
         else{

                 ioport_set_pin_dir(PIR_EN_PIN, IOPORT_DIR_INPUT);
                 ioport_set_pin_level(PIR_EN_PIN, IOPORT_PIN_LEVEL_LOW);
         }

           //! [enable_eic_line]
           eic_line_enable(EIC, PIR_EIC_LINE);
           //! [enable_eic_line]
}

void Detection (bool STATE){

         if(STATE== ENABLE){
                 eic_line_clear_interrupt(EIC, PIR_EIC_LINE);
                 eic_line_enable(EIC, PIR_EIC_LINE);
         }
         if(STATE== DISABLE){
                 eic_line_disable(EIC, PIR_EIC_LINE);
         }
}



/**
 * \file
 *
 * \brief PIR Driver.
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
 *
 *
```

```
 * \license_stop
 *
 */
#ifndef PIR_H
#define PIR_H


/// PIR POWER ENABLE
#define PIR_PIN                         PIN_PA17
#define PIR_POWER_EN_PIN                PIN_PA00

//! \name PIR definitions
//@{
#define GPIO_PIR_PIN                            PIN_PA17 /* Wrapper definition */
#define PIR_PIN                 PIN_PA17
#define PIR_ACTIVE              true
#define PIR_INACTIVE            !USER_SW_ACTIVE
#define PIR_EIC_PIN             PIN_PA17C_EIC_EXTINT2
#define PIR_EIC_PIN_MUX         MUX_PA17C_EIC_EXTINT2
#define PIR_EIC_LINE            2
#define PIR_EIC_IRQ                                     EIC_2_IRQn


#define PIR_EN_PIN                          PIN_PA00


/******************** Exported Functions **********************/

void Init_PIR (void);

void Power_PIR (bool STATE);

void Detection (bool STATE);

#endif /* PIR_H_INCLUDED */
```

6. INTERFACE USUARIO

```
/*
 * HMI.c
 *
 * Created: 11/08/2016 21:12:38
 *  Author: 108778
 */

#include <asf.h>
#include "UserButton.h"
#include "BoardLed.h"
#include "HMI.h"
#include "CAMERA/ML_Camera.h"
#include "main.h"


extern CAPTURE_MODE ML_MODE;

static bool GreenLed_State=false,RedLed_State=false,BlueLed_State=false;

static bool GreenLed_Blink=false,RedLed_Blink=false,BlueLed_Blink=false;


/**
 * \brief Interrupt handler for EIC interrupt.
 */
//! [set_eic_callback]
static void sw_eic_callback(void)
{

	/* Check if EIC push button line interrupt line is pending. */
	if (eic_line_interrupt_is_pending(EIC, USER_SW_EIC_LINE)) {
		eic_line_clear_interrupt(EIC, USER_SW_EIC_LINE);

		switch(ML_MODE){
			case MODE_SINGLE:
				ML_MODE=MODE_CONTINUOUS;
			break;

			case MODE_CONTINUOUS:
				ML_MODE=MODE_SINGLE;
			break;
		}


	}

}
//! [set_eic_callback]


void Init_HMI (void){

/* Initialize USER SW */
ioport_set_pin_dir(USER_SW_PIN, IOPORT_DIR_INPUT);
ioport_set_pin_mode(USER_SW_PIN, IOPORT_MODE_PULLDOWN);

ioport_set_pin_mode(USER_SW_EIC_PIN, IOPORT_MODE_PULLDOWN|USER_SW_EIC_PIN_MUX);
ioport_disable_pin(USER_SW_EIC_PIN);


eic_enable(EIC);

struct eic_line_config eic_line_conf;
eic_line_conf.eic_mode = EIC_MODE_EDGE_TRIGGERED;
eic_line_conf.eic_edge = EIC_EDGE_RISING_EDGE;
eic_line_conf.eic_level = EIC_LEVEL_HIGH_LEVEL;
eic_line_conf.eic_filter = EIC_FILTER_DISABLED;
eic_line_conf.eic_async = EIC_ASYNCH_MODE;

//! [configure_eic_mode]
eic_line_set_config(EIC, USER_SW_EIC_LINE,    &eic_line_conf);
//! [configure_eic_mode]

//! [set_eic_callback_1]
eic_line_set_callback(EIC, USER_SW_EIC_LINE, sw_eic_callback,
```

```
GPIO_USER_SW_EIC_IRQ, 6);
//! [set_eic_callback_1]

//! [enable_eic_line]
eic_line_enable(EIC, USER_SW_EIC_LINE);
//! [enable_eic_line]


/* Configure the pins connected to LEDs as output and set their
          * default initial state to high (LEDs off).
          */

         ioport_set_pin_dir(USER_LED1_PIN, IOPORT_DIR_OUTPUT);
                  ioport_set_pin_dir(USER_LED2_PIN, IOPORT_DIR_OUTPUT);
                  ioport_set_pin_dir(USER_LED3_PIN, IOPORT_DIR_OUTPUT);

                   ioport_set_pin_mode(USER_LED1_PIN, IOPORT_MODE_PULLUP | IOPORT_MODE_DRIVE_STRENGTH);
                   ioport_set_pin_mode(USER_LED2_PIN, IOPORT_MODE_PULLUP | IOPORT_MODE_DRIVE_STRENGTH);
                   ioport_set_pin_mode(USER_LED3_PIN, IOPORT_MODE_PULLUP | IOPORT_MODE_DRIVE_STRENGTH);


                  ioport_set_pin_level(USER_LED1_PIN, IOPORT_PIN_LEVEL_HIGH);
                  ioport_set_pin_level(USER_LED2_PIN, IOPORT_PIN_LEVEL_HIGH);
                  ioport_set_pin_level(USER_LED3_PIN, IOPORT_PIN_LEVEL_HIGH);

}

void Led_set(ioport_pin_t pin){

         ioport_set_pin_level(pin, LOW);

         switch (pin){

                  case USER_LED3_PIN:
                           GreenLed_State = ENABLE;
                           break;
                  case USER_LED2_PIN:
                           RedLed_State = ENABLE;
                           break;
                  case USER_LED1_PIN:
                           BlueLed_State = ENABLE;
                           break;
         }

}


void Led_reset(ioport_pin_t pin){

         ioport_set_pin_level(pin, HIGH);

         switch (pin){

                  case USER_LED3_PIN:
                  GreenLed_State = DISABLE;
                  GreenLed_Blink = DISABLE;
                  break;
                  case USER_LED2_PIN:
                  RedLed_State = DISABLE;
                  RedLed_Blink = DISABLE;
                  break;
                  case USER_LED1_PIN:
                  BlueLed_State = DISABLE;
                  BlueLed_Blink = DISABLE;
                  break;
         }

}

void Led_toggle(ioport_pin_t pin){

         ioport_toggle_pin_level(pin);

}

void Led_blink (ioport_pin_t pin, bool STATE){
         switch (pin){

                  case GREEN_LED_PIN:
```

```
                    GreenLed_Blink = STATE;
                    GreenLed_State = ENABLE;
                    break;
                    case RED_LED_PIN:
                    RedLed_Blink = STATE;
                    RedLed_State = ENABLE;
                    break;
                    case BLUE_LED_PIN:
                    BlueLed_Blink = STATE;
                    BlueLed_State = ENABLE;
                    break;
            }

}


void Blink_Leds (void){
            if (GreenLed_Blink && GreenLed_State)
            {
                    Led_toggle(USER_LED3_PIN);
            }
            if (RedLed_Blink && RedLed_State)
            {
                    Led_toggle(USER_LED2_PIN);
            }
            if (BlueLed_Blink && BlueLed_State)
            {
                    Led_toggle(USER_LED1_PIN);
            }
}

bool Read_UserButton_State (void){

}


void Set_UserButton_Callback (eic_callback_t user_callback, uint8_t Callback_Priority){

            //! [disable_eic_line]
            eic_line_enable(EIC, USER_SW_EIC_LINE);
            //! [enable_eic_line]

            //! [disable_callback_1]
            eic_line_set_callback(EIC, USER_SW_EIC_LINE, user_callback,
            GPIO_USER_SW_EIC_IRQ, 1);
            //! [set_eic_callback_1]

            //! [enable_eic_line]
            eic_line_enable(EIC, USER_SW_EIC_LINE);
            //! [enable_eic_line]
}



/*
 * HMI.h
 *
 *
 *  Author: 108778
 */


#ifndef HMI_H_
#define HMI_H_

#include "BoardLed.h"
#include "UserButton.h"

/******************** Exported Functions **********************/
void Init_HMI (void);

void Led_set (ioport_pin_t pin);
void Led_reset (ioport_pin_t pin);
void Led_toggle (ioport_pin_t pin);
void Led_blink (ioport_pin_t pin, bool STATE);

void Blink_Leds (void);
```

```
bool Read_UserButton_State (void);
void Set_UserButton_Callback (eic_callback_t user_callback, uint8_t Callback_Priority);


#endif /* HMI_H_ */


/*
 * BoardLed.h
 *
 * Created: 11/08/2016 21:13:17
 *  Author: 108778
 */


#ifndef BOARDLED_H_
#define BOARDLED_H_


#define USER_LED1_PIN                   PIN_PC25
#define USER_LED2_PIN                   PIN_PC26
#define USER_LED3_PIN                   PIN_PC27

#define BLUE_LED_PIN                                                    USER_LED1_PIN
#define RED_LED_PIN                                                             USER_LED2_PIN
#define GREEN_LED_PIN                                                   USER_LED3_PIN


#endif /* BOARDLED_H_ */



/*
 * UserButton.h
 *
 * Created: 11/08/2016 21:07:08
 *  Author: 108778
 */


#ifndef USERBUTTON_H_
#define USERBUTTON_H_

//! \name USER BUTTON definitions
//@{

#define GPIO_USER_SW_PIN                        PIN_PC24 /* Wrapper definition */
#define USER_SW_PIN                     PIN_PC24
#define USER_SW_ACTIVE                  true
#define USER_SW_INACTIVE                !USER_SW_ACTIVE
#define USER_SW_EIC_PIN                 PIN_PC24B_EIC_EXTINT1
#define USER_SW_EIC_PIN_MUX             MUX_PC24B_EIC_EXTINT1
#define USER_SW_EIC_LINE                1
#define GPIO_USER_SW_EIC_IRQ                    EIC_1_IRQn



#endif /* USERBUTTON_H_ */
```

7. GESTION ENERGÉTICA

```
/**
 * \file
 *
 * \brief Multisensor Wireless Power Management
 *
 * Copyright (c) 2015 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */
#include <asf.h>
#include "compiler.h"
#include "CAMERA/LinkSprite.h"
#include "AUDIO/ML_uPhone.h"
#include "WIFI/RN171.h"
#include "SHT/SHT31.h"

#include "ML_PowerMgt.h"


 /********************* Private functions *********************/

void adcife_read_vbat_result(void);
void adcife_read_vin_result(void);

extern uint16_t Vbat;
extern uint16_t Vin;
extern uint16_t register_value;

 extern bool Camera_Busy;

 /********************* Exported Power Status Variables *********************/

 bool Camera_Sht_Rqst=false;
 bool Camera_Busy=false;

 bool Wifi_Sht_Rqst = false;
 bool Wifi_Busy=false;

 bool uPhone_Sht_Rqst = false;
 bool uPhone_Busy=false;

 bool SHT_Sht_Rqst = false;
 bool SHT_Busy = false;

// Vin monitor configuration of the Sequencer
static struct adc_seq_config vin_adc_seq_cfg;
// Vbat monitor configuration of the Sequencer
static struct adc_seq_config vbat_adc_seq_cfg;

// Vmon ADC Configuration structure
static struct adc_config vmon_adc_cfg;

/** Vmon ADC instance */
struct adc_dev_inst vmon_adc_inst;



/**
 * \brief PowerMgt_Init
 *
 * \note Initializes Power Management Hardware
 *
 * \param none
 *
 * \return none
 */
void PowerMgt_Init (void){
```

```
// CONFIGURE VOLTAGE MONITORS PINS

ioport_set_pin_mode(VIN_MON_PIN,VIN_MON_MUX);
ioport_disable_pin(VIN_MON_PIN);

ioport_set_pin_mode(VBAT_MON_PIN,VBAT_MON_PIN);
ioport_disable_pin(VBAT_MON_PIN);


// configure parameters of Vmon ADCs configuration structs

// Vmon ADC Configuration structure
         /* System clock division factor is 4 */
         vmon_adc_cfg.prescal = ADC_PRESCAL_DIV4;
         /* The APB clock is used */
         vmon_adc_cfg.clksel = ADC_CLKSEL_APBCLK;
         /* Max speed is 150K */
         vmon_adc_cfg.speed = ADC_SPEED_75K;
         /* ADC Reference voltage is Vref */
         vmon_adc_cfg.refsel = ADC_REFSEL_2;
         /* Enables the Startup time */
         vmon_adc_cfg.start_up = CONFIG_ADC_STARTUP;

}

/**
 * \brief Capture_Vin_Value
 *
 * \note Start ADC conversion of Vin voltage
 *
 * \param none
 *
 * \return none
 */
void Get_Vin_Value (void){

// Vin monitor configuration of the Sequencer

         /* Select Vref for shift cycle */
         vin_adc_seq_cfg.zoomrange = ADC_ZOOMRANGE_0;
         /* Pad Ground */
         vin_adc_seq_cfg.muxneg = ADC_MUXNEG_1;
         /* DAC Internal */
         vin_adc_seq_cfg.muxpos = ADC_MUXPOS_3;
         /* Enables the internal voltage sources */
         vin_adc_seq_cfg.internal = ADC_INTERNAL_3;
         /* Disables the ADC gain error reduction */
         vin_adc_seq_cfg.gcomp = ADC_GCOMP_DIS;
         /* Disables the HWLA mode */
         vin_adc_seq_cfg.hwla = ADC_HWLA_DIS;
         /* 12-bits resolution */
         vin_adc_seq_cfg.res = ADC_RES_12_BIT;
         /* Enables the single-ended mode */
         vin_adc_seq_cfg.bipolar = ADC_BIPOLAR_SINGLEENDED;

         vin_adc_seq_cfg.gain = ADC_GAIN_1X;

         sysclk_enable_peripheral_clock(SYSCLK_ADCIFE);

         adc_init(&vmon_adc_inst, ADCIFE, &vin_adc_seq_cfg);
         adc_enable(&vmon_adc_inst);

         struct adc_ch_config vmon_adc_ch_cfg = {
                  .seq_cfg = &vin_adc_seq_cfg,
                  /* Window monitor mode is off */
                  .window_mode = ADC_WM_OFF,
                                    };

         adc_ch_set_config(&vmon_adc_inst, &vmon_adc_ch_cfg);

         adc_configure_trigger(&vmon_adc_inst, ADC_TRIG_SW );
         adc_configure_gain(&vmon_adc_inst, ADC_GAIN_1X);


         // Set interruption
          adc_set_callback(&vmon_adc_inst, ADC_SEQ_SEOC, adcife_read_vin_result,
          ADCIFE_IRQn, 1);
          adc_get_last_conv_value(&vmon_adc_inst);
         adc_start_software_conversion(&vmon_adc_inst);
```

```
        }


/**
 * \brief Capture_Vbat_Value
 *
 * \note Start ADC conversion of Vbat voltage
 *
 * \param none
 *
 * \return none
 */
void Get_Vbat_Value (void){

        // Vbat monitor configuration of the Sequencer

        /* Select Vref for shift cycle */
        vbat_adc_seq_cfg.zoomrange = ADC_ZOOMRANGE_0;
        /* Pad Ground */
        vbat_adc_seq_cfg.muxneg = ADC_MUXNEG_1;
        /* DAC Internal */
        vbat_adc_seq_cfg.muxpos = ADC_MUXPOS_4;
        /* Enables the internal voltage sources */
        vbat_adc_seq_cfg.internal = ADC_INTERNAL_3;
        /* Disables the ADC gain error reduction */
        vbat_adc_seq_cfg.gcomp = ADC_GCOMP_DIS;
        /* Disables the HWLA mode */
        vbat_adc_seq_cfg.hwla = ADC_HWLA_DIS;
        /* 12-bits resolution */
        vbat_adc_seq_cfg.res = ADC_RES_12_BIT;
        /* Enables the single-ended mode */
        vbat_adc_seq_cfg.bipolar = ADC_BIPOLAR_SINGLEENDED;

        vbat_adc_seq_cfg.gain = ADC_GAIN_1X;

        sysclk_enable_peripheral_clock(SYSCLK_ADCIFE);

        adc_init(&vmon_adc_inst, ADCIFE, &vbat_adc_seq_cfg);
        adc_enable(&vmon_adc_inst);

        struct adc_ch_config vmon_adc_ch_cfg = {
                .seq_cfg = &vbat_adc_seq_cfg,
                /* Window monitor mode is off */
                .window_mode = ADC_WM_OFF,
                                };

        adc_ch_set_config(&vmon_adc_inst, &vmon_adc_ch_cfg);
        adc_configure_trigger(&vmon_adc_inst, ADC_TRIG_SW );
        adc_configure_gain(&vmon_adc_inst, ADC_GAIN_1X);

                // Set interruption
                adc_set_callback(&vmon_adc_inst, ADC_SEQ_SEOC, adcife_read_vbat_result,
                ADCIFE_IRQn, 1);



    adc_get_last_conv_value(&vmon_adc_inst);
        adc_start_software_conversion(&vmon_adc_inst);
        }

/**
 * \brief adcife_read_vin_result
 *
 * \note Callback of Vin voltage conversion. This function refresh Vin variable value
 *
 * \param none
 *
 * \return none
 */
void adcife_read_vin_result(void)
            {
                    // Check the ADC conversion status
                    Vin = adc_get_last_conv_value(&vmon_adc_inst);
                    adc_clear_status(&vmon_adc_inst, ADCIFE_SCR_SEOC);
                    if(Vbat==0x0fff){
                            Vbat=0;
                            adc_disable(&vmon_adc_inst);
                            Get_Vbat_Value ();
```

```
                                }else{
                                Vbat=(Vbat*3.4652);
                        }

                }

/**
 * \brief adcife_read_bat_result
 *
 * \note Callback of Vbat voltage conversion. This function refresh Vbat variable value
 *
 * \param none
 *
 * \return none
 */
void adcife_read_vbat_result(void)
                {
                // Check the ADC conversion status
                        Vbat = adc_get_last_conv_value(&vmon_adc_inst);
                        adc_clear_status(&vmon_adc_inst, ADCIFE_SCR_SEOC);
                        if(Vbat==0x0fff){
                                Vbat=0;
                                 adc_disable(&vmon_adc_inst);
                                 Get_Vbat_Value ();
                        }else{
                                 register_value=Vbat;
                                 Vbat=(Vbat*2.01465); /// Vref=3.3v
                        }
                }

/**
 * \brief Check_Shutdown_Rqst
 *
 * \note Check if any peripheral can be powerdown
 *
 * \param none
 *
 * \return none
 */
void Check_Shutdown_Rqst (void){
         if (Camera_Sht_Rqst && !Camera_Busy)
         {
                  Camera_Power(DISABLE);
                  Camera_Sht_Rqst=false;
         }
         if(Wifi_Sht_Rqst && !Wifi_Busy){
                  RN171_Power(DISABLE);
         }
         if(uPhone_Sht_Rqst && !uPhone_Busy){
                  uPhone_Power (DISABLE);
                  uPhone_Sht_Rqst=false;
         }
         if(SHT_Sht_Rqst && !SHT_Busy){
                  SHT_Power(DISABLE);
                  SHT_Sht_Rqst = false;
         }

}

void Camera_Sutdown_Rqst (void){
         Camera_Sht_Rqst=true;
}

void Wifi_Sutdown_Rqst (void){
         Wifi_Sht_Rqst=true;
}

void uPhone_Sutdown_Rqst (void){
         uPhone_Sht_Rqst=true;
}
void SHT_Sutdown_Rqst(void){
         SHT_Sht_Rqst = true;
}



/**
 * \file
 *
```

```
 * \brief Includes of Multisensor Wireless Power Management.
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
 *
 *
 * \license_stop
 *
 */

 #ifndef ML_POWERMGT_H
 #define ML_POWERMGT_H

 #define VIN_MON_PIN                PIN_PB02A_ADCIFE_AD3  /**< \brief ADCIFE signal: AD3 on PB02 mux A */
 #define VIN_MON_MUX                PIN_PB02A_ADCIFE_AD3


 #define VBAT_MON_PIN               PIN_PB03A_ADCIFE_AD4  /**< \brief ADCIFE signal: AD4 on PB03 mux A */
 #define VBAT_MON_MUX               MUX_PB03A_ADCIFE_AD4


/******************** Exported Functions **********************/

void PowerMgt_Init (void);

void Get_Vin_Value (void);
void Get_Vbat_Value (void);

void Check_Shutdown_Rqst (void);
void Camera_Sutdown_Rqst (void);
void Wifi_Sutdown_Rqst (void);
void uPhone_Sutdown_Rqst (void);
void SHT_Sutdown_Rqst (void);

 #endif // USER_BOARD_H
```

8. CÁMARA

```
/**
 * \file
 *
 * \brief Multisensor Wireless JPEG Camera App
 *
 * Copyright (c) 2016 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */

#include "ML_Camera.h"
#include "ENERGY MGT/ML_PowerMgt.h"
#include "LinkSprite.h"
#include "main.h"

/// External variables

extern bool SD_ENABLE;
extern bool Camera_Busy;

extern bool CAMERA_UART_CONFIGURED;
extern bool CAMERA_POWERED;

//// Status variables

bool PICTURE_CAPTURED=false;


/**
 * \brief PictureShooting
 *
 * \note Shooting a new picture at this moment and save in the volatile memory
 *
 * \param none
 *
 * \return none
 */
void PictureShooting (void){
	Camera_Busy=true;
	if (!CAMERA_UART_CONFIGURED)
		{
			Init_LinkSprite ();
			//Send Reset command
			LinkSprite_SendResetCmd();
		}

	if (!CAMERA_POWERED)
	{
		Camera_Power(ENABLE);
	}
	if (!CAMERA_UART_CONFIGURED)
	{
		Init_LinkSprite ();
		//Send Reset command
		LinkSprite_SendResetCmd();
	}

	SendTakePhotoCmd();

	PICTURE_CAPTURED= true;
	Camera_Busy=false;
}

/**
 * \brief PictureShooting
 *
 * \note Delate the picture stored in the volatile memory and shutdown the camera.
 *
 * \param none
```

```
 *
 * \return none
 */
void PictureClear (void){
        PICTURE_CAPTURED= false;
        Camera_Sutdown_Rqst();
}

/**
 * \brief PictureShooting
 *
 * \note Save a picture in the SD Memory and shutdown the camera.
 *                          - If a picture is save in the volatile memory, save It in the SD.
 *                          - If not been saved a picture in the volatile memory, shooting a new picture and
save it in the SD.
 *
 * \param none
 *
 * \return none
 */
void PictureToSD (char *file_name){
        if (SD_ENABLE)
        {
                CAMERA_ERR error = NO_ERR;
                Camera_Busy=true;

                if (!CAMERA_UART_CONFIGURED)
                {
                        error = Init_LinkSprite ();
                        //Send Reset command
                        error = LinkSprite_SendResetCmd();
                }
                if (!CAMERA_POWERED)
                {
                        PICTURE_CAPTURED= false;
                        error = Camera_Power(ENABLE);
                        //Send Reset command
                        error = LinkSprite_SendResetCmd();
                }

                if (!PICTURE_CAPTURED)
                {
                        //Send take picture command
                        error = SendTakePhotoCmd();
                }

                error = GetPictureSize ();

                if (error == NO_ERR)
                {
                        ReadPictureCmd (file_name);
                }else{
                        Camera_Busy=false;
                }

                Camera_Sutdown_Rqst();

                wait(50); // Delay for discard interrupt overlap
                PICTURE_CAPTURED= false;

        }
}

/**
 * \file
 *
 * \brief Multisensor Wireless JPEG Camera APP
 *
 * Copyright (c) 2016 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
 *
 *
 * \license_stop
 *
```

```
 */


#ifndef ML_CAMERA_H_
#define ML_CAMERA_H_


/******************** Exported Functions **********************/

void PictureShooting (void);
void PictureClear (void);

void PictureToSD (char *file_name);


#endif /* ML_CAMERA_H_ */

/**
 * \file
 *
 * \brief LinkSprite Driver
 *
 * Copyright (c) 2016 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */

#include "asf.h"
#include "compiler.h"
#include "LinkSprite.h"
#include "HMI/HMI.h"
#include "main.h"

/// External variable
extern bool Camera_Busy;


uint8_t RX_BUFFER_INDEX=0;
uint8_t TX_BUFFER_INDEX=0;

uint8_t RX_BYTE_TO_TRANSFER=50;
uint8_t TX_BITS_TO_TRANSFER=0;

#define MAX_BITS_TO_RECIVE 50
#define MAX_BITS_TO_TRANSFER 50

 char *TX_BUFFER;
uint32_t RX_BUFFER[MAX_BITS_TO_RECIVE];

uint32_t *RX_BUFFER_POINTER= &RX_BUFFER[0];

static uint32_t PictureArray[16];

uint16_t Picture_Buffer[40];
uint8_t   Picture_BufferIndex=0;


uint8_t PictureAdd_MH, PictureAdd_ML;

uint16_t PictureAdd=0;
uint16_t PictureSize=0;

uint32_t DataToSave=0;

/** State of Camera UART. */
UART_STATE g_uc_state = UART_STATE_EMPTY;

/** State of Camera */
static CAMERA_STATUS camera_state= STATE_POWERUP;


bool CAMERA_UART_CONFIGURED=false;
bool CAMERA_POWERED=false;
```

```
char test_picture_name[] = "0:test_image2.jpeg";

/// Private functions
void LinkSprite_CheckMsg(void);
void LinkSprite_SaveData (uint16_t data);

////////////////////////////////////////////////////////////////////////

/**
 * \brief Interrupt handler for USART. Echo the bytes received and start the
 * next receive.
 */
void LinkSprite_UART_Handler(void){


	uint32_t ul_status;

	uint_fast8_t		status;


	/* Read USART Status. */
	ul_status = usart_get_status(LinkSprite_UART);

	/* Receive buffer is full. */
	if ((ul_status & US_IER_USART_RXRDY)) {
		if (camera_state!=STATE_SAVING){
			#if MAX_BITS_TO_RECIVE<RX_BUFFER_INDEX
			#error "El número de bits a recivir es mayor que el del buffer"
			#endif
			usart_read(LinkSprite_UART, RX_BUFFER_POINTER+RX_BUFFER_INDEX);

			if(RX_BUFFER_INDEX==(RX_BYTE_TO_TRANSFER-1)){
				usart_disable_interrupt(LinkSprite_UART, US_IDR_RXRDY);
				LinkSprite_CheckMsg();
				g_uc_state = UART_STATE_EMPTY;
				}else{
				g_uc_state = UART_STATE_READ;
				RX_BUFFER_INDEX++;
			}
		}
		if (camera_state==STATE_SAVING){

			usart_read(LinkSprite_UART, &DataToSave);
			LinkSprite_SaveData (DataToSave);
		}


	}

	/*TX buffer is ready */
	if ((ul_status & US_CSR_TXRDY)) {

		#if MAX_BITS_TO_TRANSFER<TX_BITS_TO_TRANSFER
			#error "El número de bits a transmitir es mayor que el del buffer"
		#endif
		uint32_t *punt = (TX_BUFFER+(TX_BUFFER_INDEX*4));
		uint32_t datatt = *punt;
		usart_write(LinkSprite_UART,datatt);

			if(TX_BUFFER_INDEX==(TX_BITS_TO_TRANSFER-1)){
				usart_disable_interrupt(LinkSprite_UART, US_IDR_TXRDY);
				if(RX_BYTE_TO_TRANSFER==0){
				g_uc_state = UART_STATE_EMPTY;
				}else{
					g_uc_state = UART_STATE_READ;
				}

			}else{
				TX_BUFFER_INDEX++;
			}

		}
	else{

	}

}
```

```
/**
 * \brief Init_LinkSprite
 *
 * \note
 *
 * \param none
 *
 * \return none
 */
CAMERA_ERR Init_LinkSprite (void){

        /*Configure UART HARDWARE */
        ioport_set_pin_mode(PIN_PA05B_USART0_RXD,MUX_PA05B_USART0_RXD);
        ioport_disable_pin(PIN_PA05B_USART0_RXD);

        ioport_set_pin_mode(PIN_PA07B_USART0_TXD,MUX_PA07B_USART0_TXD);
        ioport_disable_pin(PIN_PA07B_USART0_TXD);

        /* Configure ENABLE pin */

        ioport_set_pin_dir(CAMERA_EN_PIN, IOPORT_DIR_OUTPUT);
        ioport_set_pin_mode(CAMERA_EN_PIN, IOPORT_MODE_PULLUP);



  /*Configure UART settings */
                const sam_usart_opt_t usart_camera_settings = {
                        LinkSprite_UART_BAUDRATE,
                        LinkSprite_UART_CHAR_LENGTH,
                        LinkSprite_UART_PARITY,
                        LinkSprite_UART_STOP_BITS,
                        US_MR_CHMODE_NORMAL,
                        /* This field is only used in IrDA mode. */
                        0
                };

        /* Enable the peripheral clock in the PMC. */
        sysclk_enable_peripheral_clock(LinkSprite_UART);


        usart_init_modem(LinkSprite_UART,&usart_camera_settings,
                         sysclk_get_peripheral_bus_hz(LinkSprite_UART));

        /* Disable all the interrupts. */
        usart_disable_interrupt(LinkSprite_UART, ALL_INTERRUPT_MASK);
        usart_enable_interrupt(LinkSprite_UART, US_IER_USART_RXRDY);

        /* Enable the receiver and transmitter. */
        usart_enable_tx(LinkSprite_UART);
        usart_enable_rx(LinkSprite_UART);
        CAMERA_UART_CONFIGURED=true;
        /* Configure and enable interrupt of USART. */
        NVIC_SetPriority(LinkSprite_UART_IRQn, 0);
        NVIC_EnableIRQ(LinkSprite_UART_IRQn);
        Camera_Power (ENABLE);

}

/**
 * \brief Camera_Power
 *
 * \note: Turn on or turn off the power supply off JPEG Camera.
 *
 * \param STATE: ENABLE or DISABLE
 *
 * \return none
 */
CAMERA_ERR Camera_Power (bool STATE){
  if(STATE == ENABLE){
        RX_BYTE_TO_TRANSFER=10;
        RX_BUFFER_INDEX=0;

        camera_state=STATE_POWERUP;
        ioport_set_pin_level(CAMERA_EN_PIN, HIGH);
                 wait(1);

        if(camera_state!=STATE_RUN){
```

```
                camera_state=STATE_NOCAMERA;
                return ERR_NOCAMERA;
        }else{
                return NO_ERR;
        }
        CAMERA_POWERED=true;
  }
  else{
        camera_state=STATE_SHUTDOWN;
        ioport_set_pin_level(CAMERA_EN_PIN, LOW);
        CAMERA_POWERED=false;
  }
}



void LinkSprite_Write (uint8_t nDATA, const char *string){

        if (g_uc_state == UART_STATE_READ || g_uc_state == UART_STATE_EMPTY)
        {
                g_uc_state = UART_STATE_WRITE;
                TX_BUFFER_INDEX=0;
                TX_BITS_TO_TRANSFER=nDATA;
                TX_BUFFER=string;
        usart_enable_interrupt(LinkSprite_UART, US_IER_USART_TXRDY);
        }

}



//Send Reset command
CAMERA_ERR LinkSprite_SendResetCmd (void){
     camera_state = STATE_RESET;

                PictureArray[0]=0x56;
                PictureArray[1]=0x00;
                PictureArray[2]=0x26;
                PictureArray[3]=0x00;


                RX_BYTE_TO_TRANSFER=10;
                RX_BUFFER_INDEX=0;

                LinkSprite_Write(4,&PictureArray[0]);
        wait(1);

        if(camera_state!=STATE_RUN){
                camera_state=STATE_NOCAMERA;
                return ERR_NOCAMERA;
        }else{
                return NO_ERR;
        }
}

//Send take picture command
CAMERA_ERR SendTakePhotoCmd (void) {
 if (camera_state!=STATE_NOCAMERA){
           camera_state = STATE_TAKEPICTURE;


                PictureArray[0]=0x56;
                PictureArray[1]=0x00;
                PictureArray[2]=0x36;
                PictureArray[3]=0x01;
                PictureArray[4]=0x00;

                RX_BYTE_TO_TRANSFER=5;
                RX_BUFFER_INDEX=0;

                LinkSprite_Write(5,&PictureArray[0]);
        wait(10);
        if(camera_state!=STATE_PICTURECAPTURED){
                return ERR_SHOOT;
        }else{
                        return NO_ERR;
```

```
        }

 }
}

CAMERA_ERR GetPictureSize (void){
        if (camera_state!=STATE_NOCAMERA)
        {
        camera_state = STATE_GETPICTURESIZE;


                PictureArray[0]=0x56;
                PictureArray[1]=0x00;
                PictureArray[2]=0x34;
                PictureArray[3]=0x01;
                PictureArray[4]=0x00;

                RX_BYTE_TO_TRANSFER=9;
                RX_BUFFER_INDEX=0;

                LinkSprite_Write(5,&PictureArray[0]);
                wait(2);
                if (camera_state!= STATE_READYTOREAD){
                        return ERR_GETSIZE;
                }else{
                        return NO_ERR;
                }
        }
}

void ReadPictureCmd (char *file_name){
        if (camera_state!=STATE_NOCAMERA)
        {
                if(camera_state!=STATE_SAVING){
                        /// EXTENSION OF FILE
                                file_name[25]='P';
                                file_name[26]='I';
                                file_name[27]='C';
                                file_name[28]='.';
                                file_name[29]='j';
                                file_name[30]='p';
                                file_name[31]='g';
                        SD_OpenFile (file_name,OVERWRITE);
                        ioport_set_pin_level(BLUE_LED_PIN, IOPORT_PIN_LEVEL_LOW);
                        PictureAdd=0;
                }


                camera_state = STATE_READPICTURE;
                RX_BYTE_TO_TRANSFER=5;
                RX_BUFFER_INDEX=0;


                PictureAdd_ML= (uint8_t) PictureAdd;
                PictureAdd_MH= (PictureAdd>>8);

                Picture_BufferIndex=0;

                PictureArray[0]=0x56;
                PictureArray[1]=0x00;
                PictureArray[2]=0x32;
                PictureArray[3]=0x0C;
                PictureArray[4]=0x00;
                PictureArray[5]=0x0A;
                PictureArray[6]=0x00;
                PictureArray[7]=0x00;
                PictureArray[8]=PictureAdd_MH;
                PictureArray[9]=PictureAdd_ML;
                PictureArray[10]=0x00;
                PictureArray[11]=0x00;
                PictureArray[12]=0x00; /// MSB READ SIZE PACKET
                PictureArray[13]=0x20; /// LSB READ SIZE PACKET
                PictureArray[14]=0x00;
                PictureArray[15]=FRAMES_DELAY;

                LinkSprite_Write(16,&PictureArray[0]);
        }
```

```
}

/**
 * \brief Change_ImageSize
 *
 * \note This function change the image size of LinkSprite Camera.
 *                          Image size options:
 *                                  - 320x240
 *                                  - 640x480
 *                                  - 160x120
 * \param image_size: select the image size
 *                                  - IS_320_240
 *                                  - IS_640_480
 *                                  - IS_160_120
 *
 * \return none
 */

void Change_ImageSize (uint8_t image_size){

        camera_state = STATE_CONFIG;

         static uint32_t SizeArray[9];
        SizeArray[0]=0x56;
        SizeArray[1]=0x00;
        SizeArray[2]=0x31;
        SizeArray[3]=0x05;
        SizeArray[4]=0x04;
        SizeArray[5]=0x01;
        SizeArray[6]=0x00;
        SizeArray[7]=0x19;
        SizeArray[8]=0x22;

                RX_BYTE_TO_TRANSFER=0;
                RX_BUFFER_INDEX=0;

        LinkSprite_Write(9,&SizeArray[0]);
}

void LinkSprite_SaveData (uint16_t data){

        bool EndFile=false;

        Picture_Buffer[Picture_BufferIndex]=data;

        Picture_BufferIndex++;

        if (Picture_BufferIndex>37){
                uint8_t count;

                for (count=1;(count<=32) ;count++)
                {
                        if(!EndFile){

                        SD_PrintInFile ((char) Picture_Buffer[count]);
                        if (Picture_Buffer[count-1]==0xff && Picture_Buffer[count]==0xD9){
                                bool EndFile=true;
                                SD_CloseFile ();
                                ioport_set_pin_level(BLUE_LED_PIN, IOPORT_PIN_LEVEL_HIGH);
                                camera_state=STATE_RUN;
                                Camera_Busy=false;
                        }
                        PictureAdd++;
                        }

                }

                if (!EndFile && (PictureAdd<PictureSize))
                {
                        ReadPictureCmd (0x000);
                }
        }

}
/**
 * \brief LinkSprite_CheckMsg
 *
 * \note This function interpret the messages of camera
 *
```

```
 * \param  none
 *
 * \return none
 */
void LinkSprite_CheckMsg(void){

        switch(camera_state){

                case STATE_POWERUP:
                        if((RX_BUFFER[7]== 0x6e) && (RX_BUFFER[8]== 0x64)){
                                camera_state = STATE_RUN;
                        }

                break;

                case STATE_RESET:
                        if((RX_BUFFER[0]== 0x0d) && (RX_BUFFER[1]== 0x0a) &&(RX_BUFFER[2]== 0x49)){
                                camera_state = STATE_RUN;
                        }
                break;

                case STATE_RUN:

                break;

                case STATE_GETPICTURESIZE:
                if((RX_BUFFER[0]== 0x76) && (RX_BUFFER[2]== 0x34)){
                                uint8_t   PictureSize_XH, PictureSize_XL;
                                PictureSize_XH=RX_BUFFER[7];
                                PictureSize_XL=RX_BUFFER[8];
                                PictureSize= PictureSize_XH<<8 | PictureSize_XL;
                                camera_state = STATE_READYTOREAD;

                        }
                break;

                case STATE_READPICTURE:
                if((RX_BUFFER[0]== 0x76) && (RX_BUFFER[2]== 0x32)){
                        camera_state=STATE_SAVING;
                }
                break;

                case STATE_TAKEPICTURE:
                if((RX_BUFFER[0]== 0x76) && (RX_BUFFER[2]== 0x36)){
                                camera_state = STATE_PICTURECAPTURED;
                        }

                break;
                case STATE_CONFIG:
                        if(RX_BUFFER[0]==0x76 && RX_BUFFER[2]==0x31){
                                LinkSprite_SendResetCmd ();
                        }
                break;

                default:

                break;
        }
        int i;
        for(i=RX_BUFFER_INDEX;i>=0;i=i-1){
                RX_BUFFER[i]=0;
        }
        RX_BUFFER_INDEX=0;
        usart_enable_interrupt(LinkSprite_UART, US_IER_USART_RXRDY);

}

/**
 * \file
 *
 * \brief LinkSprite Driver.
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
 *
 * \license_start
 *
 * \page License
```

```
 *
 *
 * \license_stop
 *
 */

#ifndef LINKSPRITE_H_INCLUDED
#define LINKSPRITE_H_INCLUDED

#include "compiler.h"

/// @cond 0
/**INDENT-OFF**/
#ifdef __cplusplus
extern "C" {
        #endif
        /**INDENT-ON**/
        /// @endcond


/// PIN CONFIGURATIONS FOR LinkSprite

/// UART JPEG CAMERA
#define CAMERA_USART_RX_PIN             PIN_PA05B_USART0_RXD
#define CAMERA_USART_RX_GPIO            GPIO_PA05B_USART0_RXD
#define CAMERA_USART_RX_MUX             MUX_PA05B_USART0_RXD

#define CAMERA_USART_TX_PIN             PIN_PA07B_USART0_TXD
#define CAMERA_USART_TX_GPIO            GPIO_PA07B_USART0_TXD
#define CAMERA_USART_TX_MUX             MUX_PA07B_USART0_TXD

#define CAMERA_EN_PIN                       PIN_PA11

/** USART Interface */
#define LinkSprite_UART             USART0
#define LinkSprite_USART_USART_ID           ID_USART0
/** Baudrate setting */
#define LinkSprite_UART_BAUDRATE    115200

/** Character length setting */
#define LinkSprite_UART_CHAR_LENGTH  US_MR_CHRL_8_BIT
/** Parity setting */
#define LinkSprite_UART_PARITY       US_MR_PAR_NO
/** Stop bits setting */
#define LinkSprite_UART_STOP_BITS    US_MR_NBSTOP_1_BIT

#define LinkSprite_UART_Handler             USART0_Handler

#define LinkSprite_UART_IRQn                USART0_IRQn



/*** CAMERA STATUS ***/
typedef enum {
        STATE_POWERUP=0,
        STATE_RESET,
        STATE_CONFIG,
        STATE_TAKEPICTURE,
        STATE_PICTURECAPTURED,
        STATE_READPICTURE,
        STATE_GETPICTURESIZE,
        STATE_READYTOREAD,
        STATE_SAVING,
        STATE_RUN,
        STATE_SHUTDOWN,
        STATE_NOCAMERA
} CAMERA_STATUS;

typedef enum {
        ERR_NOCAMERA=0,
        ERR_SHOOT,
        ERR_GETSIZE,
        NO_ERR
} CAMERA_ERR;

typedef enum{
        UART_STATE_READ=0,
```

```
        UART_STATE_WRITE,
        UART_STATE_EMPTY,
} UART_STATE;


/** All interrupt mask. */
#define ALL_INTERRUPT_MASK  0xffffffff

/** Image sizes **/

#define IS_320_240                          0x00
#define IS_640_480                          0x11
#define IS_160_120                          0x22

/* Delay betwen frames*/
#define FRAMES_DELAY                        0x01
/******************** Exported Functions **********************/

CAMERA_ERR Init_LinkSprite (void);
CAMERA_ERR Camera_Power (bool STATE);

void LinkSprite_Write (uint8_t nDATA, const char *string);

//Send Reset command
CAMERA_ERR LinkSprite_SendResetCmd(void);
//Send take picture command
CAMERA_ERR SendTakePhotoCmd(void);

void ReadPictureCmd (char *file_name);
CAMERA_ERR GetPictureSize (void);

void Change_ImageSize (uint8_t image_size);

#endif /* LINKSPRITE_H_INCLUDED */
```

9. AUDIO

```
/**
 * \file
 *
 * \brief RN171 Driver
 *
 * Copyright (c) 2015 GABRIEL AZNAR LAPUENTE. All rights reserved.
 *
 * \license_start
 *
 * \page License
 *
 * \license_stop
 *
 */
#include "asf.h"
#include "compiler.h"
#include "ML_uPhone.h"


#include "ENERGY MGT/ML_PowerMgt.h"
#include "HMI/HMI.h"


#include "SD MEMORY/ML_SDcard.h"

char audio_file_name[] = "0:audio.txt";

extern bool uPhone_Busy;

/** Low threshold */
static uint16_t gs_us_low_threshold = 0;
/** High threshold */
static uint16_t gs_us_high_threshold = 0;

/** ADC instance */
struct adc_dev_inst g_adc_inst;

static uint8_t AUDIO_BUFFER[20000];
static uint32_t nMuestras = 20000;

static uint32_t AUDIO_BUFFER2;
static uint8_t AUDIO_BUFFER2_INDEX=0;


//static uint8_t AUDIO_BUFF_CENT, AUDIO_BUFF_DEC, AUDIO_BUFF_UNI;
static uint32_t AUDIO_BUFFER_INDEX=0;



bool Recording = false;

char test_audio_name[] = "0:test_audio.txt";

//Private Functions
void Init_uPhone (void);
void StartToRecord (void);
void StopRecord (void);

/**
 * \brief Callback function for ADCIFE interrupt.
 */
static void adcife_wm_handler(void)
{

	AUDIO_BUFFER2=AUDIO_BUFFER2+(adc_get_last_conv_value(&g_adc_inst));
	AUDIO_BUFFER2_INDEX++;

	if (AUDIO_BUFFER2_INDEX>=6)
	{
		AUDIO_BUFFER[AUDIO_BUFFER_INDEX]=AUDIO_BUFFER2/6;
		AUDIO_BUFFER_INDEX++;
		AUDIO_BUFFER2_INDEX=0;
	}
```

```
		if (AUDIO_BUFFER_INDEX>=nMuestras)
		{
			adc_disable_interrupt(&g_adc_inst, ADCIFE_IRQn);
			StopRecord();
		}
		adc_clear_status(&g_adc_inst, ADCIFE_SCR_SEOC);
}

void AudioClipToSD (char *file_name){

		uPhone_Busy=true;
		/// EXTENSION OF FILE
		file_name[25]='A';
		file_name[26]='U';
		file_name[27]='D';
		file_name[28]=46;
		file_name[29]=116;
		file_name[30]=120;
		file_name[31]=116;



		Init_uPhone();

		Led_set(RED_LED_PIN);
		StartToRecord();

		uPhone_Sutdown_Rqst();
		while(Recording);
		Led_reset(RED_LED_PIN);

		SD_OpenFile (file_name, OVERWRITE);


			Led_set(BLUE_LED_PIN);
		uint32_t i;
		uint16_t audio_data;
		for (i=0;i<nMuestras;i++){
			SD_PrintInFile (AUDIO_BUFFER[i]);

		}
		SD_CloseFile ();
		Led_reset(BLUE_LED_PIN);


}


void Init_uPhone (void)
{
			//CONFIGURE PIN MUX

				ioport_set_pin_mode(PIN_UPHONE_AD,MUX_UPHONE_AD);
				ioport_disable_pin(PIN_UPHONE_AD);

			/* Initialize threshold. */
			gs_us_low_threshold = 500;
			gs_us_high_threshold = 2000;

			struct adc_config adc_cfg = {
				/* System clock division factor is 4 */
				.prescal = ADC_PRESCAL_DIV4,
				/* The APB clock is used */
				.clksel = ADC_CLKSEL_APBCLK,
				/* Max speed is 150K */
				.speed = ADC_SPEED_75K,
				/* ADC Reference voltage is Vref */
				.refsel = ADC_REFSEL_2,
				/* Enables the Startup time */
				.start_up = CONFIG_ADC_STARTUP
			};
			struct adc_seq_config adc_seq_cfg = {
				/* Select Vref for shift cycle */
				.zoomrange = ADC_ZOOMRANGE_0,
				/* Pad Ground */
				.muxneg = ADC_MUXNEG_1,
				/* DAC Internal */
				.muxpos = ADC_MUXPOS_5,
```

```
                        /* Enables the internal voltage sources */
                        .internal = ADC_INTERNAL_3,
                        /* Disables the ADC gain error reduction */
                        .gcomp = ADC_GCOMP_DIS,
                        /* Disables the HWLA mode */
                        .hwla = ADC_HWLA_DIS,
                        /* 12-bits resolution */
                        .res = ADC_RES_8_BIT,
                        /* Enables the single-ended mode */
                        .bipolar = ADC_BIPOLAR_DIFFERENTIAL
                };


                adc_init(&g_adc_inst, ADCIFE, &adc_cfg);
                adc_enable(&g_adc_inst);

                struct adc_ch_config adc_ch_cfg = {
                .seq_cfg = &adc_seq_cfg,
                /* Window monitor mode is off */
                .window_mode = ADC_WM_OFF,
                                };

                adc_ch_set_config(&g_adc_inst, &adc_ch_cfg);

                uPhone_Power(ENABLE);


}

void uPhone_Power (bool STATE){
        if(STATE==ENABLE){
                //ENABLE UPHONE    POWER
                ioport_set_pin_dir(PIN_PB00, IOPORT_DIR_OUTPUT);
                ioport_set_pin_mode(PIN_PB00, IOPORT_MODE_DRIVE_STRENGTH);
                ioport_set_pin_level(PIN_PB00, IOPORT_PIN_LEVEL_HIGH);
        }
        if(STATE==DISABLE){
                //ENABLE UPHONE    POWER
                ioport_set_pin_dir(PIN_PB00, IOPORT_DIR_INPUT);
                ioport_set_pin_level(PIN_PB00, IOPORT_PIN_LEVEL_LOW);
        }

}

void StartToRecord (void){

        Recording=true;
        AUDIO_BUFFER_INDEX=0;
                //empezar conversión
        //adc_enable(&g_adc_inst);
        adc_configure_trigger(&g_adc_inst, ADC_TRIG_CON);
        adc_configure_gain(&g_adc_inst, ADC_GAIN_1X);

                        adc_set_callback(&g_adc_inst, ADC_SEQ_SEOC, adcife_wm_handler,
        ADCIFE_IRQn, 1);


        adc_start_software_conversion(&g_adc_inst);

}

void StopRecord (void){


        adc_disable(&g_adc_inst);
        Recording=false;
        uPhone_Busy= false;
}


/**
 * \file
 *
 * \brief RN171 Driver.
 *
 * Copyright (c) 2015 Gabriel Aznar Lapuente. All rights reserved.
 *
 * Project MEMORY LANE
```

```
 *
 * \license_start
 *
 * \page License
 *
 *
 * \license_stop
 *
 */


#ifndef ML_UPHONE_H_
#define ML_UPHONE_H_

#define PIN_UPHONE_AD                          PIN_PB04A_ADCIFE_AD5              /**< \brief ADCIFE signal:
AD5 on PB04 mux A */
#define MUX_UPHONE_AD                          MUX_PB04A_ADCIFE_AD5



/******************** Exported Functions **********************/
void uPhone_Power (bool STATE);

void AudioClipToSD (char *file_name);


#endif /* ML_UPHONE_H_ */
```