CAPÍTULO 4.
WASPMOTE 868 MHz.

En el presente proyecto se han elegido los sensores inalámbricos Waspmote v1.0, de la empresa española Libelium. Como se verá en este capítulo, es un dispositivo modular, que permite instalar distintos sensores de diversas clases y también de diversos transceptores radio xBee de la empresa Digi.

Gracias a esta modularidad, y a que Libelium provee de APIs para comunicaciones que facilitan la programación, se pensó en dos tecnologías, la de 868 MHz (protocolo privativo *Digi RF*) y la de 2.45 GHz (*ZigBee*).

Debido a las largas distancias a cubrir y a que el canal tiene unos puntos bien delimitados (almenaras y caudalímetros), se descartó la tecnología ZigBee a favor de la de 868 MHz. De ahí que se llame el capítulo "Waspmote 868 MHz".

En este capítulo se describe de manera resumida, las características más destacables de este dispositivo.

4.1. Hardware.

4.1.1. Arquitectura modular.

Waspmote se basa en una arquitectura modular. El usuario puede cambiar o integrar los módulos necesarios en los diferentes sockets (ver figura 4.1) según las necesidades.

Los módulos disponibles para integrar en Waspmote se clasifican en:

- Módulos ZigBee/802.15.4 (2.4GHz, 868MHz, 900MHz). Baja y alta potencia.
- Módulo GSM/GPRS (Quadband: 850MHz/900MHz/1800MHz/1900MHz).
- Módulo GPS.
- Módulos Sensoriales (Placas de Sensores).
- Módulo de almacenamiento: SD Memory Card.

4.1.2. Especificaciones técnicas.

Microcontrolador: ATmega1281
Frecuencia: 8MHz
SRAM: 8KB
EEPROM: 4KB
FLASH: 128KB
SD Card: 2GB
Peso: 20gr

Dimensiones: 73.5 x 51 x 13 mm

Rango de Temperatura: [-20ºC, +65ºC]

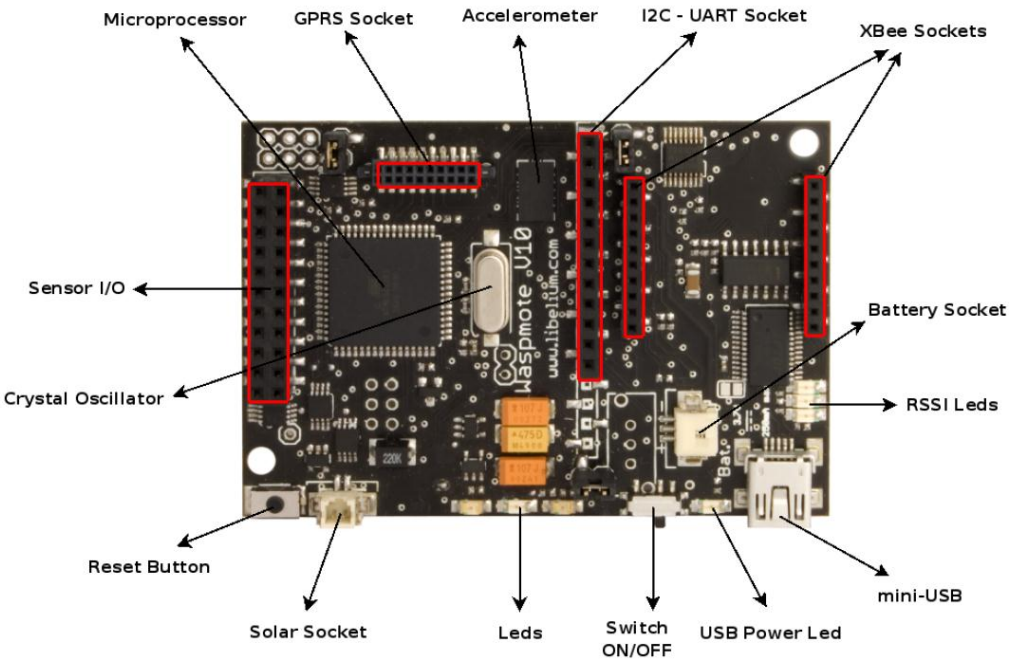

Figura 4.1: Cara superior del Waspmote y los sockets disponibles.

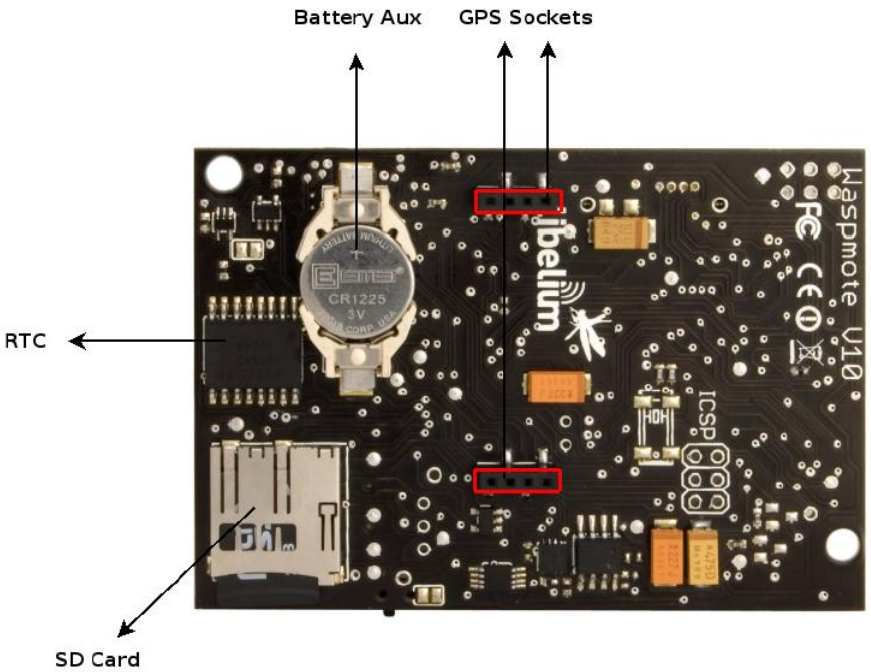

Figura 4.2: Parte inferior del Waspmote y sockets disponibles.

4.1.3. Diagrama de bloques.

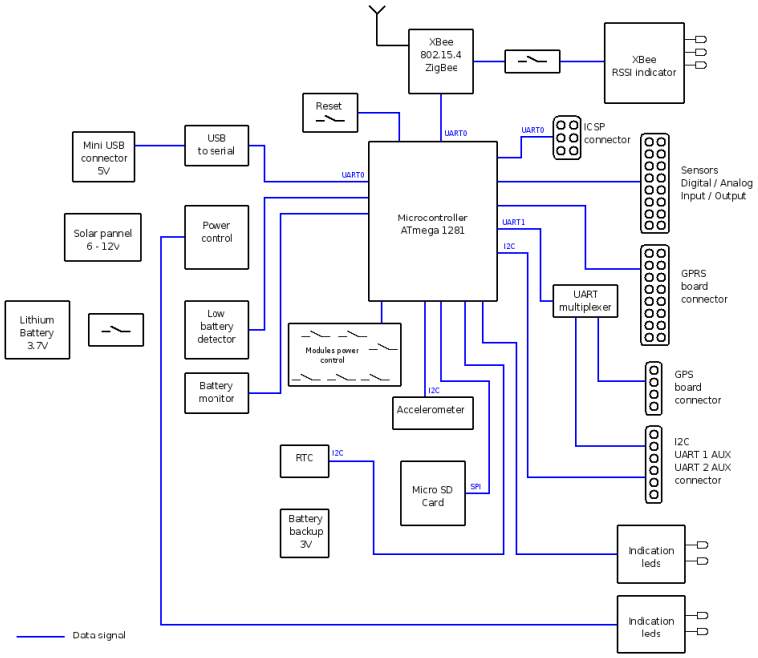

Figura 4.3: Diagrama de bloques y señales de datos del Waspmote.

En la figura se pueden observar los diferentes bloques que componen el Waspmote y las señales de datos.

4.1.4. Entrada/Salida.

Waspmote puede comunicarse con otros dispositivos externos mediante los diferentes puertos de entrada/salida que posee.

Los dispositivos con los que Waspmote se puede comunicar pueden ser cualquier sensor, componente o módulo electrónico siempre y cuando se respeten las especificaciones requeridas por cada puerto.

Dispone de dos conectores, el “Sensor I/O” y el “I2C UART socket” como se puede observar en la figura 4.1.

Las entradas y salidas del Waspmote se pueden resumir en las siguientes:

- **Analógicas.** 7 entradas accesibles en el conector de sensores y conectadas directamente al microcontrolador (éste tiene un ADC de 10 bits cuyo valor máximo de entrada es 3.3 V).
- **Digitales.** 8 pines digitales configurables como entrada o salida, cuyos valores lógicos 0 y 1 corresponden a 0V y 3.3V, respectivamente.

- **PWM (Pulse Width Modulation).** Existe un pin que permite simular una onda analógica entre 0 y 3.3 V mediante modulación por anchura de pulso.
- **UART.** Waspmote dispone de 6 puertos serie. Una de las UART del microcontrolador está conectada simultáneamente al módulo de comunicación XBee y al puerto USB. (Por tanto, no se pueden usar simultáneamente el xBee y el puerto USB).
  La otra UART del microcontrolador está conectada a un multiplexor de cuatro canales, pudiendo seleccionar desde el código cuál de las cuatro nuevas UART queremos conectar a la UART del microcontrolador. Estas cuatro nuevas UART están conectadas dela siguiente manera. Una está conectada a la placa GPRS, otra al GPS y las otras dos quedan accesibles al usuario en el conector I2C – UART auxiliares.
- **I2C.** En Waspmote también se utiliza el bus de comunicación I2C, donde se conectan en paralelo tres dispositivos: el acelerómetro, el RTC y el potenciómetro que configura el nivel de umbral de alarma por batería baja. En todos los casos el microcontrolador actúa como maestro (master) mientras que el resto de los dispositivos conectados al bus actúan como esclavos (slave).
- **SPI.** El puerto SPI del microcontrolador se utiliza para la comunicación de éste con la tarjeta micro SD. Todas las operaciones de uso del bus son realizadas por la librería específica de forma transparente.
- **USB.** La comunicación USB se utiliza en Waspmote para la comunicación con un ordenador. Esta comunicación permite la carga del programa al microcontrolador y la comunicación de datos durante la ejecución del programa. Para la comunicación USB se utiliza una de las UART del microcontrolador y de la conversión al estándar USB se encarga el FT232RL. De esta forma en el ordenador tendremos un nuevo puerto de comunicación serie (virtual) listo para comunicarse con Waspmote.

4.1.5. RTC.

El RTC (Real Time Clock) o reloj de tiempo real integrado de tipo DS3231SN de Maxim, mantiene informado al Waspmote del momento temporal en el que se encuentra. Se programa a través del bus I2C y permite programar acciones, como por ejemplo, hacer que el mote hiberne y se despierte en un determinado momento gracias al uso de interrupciones.

Por otra parte, dispone de un sensor de temperatura integrado, que utiliza para recalibrarse, y al que se accede mediante el bus I2C. El sensor se presenta en un formato de 10 bits complemento a dos, ocupando 2 registros. Tiene una resolución de +0.25 °C. El rango de temperaturas que se pueden medir está comprendido entre -40 °C y +85 °C.

En el proyecto se ha hecho uso de este sensor, para mandar información de temperatura en los mensajes de Keepalive enviados periódicamente por los Waspmotes.

4.1.6. Acelerómetro.

Waspmote tiene integrado el sensor de aceleración LIS3LV02DL de STMicroelectronics que informa al mote de las variaciones de aceleración sufridas en cada uno de los 3 ejes (X,Y,Z). La integración de este sensor permite la medición de la aceleración en los 3 ejes (X,Y,Z), estableciendo 2 tipos de eventos (interrupciones): caída libre (Free Fall) y detección de cambio de dirección (Direction Detection Change). De esta forma, se pueden realizar acciones como despertar al mote cuando se produce una caída libre o un cambio de dirección y realizar la acción asignada a tal efecto, en caso de tener al mote dormido.

4.1.7. Fuentes de alimentación.

Para alimentar el Waspmote hay 3 opciones:

- **Batería:** Se incluye una batería de Ion de Litio (Li-Ion) de tensión nominal 3.7V. En lo referente a la capacidad de la batería existen dos posibilidades: 1150 mAh y 2300 mAh.
- **Panel solar:** Waspmote dispone de un conector y un cable especial para conectar a un panel solar. Se admiten placas solares de hasta 12V y corriente máxima de carga a través de placa de 240 mA.
- **USB:** Permite conectar mediante un cable USB a un PC, una fuente de 220V o un conector de vehículo.

Para la implementación real, se pueden alimentar mediante USB los nodos situados en Almenaras y Caudalímetros, mientras que en el caso de los nodos repetidores, al estar en puntos sin conexión a la red eléctrica, se pueden alimentar mediante paneles solares. No es necesario, por tanto, el uso de baterías.

4.1.8. Otros componentes.

Además de los componentes descritos en los apartados anteriores, el Waspmote dispone de otros componentes que se nombran a continuación:

- **LEDs programables.** Se dispone de 2 leds que se pueden programar con funciones de la API de Waspmote.
- **Memoria EEPROM.** Existe una memoria no volátil de 4KB donde se pueden guardar valores de forma permanente.
- **Tarjeta de memoria flash micro-SD.** Waspmote integra un soporte de almacenamiento externo como son las tarjetas micro-SD (Secure Digital). Waspmote usa el sistema de ficheros FAT16 y puede soportar tarjetas de hasta 2GB y la comunicación desde el microcontrolador hasta la tarjeta se realiza mediante el bus SPI. Esta es una alternativa para grabar datos de manera masiva frente a la EEPROM.

- **Módulos externos.** Libelium pone a la disposición una serie de módulos que se pueden instalar en el Waspmote, como son por ejemplo el módulo GSM/GPRS, que permite recibir datos, llamadas o SMS de la red móvil, o el GPS, que permite conocer la localización exacta del mote en exteriores y sincronizar el RTC interno con la hora real. Además, se pueden conectar placas de sensores de gases, eventos, prototipado, agricultura, agua, etc., por lo que el dispositivo es extremadamente versátil.

En el proyecto se han utilizado los componentes internos que venían en el kit de Waspmotes, como son los LEDs y la memoria EEPROM con la excepción de las tarjetas micro-SD, que sí se han utilizado para hacer pruebas indoor con el objetivo de guardar datos de monitorización de manera temporal.

4.2. xBee 868.

Aunque el transceptor radio xBee 868 forma parte del Waspmote (se coloca en el *socket* xBee, ver figura 4.1), es un elemento independiente y de gran importancia para el proyecto, y es por ello, por lo que se le dedica un apartado independiente en este capítulo.

Figura 4.4: xBee 868 MHz con conector RP-SMA y antena sleeve dipole de 0 dBi.

Las características generales se resumen en la siguiente tabla:

Módulo	Frecuencia	Potencia de Transmisión	Sensibilidad de Recepción	Canales
xBee 868	869.4 – 869.65MHz	315 mW (25 dBm)	-112 dBm	1

Tabla 4.1: Características generales del transceptor xBee 868.

La frecuencia utilizada es la banda de 869MHz (Europa), utilizando 1 único canal. El uso de este módulo está permitido únicamente en Europa. Se proporciona cifrado mediante el algoritmo AES 128b, concretamente mediante el tipo AES-CTR.

La topología clásica de este tipo de red es P2P, puesto que los nodos pueden establecer conexiones punto a punto con los nodos hermanos mediante el uso de parámetros como la

dirección MAC o la de red. Uno de los objetivos del proyecto, es poder encaminar paquetes, creando una topología más compleja, como se verá en apartados posteriores.

4.2.1. Parámetros de los nodos xBee 868.

Los parámetros manejables para identificar un nodo en la red y poder usar las funciones de la API de xBee son los siguientes:

- **Dirección MAC.** Es la dirección IEEE física de 64 bits del dispositivo que lo identifica de manera unívoca.
- **PAN ID.** Identificador de la red de 16 bits. Todos los nodos de una misma red deberían tener el mismo PAN ID.
- **Identificador de nodo (NI).** Una cadena de caracteres ASCII con longitud máxima 20 caracteres (20 bytes) que identifica a un nodo en la red en el nivel de aplicación. Sirve para buscar un nodo por su NI.
- **Identificador de red (NA).** Es un identificador de 16 bits que permite distinguir los nodos cuando se transmite en modo broadcast. No hay que confundirlo con el PAN ID y no sirve en modo unicast.

En el proyecto se ha usado las direcciones MAC para las comunicaciones a nivel 2, debido al número limitado de nodos de la red y se ha usado un direccionamiento para el nivel de red similar a las direcciones NA que proporciona la API de xBee, para identificar a los nodos de manera más sencilla. La razón de esto, es que en realidad, cuando se mandan los paquetes, las direcciones NA, que podría pensarse que son aptas para usarse como direcciones de red, siempre se cambian a '0xFFFE', por lo que no se pueden utilizar para distinguir origen y destino. Por otra parte, puesto que sólo hay una red, se ha obviado el PAN ID.

4.2.2. Modos de transmisión.

Hay dos modos de transmisión:

- **Unicast.** La dirección MAC de destino corresponde a la dirección de MAC del destino. Sólo el equipo con dicha dirección aceptará la trama.
- **Broadcast.** La dirección MAC de destino es la dirección de broadcast. Todos los equipos aceptarán tramas broadcast.

En la mayoría de los casos se usa el modo unicast, ya que los mensajes se envían desde un nodo hacia otro, encaminándose hacia el destino final (el nodo de la oficina). Tan sólo se ha utilizado el modo broadcast en unos mensajes especiales usados por el protocolo de red que se difunden periódicamente hacia todos los demás nodos con el objetivo de informar a los nodos adyacentes de que el nodo sigue vivo, y en el caso que corresponda, que dichos nodos actualicen sus tablas de encaminamiento con el objetivo de que los nodos vayan por la ruta óptima.

4.2.3. Protocolo RF de Digi.

Al contrario que en el caso de los xBee ZigBee, por ejemplo, en el caso del xBee 868 el protocolo de transmisión que se utiliza no dispone de nivel de red, y es por eso que el único tipo de topología posible (en principio) es *peer-to-peer* (punto a punto) y las direcciones que se usan realmente son direcciones MAC (nivel de enlace).

Según el fabricante, Digi, las capas de la torre del protocolo RF del xBee 868 son las que se ven en la figura 4.5.

Figura 4.5: Torre del protocolo RF de Digi.

- **Seguridad.** Se encarga de encriptar los datos que se transmiten via radiofrecuencia usando encriptación AES 128 bits. Por otra parte, también se encarga de desencriptar los paquetes encriptados que se reciben.
- **Nivel de Red.** No existe en el caso del xBee 868, por tanto, no es posible (en principio) una comunicación entre nodos separados varios saltos (es decir, alcance radio directo), ni tampoco entre varias redes. Por tanto, el protocolo no proporciona la capacidad de encaminamiento.
- **MAC.** La capa MAC o nivel de enlace, proporciona entrega fiable punto a punto. Añade un código CRC al final de las tramas RF para asegurara la integridad de las mismas. Por otra parte, realiza retransmisiones y asentimientos para mejorar la fiabilidad en la entrega de las tramas.
- **Banda base.** La capa de banda base o nivel físico es la responsable de codificar y transmitir o recibir y decodificar los bits de datos RF.

Uno de los objetivos del proyecto, es implementar parte de las funciones de un nivel de red, como es el caso del enrutamiento en enlaces de varios saltos y la elección de la ruta óptima según la métrica elegida, que como se verá es la calidad del radioenlace.

4.2.4. Duty cycle.

Una limitación a tener en cuenta del módulo xBee 868 es el *duty cycle*. El *duty cycle* o ciclo de trabajo es la parte del tiempo total transcurrido, en el que el xBee ha estado transmitiendo. En el caso del xBee 868 existe una limitación de duty cycle del 10%, medido sobre un tiempo total de 1 hora. Es decir, que cada hora, se dispone de un tiempo neto de transmisión de 6 minutos. O lo que es lo mismo, el tiempo máximo de transmisión ininterrumpida es de 6 minutos.

Esto se debe a restricciones térmicas del módulo para prevenir sobrecalentamiento. En caso de superarse los 60 °C, la restricción de duty cycle será aún mayor, permitiéndose transmitir sólo un 10% medido en intervalos de 1 segundo, es decir, transmitir como máximo, 100 ms de manera ininterrumpida.

"Engañando" al dispositivo con secuencias de reseteo, se puede conseguir temporalmente obviar la limitación del 10% del tiempo medido en un intervalo de una hora, pero la segunda restricción del 10% medido sobre un segundo se acaba imponiendo, cuando el dispositivo va aumentando de temperatura y alcanza los 60 °C.

4.2.5. Modos de operación.

La interfaz serie del módulo xBee se puede configurar en dos modos de funcionamiento:

- **Modo transparente.**
- **Modo API.**

Grosso modo, se puede decir que el *modo transparente* simplemente manda datos en formato "RAW" por radiofrecuencia. Es decir, es como si entre dos puertos series, en lugar de haber un cable, hubiera un medio radioeléctrico.

Por otra parte, el *modo API* introduce una cabecera adicional a la trama (*Application Header*), en la que se incluye información extra que permite, entre otras cosas:

- Identificar al nodo de origen y destino del paquete.
- Fragmentar los paquetes e identificar los fragmentos que llegan y comprobar que llegan en orden.

En la figura 4.6 se puede ver la trama RF que se envía en modo API con las cabeceras desglosadas. De los cuatro diagramas que se ven, el último es la cabecera API, formada por el "Application ID", "Fragment number", etc.

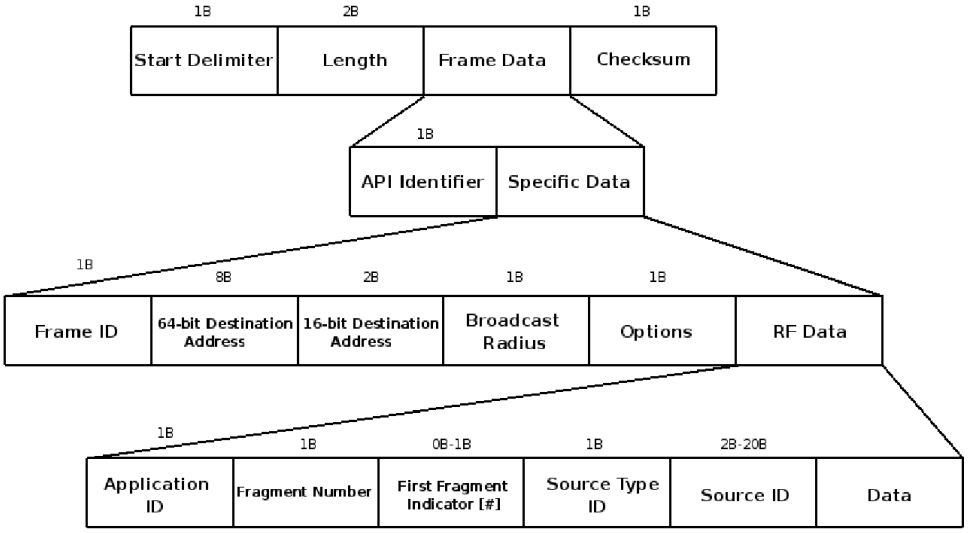

Figura 4.6: Estructura de la trama RF en modo API.

En la figura 4.7 se ve la cabecera API (Application Header) con los siguientes campos:

- **Application ID**. Es un ID de aplicación que sirve para identificar un determinado paquete entre varios en el receptor.
- **Fragment number**. El número de fragmento especifica la posición del fragmento en el paquete. El primer fragmento muestra el número total de fragmentos del paquete.
- **First Fragment Indicator [#]**. Es un campo opcional que se incluye en el primer fragmento para indicar que es el primer fragmento.
- **Source type ID**: Especifica el tipo de identificador de origen elegido. Las posibilidades son dirección MAC (64 bits), Identificador de red NA (16 bits) o Identificador de nodo NI (cadena de caracteres de 20 bytes máximo).
- **Source ID**. Especifica el identificador del nodo origen del paquete. Depende del campo *source type ID* elegido.
- **Data.** Guarda los datos enviados en cada fragmento o paquete entero (en caso de no existir fragmentación).

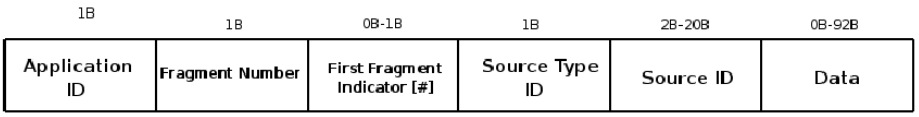

Figura 4.7: Cabecera API (Application Header).

Digi recomienda utilizar el modo API en caso de tener que enviar datos RF a múltiples destinos, o cuando un nodo recibe datos de diferentes orígenes, como es el caso de este proyecto, en el que un nodo puede enviar o recibir de distintos nodos.

Además, se ha optado por usar el modo API porque realiza tareas que facilitan la implementación de un nivel de red, como es el caso del direccionamiento o la fragmentación de paquetes. La única ventaja del modo transparente es que es más sencillo de implementar y

se transmiten menos datos. Sin embargo, como Libelium proporciona una API (conjunto de librerías) para programar de manera más cómoda el xBee, esta ventaja pierde valor.

Se ha optado en todo momento en usar el modo API, incluso en la transmisión entre el nodo de la Oficina hasta el Waspmote Gateway conectado al ordenador que ejecuta Waspmonitor puesto que el nodo de la Oficina se comunica con sus adyacentes en modo API, y al enviar cambiar para enviar tramas en modo transparente, se producían errores en la transmisión.

4.3. Sistema energético.

Waspmote tiene 4 modos de funcionamiento:

- **ON**: modo normal de funcionamiento. El consumo en este estado es **de 9mA.**
- **Sleep**: El programa principal se detiene, el microcontrolador pasa a un estado de latencia, del que puede ser despertado por **todas** las interrupciones asíncronas y por la interrupción síncrona generada por el Watchdog. El intervalo de duración de este estado va de **32ms a 8s**. El consumo en este estado es de **62µA**.
- **Deep Sleep**: El programa principal se detiene, el microcontrolador pasa a un estado de latencia del que puede ser despertado por **todas** las interrupciones asíncronas y por la interrupción síncrona lanzada por el RTC. El intervalo de este ciclo puede ir de **8 segundos a minutos, horas, días**. El consumo en este estado es de **62µA**.
- **Hibernate**: El programa principal se detiene, el microcontrolador y todos los módulos de Waspmote quedan completamente desconectados. La única forma de volver a activar el dispositivo es a través de la alarma previamente programada en el RTC (interrupción síncrona). El intervalo de este ciclo puede ir de **8 segundos a minutos, horas, días**. Al quedar el dispositivo totalmente desconectado de la batería principal el RTC es alimentado a través de una **batería auxiliar** de la que consume **0,7µA**.

Por otro lado, cada módulo tiene hasta 4 modos de funcionamiento:

- **ON**: modo normal de funcionamiento.
- **Sleep**: En este modo se detienen algunas funcionalidades del módulo y se pasa a un uso asíncrono, normalmente dirigido por eventos. En cada módulo funciona de una forma distinta y es específico de cada uno (programado por el fabricante).
- **Hibernate**: En este modo se detienen todas las funcionalidades del módulo y se pasa a un uso asíncrono, normalmente dirigido por eventos. En cada módulo funciona de una forma distinta y es específico de cada uno (programado por el fabricante).
- **OFF**: Mediante el uso de switches digitales controlados por el microcontrolador se apaga por completo el módulo. Este modo ha sido implementado por Libelium como capa independiente de control energético, de forma que se pueda reducir el consumo al mínimo (~0µA) sin relegar en las técnicas implementadas por el fabricante.

En el proyecto, siempre se ha utilizado el estado ON tanto para el Waspmote, como para los módulos usados puesto que se supone que los nodos han de estar siempre alimentados por la red eléctrica o paneles solares y no exclusivamente por baterías.

4.4. Timers.

Waspmote utiliza un oscilador de cuarzo que trabaja a una frecuencia de 8MHz como reloj del sistema. De esta forma, cada 125ns el microcontrolador ejecuta una instrucción de bajo nivel (lenguaje máquina). Hay que tener en cuenta que cada línea de código C++ de un programa compilado para Waspmote engloba varias de instrucciones de lenguaje máquina.

En el proyecto no se han utilizado los dos timers que proporciona Waspmote, debido a que su principal objetivo es generar interrupciones, a modo de alarma, para despertar al microcontrolador de un estado energético de bajo consumo. Como ya se dijo en el apartado anterior, no se usarán dichos estados energéticos. Sin embargo, se describen brevemente en este apartado puesto que son aspectos importantes del dispositivo si se desea alimentar sólo por baterías.

4.4.1. Watchdog.

El microcontrolador Atmega 1281 tiene un reloj Watchdog interno mejorado (Enhanced Watchdog Timer, WDT). El WDT se encarga de contar de forma precisa ciclos de reloj generados por un oscilador de 128KHz. El WDT genera una señal de interrupción cuando el contador alcanza el valor establecido.

El WDT permite despertar al microcontrolador del estado Sleep de bajo consumo generando una interrupción. Por ello, se utiliza este reloj a modo de alarma temporal asociado al modo Sleep del microcontrolador. Esto permite controlar intervalos de tiempo muy pequeños y de forma muy precisa, estos intervalos son: 16ms, 32ms, 64ms, 128ms, 256ms, 500ms, 1s, 2s, 4s, 8s. Para intervalos mayores a 8s (modo Deep Sleep) se usa el RTC.

4.4.2. RTC.

Waspmote dispone de un reloj en tiempo real (RTC) a 32KHz (32.768Hz) que permite establecer una base de tiempos absoluta para la utilización del dispositivo.

El RTC permite despertar al microcontrolador del estado de bajo consumo generando una interrupción. Por ello, se ha asociado al modo DeepSleep e Hibernate del microcontrolador, permitiendo poner a dormir el microcontrolador activando una alarma en el RTC para poder despertarlo. Los intervalos pueden ir desde los 8s en modo Deep Sleep, hasta minutos, horas o incluso días en Hibernate.

4.5. Interrupciones.

A instancias del microcontrolador, todos los tipos de interrupciones las recibe como interrupciones hardware. Para ello se usan los pines de la UART (RXD1 y TXD1).

Dichas interrupciones se utilizan normalmente para despertar al Waspmote de un estado de bajo consumo y ejecutar una rutina de interrupción, minimizando el consumo en el caso de funcionamiento sólo con baterías. Como en el caso del proyecto, no tiene sentido tener nodos que funcionen sólo con baterías, no se han programado de esta manera y por tanto, no se han utilizado interrupciones. No obstante, en este apartado se describen brevemente.

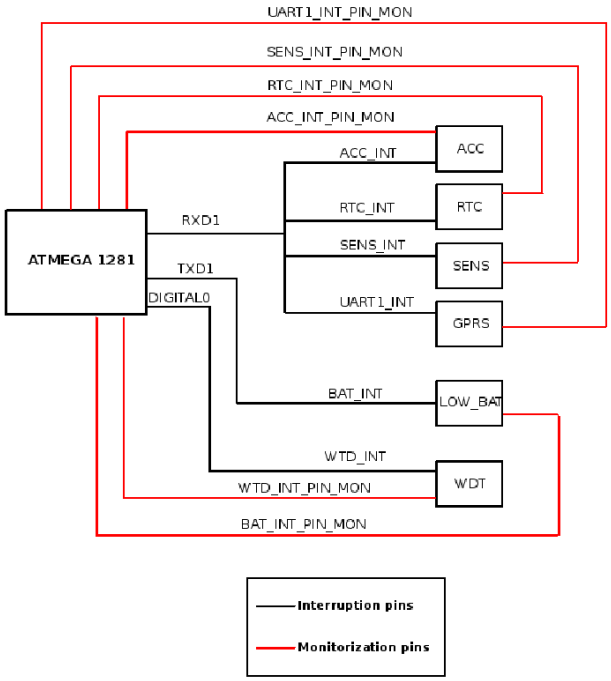

Figura 4.8: Esquema de conexiones de módulos al micro.

Waspmote dispone de 6 posibles señales de interrupción: Watchdog, RTC, acelerómetro, GPRS, sensores y batería crítica.

Las señales de interrupción procedentes del RTC, acelerómetro, GPRS y sensores se multiplexan en el pin RXD1. Debido a esta multiplexación de varias señales de interrupción se han creado una serie de vectores de flags para permitir conocer el módulo que ha generado la interrupción para poder llamar a la rutina de atención a la interrupción correspondiente.

4.6. Waspmote Gateway.

El Waspmote Gateway es, como su nombre indica en inglés, una puerta de acceso o puente de datos entre la red de sensores y el equipo receptor en el que se monitorizarán y almacenarán los datos en archivos de log. En el caso del proyecto, el equipo receptor será un ordenador portátil y el programa de monitorización será Waspmonitor.

Figura 4.9: Waspmote Gateway con antena sleeve dipole de 0 dBi.

Este dispositivo básicamente consta de los siguientes elementos:

- **xBee 868** conectado en un zócalo xBee similar al existente en el Waspmote.
- **Conversor FTDI** que se encarga de emular un puerto serie virtual mediante drivers apropiados para el SO (Windows, en nuestro caso) y hacer la conversión RS232 - USB.
- **Leds** de alimentación RX, TX y uno configurable.
- **Pulsadores de Reset**, I/O 0, I/O 1 y RTS, conectados a los correspondientes pines del xBee 868.

Este dispositivo, al igual que el Waspmote, puede ser utilizado con otros xBee que usen otra tecnología (como ZigBee), simplemente conectándolo en el zócalo.

4.7. API de Waspmote.

Libelium ha desarrollado una API (Application Programming Interface) para facilitar la programación de aplicaciones utilizando Waspmote.

Esta API engloba todos los módulos que se integran en Waspmote, así como el manejo de otras funcionalidades como las interrupciones o los diferentes estados energéticos. La API se ha desarrollado en C/C++, estructurándola de la siguiente manera:

- **Configuración general:** WaspClasses.h, WaspVariables.h, WaspConstants.h, Wconstants.h, pins_arduino.h, pins_arduino.c, utils.h, utils.cpp, WProgram.h, Wrandom.cpp
- **Comunes:** binary.h, byteordering.h, byteordering.cpp, HardwareSerial.h, HardwareSerial.cpp, WaspRegisters.h, WaspRegisters.c, wiring_analog.c, wiring.h, wiring.c, wiring_digital.c, wiring_private.h, wiring_pulse.c, wiring_serial.c, wiring_shift.c

- **Almacenamiento SD:** fat_config.h, fat.h, fat.cpp, partition_config.h, partition.h, partition.cpp, sd_raw_config.h, sd_raw.h, sd_raw.cpp, sd_reader_config.h, WaspSD.h, WaspSD.cpp
- **Comunicación I2C:** twi.h, twi.c, wire.h, wire.cpp
- **Acelerómetro:** WaspACC.h, WaspACC.cpp
- **GSM/GPRS:** WaspGPRS.h, WaspGPRS.cpp, WaspGPRSConstants.h
- **GPS:** WaspGPS.h, WaspGPS.cpp
- **Control energético:** WaspPWR.h, WaspPWR.cpp
- **RTC:** WaspRTC.h, WaspRTC.cpp
- **Sensores:** WaspSensorEvent.h, WaspSensorEvent.cpp, WaspSensorGas.h, WaspSensorGas.cpp, WaspSensorPrototyping.h, WaspSensorPrototyping.cpp
- **USB:** WaspUSB.h, WaspUSB.cpp
- **XBee:** WaspXBee.h, WaspXBee.cpp, WaspXBeeConstants.h, WaspXBeeCore.h, WaspXBeeCore.cpp, WaspXBee802.h, WaspXBee802.cpp, WaspXBeeZB.h, WaspXBeeZB.cpp, WaspXBeeDM.h, WaspXBeeDM.cpp, WaspXBee868.h, WaspXBee868.cpp, WaspXBeeXSC.h, WaspXBeeXSC.cpp
- **Interrupciones:** Winterrupts.c

De todas estas librerías, las que se han usado en el proyecto son las de configuración general, comunes, comunicación con I2C, RTC y XBee. Cabe destacar que en la librería XBee se definen las funciones necesarias para poder establecer, controlar y utilizar una red de módulos xBee, siendo WaspXBee868 donde se definen las funciones específicas del xBee 868 y se heredan las características generales de XBee.

4.7.1. Estructura packetXBee.

La estructura 'packetXBee' está definida en la API en el archivo de cabecera 'WaspXbeeCore.h' y es de gran importancia en el proyecto, puesto que se utiliza para mandar los paquetes en modo API.

Entre los muchos campos que componen la estructura, destacamos los que se han usado en el proyecto (muchos se rellenan de forma transparente por la API):

- **macDL y macDH**: Dirección MAC de 64 bits de destino (L = 32 bits inferiores, H = 32 bits superiores).
- **macSL y macSH**: Dirección MAC de 64 bits de origen (L = 32 bits inferiores, H = 32 bits superiores). Con origen se refiere al nodo del que proviene el mensaje (teniendo en cuenta un solo salto).
- **macOL y mac OH**: Dirección MAC de 64 bits de origen absoluto (L = 32 bits inferiores, H = 32 bits superiores). Con origen absoluto se refiere al nodo del que proviene el mensaje (teniendo en cuenta varios saltos).
- **mode**: El modo de transmisión elegido para transmitir el paquete. Puede ser 0 (Unicast) o 1 (Broadcast).
- **address_type**: 0 (dirección de 16 bits) o 1 (dirección de 64 bits).

- **data**: Datos enviados en el paquete. Se usan para guardar los datos a enviar en transmisiones unicast y broadcast a otros nodos. La API se encarga de tener cuidado al enviar los datos y fragmentar los paquetes si se excede el payload. Su tamaño máximo viene dado por la constante MAX_DATA, definida en la API.
- **data_length**: La longitud real del campo '*data*'. El campo '*data*' es un array de un tamaño máximo definido, pero se rellena sólo una parte, por tanto cada paquete puede tener un tamaño distinto.
- **RSSI**: Indicador de Nivel de Señal Recibida. Especifica en dBm el RSSI del último paquete recibido via RF. Si es un paquete fragmentado, contiene la media de las RSSI indicadas en todos los fragmentos.
- **retries**: Número de reintentos necesarios para enviar el paquete.

4.7.2. Carga máxima de una trama RF (Payload).

La carga o *payload* máxima de una trama RF viene dada en la siguiente tabla:

	Unicast	**Broadcast**
Con encriptación	100 Bytes	100 Bytes
Sin encriptación	100 Bytes	100 Bytes

Tabla 4.2: Cargas máximas en bytes de las tramas RF.

Como se puede observar en la tabla 4.2, la carga máxima es de 100 Bytes para todos los casos (tanto encriptado y no encriptado, como unicast y broadcast), eso implica, que el tamaño máximo del campo de datos de un paquete con cabecera API, será de 68 Bytes en el primer fragmento, y de 69 Bytes en el resto (si restamos a los 100 Bytes las cabeceras API).

Por lo tanto, si un mensaje a encapsular dentro de un paquete API, tiene más de 68 bytes, será necesaria fragmentación, ya que el total de cabecera API + datos no puede superar los 100 Bytes.