UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

DEPARTAMENTO DE TECNOLOGIA ELECTRÓNICA

PROYECTO FIN DE CARRERA

INGENIERO TÉCNICO INDUSTRIAL
ESPECIALIDAD EN ELECTRÓNICA INDUSTRIAL

"SISTEMA DE MONITORIZACION MEDIANTE REDES DE SENSORES INALÁMBRICAS, BASADO EN EL KIT DE DESARROLLO ZIGBEE AMP DE MESHNETICS"

AUTOR: **Israel García Meroño**

DIRECTORES: **D. Juan Suardíaz Muro**
D. Juan A. López Riquelme

Cartagena, Octubre de 2011

A mis directores de proyecto Juan Suardíaz Muro y Juan Antonio López Riquelme por confiar en mí y brindarme la oportunidad de trabajar junto a ellos.

A mis padres y demás familiares por su apoyo y confianza durante mis años de carrera.

A mis amigos, por todo lo que me han aportado durante todos estos años de carrera.

Y, muy especialmente, a mi hermano Andrés, por su inestimable apoyo y cariño.

ÍNDICE

CAPÍTULO 1

INTRODUCCION

En este capítulo de introducción, se expondrán los objetivos del proyecto así como las fases en las que éste ha sido dividido para su elaboración. Respecto a los objetivos, se describirá con detalle todo aquello que se pretende desarrollar y se justificará el porqué de la elaboración en general de este proyecto. Se analizará la complejidad que puede entrañar así como lo que se espera conseguir a su finalización.

En cuanto a las fases, se describirán en cada una de ellas de manera clara y concisa los distintos aspectos que serán objeto de estudio y desarrollo, de manera que sirva tanto al autor en la realización del proyecto, como a cualquier persona que lea esta memoria, como guión de cómo se pretende llevar a cabo todo el proceso de realización del proyecto.

1.1 OBJETIVOS DEL PROYECTO.

El proyecto que se pretende desarrollar tiene por objetivo el estudio y desarrollo de dos aplicaciones en lenguaje de programación Java. Una para la gestión de los mensajes recibidos por el coordinador de una red de sensores inalámbricos (siglas en inglés WSN) y otra para la monitorización y gestión de los datos capturados.

Las redes de sensores inalámbricas constituyen una tecnología emergente con gran grado de aplicación en diferentes sectores industriales. Este tipo de tecnología presentan la principal ventaja de eliminar la necesidad de cableado en tareas de monitorización y actuación. Son especialmente útiles en sectores industriales como la fabricación o las aplicaciones agronómicas, en las que se evitan la instalación de buses de campo cableados que pueden originar problemas asociados al deterioro o daño en las tareas normales de funcionamiento, como por ejemplo el seccionamiento por el paso de una maquinaria pesada o tractor sobre el cable.

En este proyecto se pretende utilizar un kit de desarrollo de esta tecnología para arrancar y diseñar un sistema de monitorización basado en el protocolo de comunicaciones ZigBee. Se trata de un protocolo específicamente diseñado para tareas de monitorización de bajo consumo, factor crucial en entornos autónomos, en los que el incremento en la autonomía de la batería es un punto decisivo.

Para llevar a cabo este estudio, se utiliza el kit de desarrollo inalámbrico “ZigBit Amp”, comercializado por la empresa Meshnetics. Se explica detalladamente su contenido hardware (tipos de nodos, tipos de sensores que ofrece, características técnicas…) y todo el conjunto software que interviene en el funcionamiento del mismo.

Se utilizará en este proyecto un simple potenciómetro simulando un sensor, el cual estará conectado a un pin ADC de la ranura de expansión de una tarjeta del kit. Esta tarjeta pertenece a una red de sensores inalámbrica gobernada por el protocolo ZigBee. Para el almacenamiento y gestión de datos se usará el servidor MySQL, el cual proporcionará múltiples ventajas a la hora de obtener los datos para poder representarlos gráficamente o incluso la posibilidad de obtener los datos recibidos a través de internet para ser usados por la aplicación desarrollada.

1.2 Fases del proyecto.

Para el correcto desarrollo del proyecto se han establecido las siguientes fases que se exponen a continuación:

1. Lectura de información sobre redes de sensores
2. Aprendizaje del uso del kit de desarrollo
3. Estado del arte de los kits de desarrollo basados en ZigBee
4. Descripción de la arquitectura hardware
5. Descripción de la arquitectura software
6. Desarrollo de la electrónica necesaria para la conexión de nuevos sensores
7. Programación de los módulos correspondientes a la adquisición de datos provenientes de los sensores.
8. Programación del interfaz de monitorización y gestión de datos recibidos.
9. Verificación sobre aplicación específica.
10. Modificaciones y correcciones.

1.3 3.Desarrollo de la memoria.

La memoria de este proyecto se divide en ocho capítulos, en los que se describe por partes todo lo comentado en el apartado anterior. A continuación se muestra un pequeño avance, desarrollado posteriormente en cada uno ellos.

- Capítulo 1. Introducción.
  En este capítulo se presenta de manera general de que trata este proyecto, indicando brevemente las fases en las que está dividido.
- Capítulo 2. Estado del arte.
  Este capítulo tratará todo lo relacionado con el desarrollo actual sobre redes de sensores inalámbricas. Se tratará con más detalle el protocolo de comunicación ZigBee. Se seguirá con una breve explicación de los sistemas operativos más usados en este tipo de redes y por último se finalizará con un estudio sobre la variedad de kits de desarrollo.
- Capitulo 3. Descripción del sistema.
  En este capítulo se hará hincapié en el kit usado en este proyecto, donde se hablará de la empresa desarrolladora, los productos variados que ofrece y un extenso estudio de las tarjetas a usar. Se comentará la manera de usar el software que incluye el kit para la gestión de la red ZigBee y se finalizará con un breve análisis del programador a usar.
- Capítulo 4. Aprendizaje del uso del kit de desarrollo.
  En este capítulo se mostrará un pequeño tutorial del uso del kit de desarrollo ZigBit de la empresa MeshNetics. Se comenzará por una breve introducción al uso del entorno de programación de ATMEL llamado AVR STUDIO. Seguidamente se mostrará las maneras útil de programar las tarjetas y el software implementado en ellas. Además, se mostrará la manera de modificar el código para añadir nuevos sensores a la red ZIgBee.
- Capítulo 5. Aplicación WSN.
  En este capítulo se presentará la aplicación en java WSN. Se empezará por una pequeña introducción de la funcionalidad del programa. Se continuará por una breve explicación de la instalación de la herramienta de programación Java y por último se expondrá el diseño final de la aplicación y su funcionamiento.
- Capítulo 6. Base de datos.
  Este capítulo trata todo lo relacionado con la base de datos. Su instalación, la forma de gestionar el programa a usar y un ejemplo para sacarle mayor partido al programa.

- Capítulo 7. Aplicación WSNGui4.
  En este capítulo se expondrá la aplicación principal de este proyecto WSNGui4, desarrollada especialmente para satisfacer la necesidad de elaborar un programa para visualizar en modo de gráficas los datos obtenidos por los sensores de las tarjetas.
- Capítulo 8. Conclusiones y trabajos futuros.
  En este último capítulo se presentarán las conclusiones más importantes que se han obtenido a lo largo de la realización del presente proyecto fin de carrera. Para concluir este capítulo se detallarán una serie de posibles líneas de trabajo futuras, que son posibles con la culminación del presente trabajo.

CAPÍTULO 2

ESTADO DEL ARTE

Este capítulo tratará todo lo relacionado con el desarrollo actual sobre redes de sensores inalámbricas (en inglés Wireless Sensor Networks, WSN). Se hará una pequeña introducción al significado en cuestión, pasando por su topología, su arquitectura y los tipos de protocolos más significativos para una red WSN. A continuación se tratará con más detalle el protocolo de comunicación ZigBee pues será el que esté implementado en las tarjetas de comunicación. Se seguirá con una breve explicación de los sistemas operativos mas usados en este tipo de redes y por último se finalizará con un estudio sobre la variedad de kits de desarrollo,

2.1 Introducción a las redes de sensores inalámbricas.

Una red de sensores inalámbrica (*Wireless Sensor Network, WSN*) es una red formada por un conjunto de nodos, denominados "motes", capaces de comunicarse entre sí y, a su vez conectados a una base central de recogida de datos y de información procedente de los sensores, con el fin de supervisar características físicas o condiciones medioambientales en lugares deseados.

Las WSN comenzaron siendo utilizadas en aplicaciones militares. En la actualidad se usan en la mayoría de espacios donde se necesita un control inalámbrico mediante sensores: domótica, medicina o en fábricas industriales, entre otros. El diseño eficiente e implementación de las redes inalámbricas de sensores se ha convertido en un área de investigación emergente en los últimos años, debido al gran potencial de estas redes para cubrir grandes áreas a bajo costo.

El concepto "mote", viene del proyecto *Smart Dust* (polvo inteligente) propuesto por científicos de la Universidad de California, en el cual, se pretende conseguir una red de sensores inalámbricos donde los nodos sean de un tamaño minúsculo, parecido al de una "mota de polvo", consiguiendo redes inalámbricas muy potentes, ocupando un espacio mínimo. El tamaño de un nodo puede variar, al igual que el coste, desde cientos de euros a unos pocos céntimos, dependiendo del tamaño de la red y de la complejidad computacional requerida por los distintos nodos sensores. El tamaño y la limitación del consumo en los nodos sensores son los causantes de las limitaciones de recursos como la energía (<50mw), la memoria (<1Mbyte), la velocidad de cálculo (<500 Mhz) y el ancho de banda (250kbps). Las limitaciones de WSN pueden discutirse en términos de potencia, que se considera como el factor decisivo en el despliegue del sensor.

Los sensores inalámbricos pueden usarse para tener varias capacidades de detección si existen diferentes sensores dentro de la misma red. Es posible disponer de varios tipos de datos al mismo tiempo, los cuales se transmiten hasta la estación base de la red. Además, al tener un microcontrolador interno dentro de estos motes, se pueden realizar pequeñas procesamientos de los datos con el objetivo de reducir el tráfico circulante por la red; puesto que sólo se enviarán los datos o las informaciones estrictamente necesarias.

Las redes de sensores inalámbricas son el futuro de todas las redes de sensores existentes, puesto que la no utilización de cables, unidas a la mejora de las comunicaciones

inalámbricas en la sociedad y a la disminución progresiva en tamaño de los componentes electrónicos, posibilitan la creación de redes más potentes y pequeñas con el paso del tiempo. Por tanto, en general, es posible decir que los elementos que forman la WSN son:

- Sensores, de todos los tipos y con el fin de medir las condiciones y la información deseada del lugar o del medio en el que se encuentran.
- Motes, los cuales consisten en dispositivos capaces de recoger la información recibida por los sensores y transmitirla a otros motes o a la estación base.
- Pasarelas o Gateways, que constituyen los elementos que conectan los motes de la red de sensores.
- Estación base, consistente en un elemento que recoge todos los datos. Suele tratarse de un ordenador común o un elemento que lo sustituya.

2.1.1 Características generales de los nodos sensores.

Con lo comentado hasta ahora, se pueden establecer una serie de características que se dan en los nodos:

- Integran sensores de distintos tipos según el fenómeno a analizar, para realizar mediciones.
- Los sensores son de bajo coste.
- Están limitados en diferentes aspectos.
    - Energía: Normalmente están alimentados por medio de baterías y el bajo consumo es una de sus prioridades.
    - Memoria: La capacidad de almacenamiento es también limitada.
- Hacen un uso intensivo de la CPU para el procesamiento y de la Radio para enviar y recibir mensajes.
- Alta probabilidad de fallo, debido a las condiciones a las que se exponen y a su bajo coste.
- Son autónomos y operan de forma independiente.
- Se adaptan al entorno.

2.1.2 Componentes básicos de un nodo sensor

- Unidad sensora: Sensores y conversores analógico-digital (A/D) que convierten las señales analógicas en digitales para el microcontrolador.
  Los convertidores A/D adaptan la señal para su posterior utilización, mediante un proceso que suele constar de tres etapas: muestreo, cuantificación y codificación.

- Microcontrolador: Gestiona los procesos que permiten al nodo sensor colaborar con otros para realizar las tareas asignadas. Normalmente el procesador está asociado a una pequeña unidad de almacenamiento.
  Aunque cada vez hay procesadores más pequeños y rápidos, las unidades de procesamiento y almacenamiento en los nodos son recursos escasos.

- Transceptor: Conecta el nodo a la red, realizando las operaciones de transmisión y recepción de mensajes. Los transceptores pueden ser dispositivos ópticos activos o pasivos o dispositivos de radiofrecuencia. Radiofrecuencia (RF) requiere modulación, filtro paso banda, filtrado, demodulación y circuito multiplexor, lo que los hace complejos y caros. Aún así se prefieren transceptores RF en la mayoría de los proyectos porque los paquetes transportados son pequeños y las tasas de transferencia también.

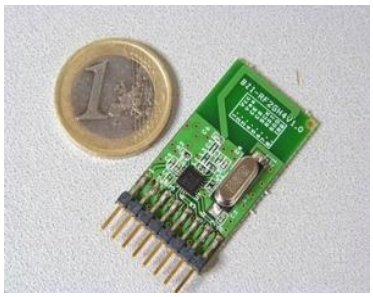


Fig 2.1 *Transceptor RF*

.

- Alimentación: Se obtiene a partir de las baterías, aunque puede estar ayudado de un generador. Es uno de los componentes más importantes, ya que el tiempo de vida útil del "mote" depende de éste elemento. El dispositivo final debe caber en un módulo del tamaño de una caja de cerillas y, a veces, en hasta de tamaño mucho más pequeño (un centímetro cúbico) para poder suspenderlo en el aire.

2.1.3 Clasificación de los sensores

Atendiendo al fundamento físico: Es decir, según la propiedad física cuya variación produce excitación:

- Resistivos: Miden variaciones de la resistencia.
- Capacitivos: Miden variaciones de la capacidad.
- Inductivos: Miden variaciones de la inductancia electromagnética o magnitud de flujo magnético.
- Generadores de tensión o intensidad: La magnitud física provoca la generación de tensión o intensidad en el dispositivo, sin necesidad de alimentación externa.

Atendiendo a la alimentación.

- Activos: Ellos mismos generan una tensión o corriente, no requiriendo, por tanto, una alimentación externa. Se basan en diferentes efectos: termoeléctrico, piezoeléctrico…
- Pasivos: Requieren de una alimentación o excitación externa para generar una señal.

Atendiendo a la salida.

- Analógicos: La salida del transductor es un nivel de tensión o intensidad que varía de forma continua con la variable a medir dentro del rango de medida del transductor. Suelen emplearse valores normalizados a 0-10V y 4-20mA.
- Digitales: La salida está codificada mediante un código binario o en forma de pulsos. Son codificaciones habituales la binaria, Gray o el BCD.
- Todo-Nada: Se consideran un caso particular de los digitales. La salida sólo presenta dos estados: activa o no activa. Es el caso de dispositivos tales como los detectores de presencia.

Atendiendo a la magnitud física a medir: posición, velocidad, aceleración, temperatura, fuerza, nivel, presión, etc.
Por otro lado, podemos establecer una serie de factores que evalúan y catalogan también los tipos de nodos sensores. Estos factores serían: energía, flexibilidad, robustez, seguridad, comunicación, computación, sincronización, tamaño y coste.

2.1.4 Arquitectura de una WSN.

Tomando como elementos de una red los nombrados anteriormente, podemos distinguir varios tipos de arquitecturas:

- Arquitectura Centralizada: En este tipo de red (fig. 2.1) los nodos que se encargan de recoger ciertas informaciones, tras lo que enviarán sus datos a la pasarela más cercana que dirigirá el tráfico de la red. Esto provocará dos problemas, como son el cuello de botella, provocando la lentitud de los envíos de los datos y como consecuencia un mayor consumo de energía, disminuyendo el tiempo de vida de los nodos.

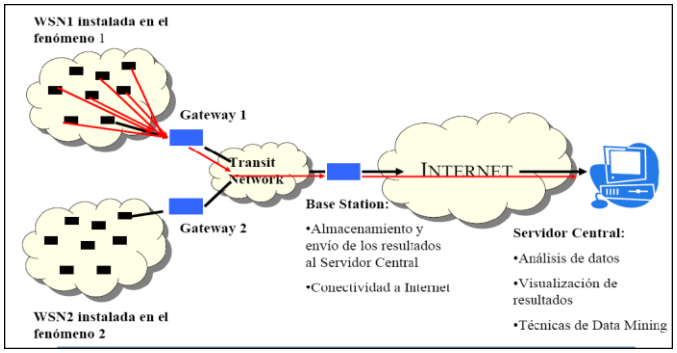


Fig 2.2 *Esquema de arquitectura centralizada*

- Arquitectura Distribuida: Los nodos se comunicarán entre sus nodos vecinos, cooperando y obteniendo una respuesta única que será enviada al "*cluster head*" que se encargará de comunicar a la estación base. Este tipo de arquitectura (fig. 2.2) evita los problemas de la arquitectura centralizada.

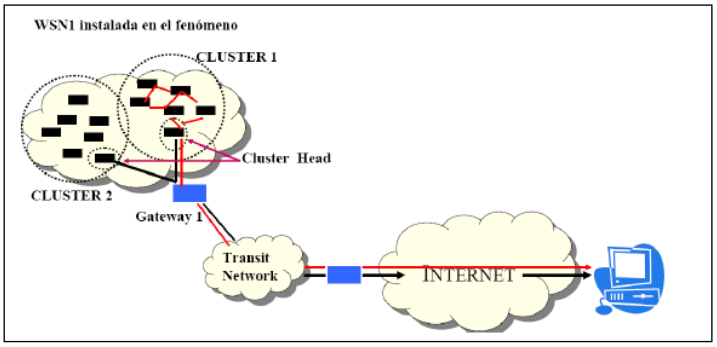


Fig 2.3 *Esquema de arquitectura distribuida*

2.1.5 Topología de red.

Las redes que se construyen dentro del estándar IEEE 802.15.4 deben de autorganizarse y automantenerse para que de esta forma se reduzcan los costes totales y se facilite su uso. Dependiendo de los requerimientos de la aplicación un red bajo el estándar IEEE 802.15.4 puede operar en cualquiera de estas configuraciones: red en estrella y *peer-to-peer*.

En la configuración en estrella, es necesario un coordinador de PAN hacia el cual los demás módulos dirigen su comunicación, y que es el encargado de iniciar, terminar o redirigir la comunicación a través de la red. Aparte de esto este nodo realiza la propia función que tenga asignada. Por ejemplo, lectura de sensores, actuar como nodo central de recogida de datos, etc. Es posible ver gráficamente como se distribuirían los módulos en estas redes en la ***Fig 2.4***

Topología en estrella y red peer-to-peer

.

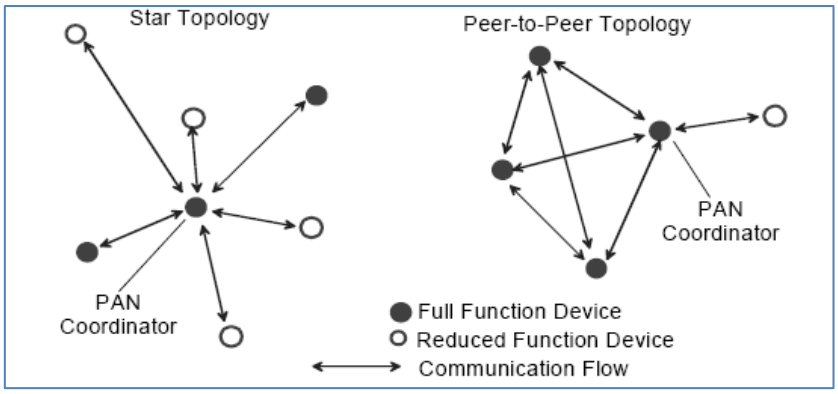


Fig 2.4 *Topología en estrella y red peer-to-peer*

En la topología de red *peer-to-peer*, también existe un coordinador de red, que se comunica con todos los módulos, pero además cada módulo puede comunicarse con otros de la misma red que estén a su alcance. Con esta topología de red es posible crear estructuras más complejas como redes enmalladas, se muestra un ejemplo de estas redes en la 4. Estas redes pueden ser ad-hoc, es decir redes sin infraestructura física preestablecida, ni necesidad de un control central que coordine su actividad, además pueden tener la propiedad de ser autorganizativas con restauración de caminos, sus sensores pueden trabajar como emisores o receptores y pueden establecer caminos de comunicación entre nodos sin visibilidad directa y modificar estos caminos si alguno de los nodos del encaminamiento falla, etc. pero estas funciones han de ser establecidas en una capa superior, la cual no forma parte de este estándar.

2.2 Protocolos de una WSN.

2.2.1 Wi-Fi.

El protocolo Wi-Fi es similar a la red Ethernet tradicional y por tanto se necesita una configuración previa para establecer la comunicación. Muchas veces, se le denomina Wi-Fi al “Ethernet sin cables”, para dar una gran idea de las ventajas e inconvenientes que tiene respesto a otras alternativas. Además Wi-Fi permite conexiones mucho más rápidas y rangos de distancias mayores, aparte de mejores mecanismos de seguridad.
Los dispositivos Wi-Fi permiten una gran movilidad, que es uno de las ventajas de esta tecnología. Por el contrario, es de muy fácil acceso para personas ajenas si la red no está bien configurada o bien protegida, siendo necesaria la seguridad para evitar este último problema.
Utiliza el mismo espectro de frecuencia que Bluetooth, con una potencia de salida mayor que lleva a conexiones más sólidas. Cabe aclarar que esta tecnología no es compatible con otros tipos de conexiones sin cables como Bluetooth, GPRS, UMTS, etc.
Existen varios tipos de dispositivos que se pueden clasificar en dos grupos:

- Dispositivos de distribución o red, entre los que cabe destacar los routers, repetidores o puntos de acceso.
- Dispositivos terminales, que suelen ser las tarjetas receptoras empleadas en la comunicación con el ordenador. Pueden ser las tarjetas PCI o bien tratarse de USB externos.

2.2.2 Bluetooth.

El protocolo Bluetooth fue diseñado con el objetivo de reemplazar la tecnología con cables, usando una conexión de radio muy seguro y de corto alcance. Permite la conexión entre dispositivos a un bajo costo y a pequeñas distancias. El alcance es generalmente de 10 metros, puesto que se hace para establecer una conexión utilizando el mínimo consumo de energía de las baterías. A pesar de esto, se puede conseguir un alcance mucho mayor, similar al Wi-Fi, con el inconveniente del aumento de consumo. También es recomendable que no exista nada físico entre los elementos que se conectan mediante este protocolo para mejorar la comunicación. La banda de radiofrecuencia utilizada sería la ISM (Industrial, Scientific and Medical) de los 2,4GHz.

Es posible diferenciar varios tipos de clases de dispositivos, “Clase 1” para una potencia máxima de 100 mW o 20 dBm y cuyo rango sería de aproximadamente unos 100 metros. “Clase 2” serían aquellos dispositivos de 2,5 mW o 4 dBm de potencia y con un rango de unos 20 m. Finalmente la “Clase 3” son los de mínimo consumo (1mW) y mínimo alcance (1m aproximadamente). También podemos diferenciar los dispositivos por su ancho de banda, donde es posible encontrar la Versión 1.2 de 1Mbits/s, Versión 2.0+EDR de 3Mbits/s y la versión 3.0+HS de 24 Mbits/s. La tecnología Bluetooth se utiliza en productos como módems, teléfonos móviles, auriculares, control remoto como el de los mandos de algunas videoconsolas, etc…

2.2.3 IEEE 802.15.4.

IEEE 802.15.4 es un estándar que define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal con tasas bajas de transmisión de datos (LR-WPAN). También es la base sobre la que se define la especificación de ZigBee, cuyo propósito es ofrecer una solución completa para este tipo de redes construyendo los niveles superiores de la pila de protocolos que el estándar no cubre.

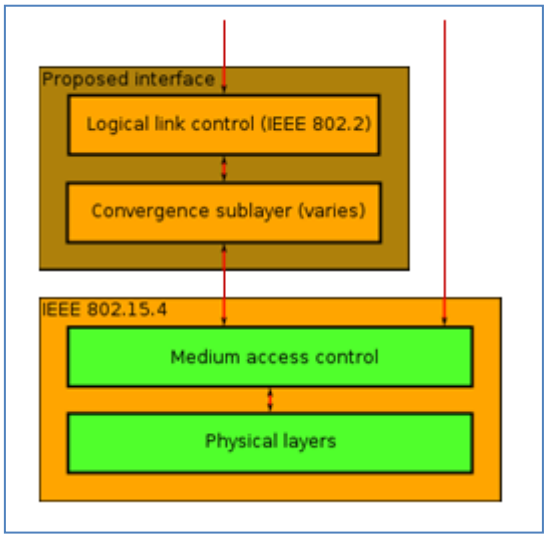


Fig 2.5 *Pila de protocolo IEEE 802.15.4*

El propósito del estándar es definir los niveles de red básicos para dar servicio a un tipo específico de red inalámbrica de área personal (WPAN) centrada en la habilitación de comunicación entre dispositivos ubicuos con bajo coste y velocidad (en contraste con esfuerzos más orientados directamente a los usuarios medios, como WiFi). Se enfatiza el bajo coste de comunicación con nodos cercanos y sin infraestructura o con muy poca, para favorecer aún más el bajo consumo.

En su forma básica se concibe un área de comunicación de 10 metros con una tasa de transferencia de 250 kbps. Se pueden lograr tasas aún menores con la consiguiente reducción de consumo de energía. Como se ha indicado, la característica fundamental de 802.15.4 entre las WPAN's es la obtención de costes de fabricación excepcionalmente bajos por medio de la sencillez tecnológica, sin perjuicio de la generalidad o la adaptabilidad.
Entre los aspectos más importantes se encuentra la adecuación de su uso para tiempo real por medio de slots de tiempo garantizados, evitación de colisiones por CSMA/CA y soporte integrado a las comunicaciones seguras. También se incluyen funciones de control del consumo de energía como calidad del enlace y detección de energía. Un dispositivo que utilice el 802.15.4 puede transmitir en las siguientes bandas de frecuencia:

- En Europa entre 868 y 868.8 MHz, permitiendo hasta tres canales.
- En América del Norte entre 902 y 928 MHz, extendido a treinta canales.
- 2400-2483.5 MHz de uso para todo el mundo, con la posibilidad de hasta dieciséis canales.

El control de acceso al medio (MAC, Medium Access Control) transmite tramas MAC usando para ello el canal físico. Además del servicio de datos, ofrece un interfaz de control y regula el acceso al canal físico y al balizado de la red. También controla la validación de las tramas y las asociaciones entre nodos y garantiza slots de tiempo. Por último, ofrece puntos de enganche para servicios seguros.

El estándar no define niveles superiores ni subcapas de interoperabilidad. Existen extensiones, como la especificación de ZigBee, que complementan al estándar en la propuesta de soluciones completas.

2.3 ZigBee.

2.3.1 Introducción.

Finalmente se encuentra el protocolo de comunicación ZigBee que se usará en este proyecto. Como se sabe, ZigBee es un conjunto de reglas que pueden ser implementadas por cualquiera que lo desee. Aunque hoy día existen una gran cantidad de estándares para las comunicaciones inalámbricas, que permiten grandes tasas de transferencia (como Wi-Fi), zigbee no aporta más velocidad (al contrario), sino bajo consumo energético. Se lo compara con bluetooth en muchos sentidos (cuestión que abordaremos más adelante), pero Zigbee ha sido diseñado para cubrir un nicho que aún está por explotar, como es el de los sensores, industriales o de cualquier tipo, domótica, donde tendrá una de sus mayores aplicaciones, e incluso según sus diseñadores, juguetería, sensores médicos y un sinfín de aplicaciones más, en las que no prima tanto la velocidad, sino la duración de la batería.
Antes de entrar a describir Zigbee, es conveniente hacer una pequeña introducción al estándar 802.15, que es en el que se enmarca esta tecnología inalámbrica.
Tanto los sensores como los actuadores u otros dispositivos pequeños de medida o control no requieren un gran ancho de banda, pero si un mínimo consumo energético y una baja latencia. ZigBee es idóneo para la comunicación de estos dispositivos.

Se define ZigBee como una pila de protocolos que permite la comunicación de forma sencilla entre múltiples dispositivos. Especifica diversas capas, adecuándose al modelo OSI. Las capas básicas, física (PHY) y de control de acceso al medio (MAC) están definidas por el estándar IEEE 802.15.4, LR-WPAN (Low Rate – Wireless Personal Area Network). Este estándar fue diseñado pensando en la sencillez de la implementación y el bajo consumo, sin perder potencia ni posibilidades.
El estándar ZigBee amplía el estándar IEEE 802.15.4 aportando una capa de red (NWK) que gestiona las tareas de enrutado y de mantenimiento de los nodos de la red; y un entorno de aplicación que proporciona una subcapa de aplicación (APS) que establece una interfaz para la capa de red, y los objetos de los dispositivos tanto de ZigBee como del diseñador (aunque parece difícil, es bastante sencillo, sigue leyendo).

Así pues, los estándares IEEE 802.15.4 y ZigBee se complementan proporcionando una pila completa de protocolos que permiten la comunicaciones entre multitud de dispositivos de una forma eficiente y sencilla.

2.3.2 Características significativas de ZigBee.

- Direccionamiento a nivel de red (16 bits)
- Soporte para enrutamiento de paquetes
- Permite topología de malla, gracias a las posibilidades de enrutamiento
- Dispositivos FFD (coordinador, router y dispositivo final) y RFD (dispositivo final)

2.3.3 Requisitos hardware.

ZigBee es un estándar que requiere una implementación para poder funcionar. Esta puede hacerse por software en multitud de arquitecturas. Sin embargo, independientemente de donde se implemente, necesita unos recursos mínimos. Ya que los dispositivos pueden efectuar diversos roles, los requisitos también varían de unos a otros.

- Un microcontrolador de 8 bits
- Pila completa, menos de 32 KiB
- Pila sencilla, 6 KiB aprox.

En cuanto a memoria RAM, cada implementación necesita una cantidad diferente, en función del grado de optimización de la misma, pero es de interés notar que los coordinadores y/o routers tendrán más exigencias puesto que necesitan mantener tablas para los dispositivos de la red, enlazado, etc.

2.3.4 Topologías de red y dispositivos.

Ahondemos un poco más en las posibilidades a la hora de crear redes IEEE 802.15.4/ZigBee. El estándar de la IEEE especifica dos tipos de dispositivos: de función reducida (RFD, Reduced Function Device) y de función completa (FFD, Full Function Device), diseñados para propósitos distintos.

El RFD esta pensado para aplicaciones muy sencillas, como interruptores de iluminación y sensores infrarrojos, que no necesitan enviar o recibir grandes cantidades de información. Solo puede comunicarse con dispositivos FFD. Todo esto permite que pueda ser implementado usando los mínimos recursos posibles, así como un ahorro energético visible. En cambio, los FFD pueden actuar como coordinadores o como dispositivos finales. Pueden comunicarse con otros FFD y RFD. Para ello necesitan más recursos, han de implementar la pila completa y precisan un consumo más exigente.

ZigBee aprovecha esta diferenciación. Además del coordinador de la red, es posible la existencia de routers, evidentemente han de ser FFD, que aumentan las posibles topologías de red, pudiendo crear no solo redes en estrella y p2p sino también mallas y árboles.

Para poder tener una red, son necesarios como mínimo dos elementos. Un coordinador (FFD) que creará la red, le asignara el NWKID (NetWork IDentifier), y poseerá los mecanismos necesarios para la incorporación y eliminación de nodos en la red. Además es necesario, como mínimo, un nodo, que puede ser FFD o RFD, con el que comunicarse.

La topología en estrella consiste en un coordinador y una serie de nodos RFD (o FFD) que sólo se comunican con el coordinador.

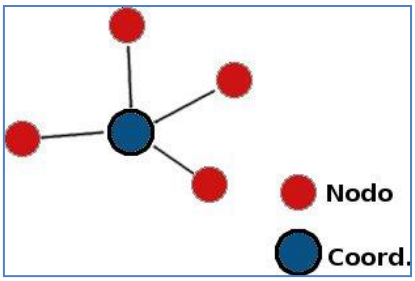


Fig 2.6 *Esquema del protocolo*

En la topología p2p, bien conocida, dos nodos solo pueden comunicarse entre sí directamente y, por tanto, si están en el radio de alcance mutuo. Esta topología permite a ZigBee crear otras más complejas, como redes en malla, siempre y cuando sea posible el enrutado de los datos de un nodo a otro.

2.3.5 Comunicaciones.

Los datos que realmente se desean enviar empiezan en las capas superiores de la pila, y cada capa añade información propia, formando los PDU (Protocol Data Unit). Así pues, cuando se envía un conjunto de datos, este contiene información de control de todas las capas de la pila. Cuando llega a su destino, cada capa extrae los datos que le concierne y, si es posible o necesario, pasa el resto a la capa superior. Así se produce una comunicación virtual entre capas de diferentes dispositivos.

Empecemos por lo más básico: la comunicación a nivel físico. Los dispositivos inalámbricos envían los datos usando ondas electromagnéticas. En este caso se utiliza modulación por frecuencia, en el espectro de los 2.4 GHz. Tenemos disponibles 16 canales en los que transmitir (separados entre si 5 MHz). El acceso al canal se hace utilizando CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) que es un mecanismo empleado para evitar que dos dispositivos usen el mismo canal a la vez, produciendo una colisión. Así pues, cuando un dispositivo transmite, el resto espera. Todos los dispositivos que estén en el radio de alcance del transmisor podrán escuchar el mensaje. Pero la mayoría de las veces queremos comunicarnos con uno solo.

Para ello, necesitamos alguna manera de identificar los dispositivos dentro de la red. Esto nos lo provee la capa superior: la capa MAC. Así pues, cada dispositivo posee una dirección MAC que debe ser única, de 64 bits. Se puede usar esta misma en las comunicaciones dentro de la red, o se puede intercambiar con el coordinador de la PAN (Personal Area Network) por una más corta de 16 bits. Esta dirección es la que identifica el origen y el destino de una trama dentro de la red. Cada trama debe tener un tamaño máximo de 127 bytes, incluyendo las cabeceras MAC, que son como máximo de 25 bytes (sin utilizar seguridad).

Teniendo esto en cuenta, y considerando las dos topologías de red posibles a este nivel, a saber estrella y peer-to-peer, solo podemos enviar datos a los nodos que estén dentro de nuestro radio de alcance. Es más, si el dispositivo es RFD, solo podrá enviar datos al coordinador. Esto es muy limitado. Para solucionarlo, debemos confiar en la siguiente capa: la capa de red (NWK, NetWorK).

Hasta ahora, todo lo visto pertenecía a las especificaciones del estándar IEEE 802.15.4. La capa de red la aporta ZigBee. El principal cometido de esta capa es proveer de un nivel mayor de abstracción. Añade una nueva dirección lógica a los dispositivos (16 bits) . Esto permite que se puedan enviar datos a otros nodos que no están dentro de la cobertura de transmisión. Para ello es necesario contar con unos dispositivos especiales que enrutan los datos a través de la red, llamados routers (que han de ser FFD). Ahora sí podemos crear una red amplia en la que cada nodo puede comunicarse con todos los otros nodos de la misma red. Además podemos hacer redes usando topologías diferentes, como malla o cluster-tree.

Para cumplir su cometido, la capa de red proporciona dos servicios, uno de datos (NLDE, Network Layer Data Entity) y otro de gestión (NLME, Network Layer Management Entity). El NLDE encapsula los datos de la capa superior añadiendo las cabeceras necesarias, y los pasa a la capa MAC, para ser enviados a su destino. También se encarga de retransmitir aquellos NPDUs (Network Protocol Data Unit) que tienen como destino otro nodo de la red (routing).

El NLME coordina las tareas de mantenimiento de la red: crear una nueva red o asociarse/desasociarse a una existente, direccionamiento de dispositivos, descubrimiento de nodos vecinos y rutas, control de recepción de datos, etc.

Con las capas PHY-MAC-NWK podemos crear una red completa, permitiendo a todos los nodos, poder comunicarse con otros nodos, de la misma o de distinta red, resolviendo problemas como el acceso simultáneo al canal o direccionamiento lógico de nodos. Aún así, para permitir más funcionalidad añadimos una capa a la pila que interactúe entre los objetos de la aplicación que desee usarla y la capa de red. Hablamos de la subcapa de Aplicación (APS, Aplication Support sub-layer).

La capa APS es la encargada de enviar los PDUs de una aplicación entre dos o mas dispositivos (llamada APSDE, APS Data Entity) y de descubrir y enlazar los dispositivos y mantener una base de datos de los objetos controlados, conocida como AIB, APS Information Base. Dos dispositivos se enlazan en la AIB en función de los servicios que ofrecen y de sus necesidades. Esto es útil para el direccionamiento indirecto. Así, es posible enviar un paquete a un dispositivo en función de la dirección de origen, ya que están vinculados. Esta tabla solo puede estar presente en el dispositivo coordinador o en uno designado a tal efecto.

Cuando la comunicación es directa, se han de especificar las direcciones de origen y destino del paquete. En cambio, en las comunicaciones indirectas cuyo origen tenga una entrada en la tabla de enlazado (AIB), el emisor solo necesita especificar el origen y enviar los datos al coordinador, que se encargará de hacerlos llegar al destino, que pueden ser varios dispositivos, en función de la AIB.

2.3.6 Encaminamiento (routing).

La capa MAC nos ofrece un identificador único de 64 bits para el direccionamiento, y la capa de red otro de 16 bits. Ello implica que podemos tener una gran cantidad de nodos en una misma red. Pero, si solo somos capaces de comunicarnos con aquellos que tenemos en el radio de alcance, la red esta muy limitada. Para ello, existen las técnicas de encaminamiento o enrutado. Se crean dispositivos que reenvían aquellos mensajes que no van dirigidos a si mismos. Así pues, cualquier nodo de la red pude comunicarse con cualquier otro nodo.

Los routers son dispositivos de propósito específico. Han de poseer toda la funcionalidad (FFD). No pueden entrar en modo ‘ahorro de energía’ como los RFD, ya que deben ser capaces de retransmitir los mensajes lo antes posible. Por la misma razón, han de escuchar el tráfico constantemente. Deben mantener tablas con las rutas descubiertas y funcionalidad para participar o iniciar el descubrimiento de nuevas o mejores rutas. También han de ser capaces de detectar y corregir errores. Las tablas contienen información sobre el coste de cada ruta. El coste determina cual es la mejor en un momento dado. La función que elige el coste de una ruta se determina a la hora de crear la implementación de la pila. Se puede basar en la latencia del recorrido de los mensajes, pero es recomendable que se tenga en consideración la carga media de la batería de los dispositivos que participan en la ruta.

En las tablas de enrutado pueden aparecer direcciones de cualquier dispositivo, pero solo los routers pueden participar en los métodos de enrutado. Si un mensaje llega a un dispositivo FFD que no es un router, comprueba la dirección de destino, y solo lo reenvía si pertenece a alguno de sus hijos (es decir, pertenece a alguno de los RFD que están asociados a él). Si es para él, lo pasa a la capa superior. En otro caso se descarta.

2.3.7 Perfiles.

Ya que ZigBee está pensado para la comunicación entre diversos dispositivos, posiblemente de fabricantes diferentes, es necesario un mecanismo para hacer compatibles los mensajes, comandos, etc. que pueden enviarse unos a otros. Para ello existen los perfiles de ZigBee.

Los perfiles son la clave para la comunicación entre dispositivos ZigBee. Definen los métodos de comunicación, el tipo de mensajes a utilizar, los comandos disponibles y las respuestas, etc. que permiten a dispositivos separados comunicarse para crear una aplicación distribuida. Casi todo tipo de operaciones han de estar definidas en un perfil. Por ejemplo, las tareas típicas de unirse a una red o descubrir dispositivos y servicios están soportadas por el 'perfil de dispositivos' ZigBee.

Cada perfil debe tener un identificador y, obviamente, este ha de ser único. Por ello, la ZigBee Alliance se reserva el derecho de asignar identificadores a los diversos perfiles. Si es necesaria la creación de un nuevo perfil, ha de hacerse la petición a la ZigBee Alliance.

Cada perfil contiene las descripciones de los dispositivos que incluye, los identificadores de cada cluster(y en su caso, sus atributos) y los tipos de servicio ofrecidos. Las descripciones de los dispositivos están definidas por un valor de 16 bits (es decir, hay 65536 posibles descripciones). Los identificadores de cluster son de 8 bits (existen 256 posibles clusters). A su vez, si el tipo de servicio ofrecido es orientado a pares clave-valor, cada cluster puede contener atributos por valor de 16 bits (o 65536 atributos por cluster). Existen muchas posibilidades dentro de un mismo perfil, y es labor del diseñador del perfil adecuar las necesidades para crear descriptores sencillos y permitir un proceso eficiente de mensajes.

Ya que cada dispositivo puede soportar más de un perfil, y cada perfil puede poseer varios clusters y múltiples descripciones, es necesaria una jerarquía de direccionamiento para acceder a los elementos del dispositivo. En primer lugar, se hace referencia al dispositivo entero usando sus direcciones IEEE y NWK. Por otro lado, se definen los Endpoints, que son campos de 8 bits que apuntan a cada una de las diferentes aplicaciones que están soportadas por un dispositivo. Por ejemplo, el 0×00 hace referencia al endpoint del perfil de dispositivo, y el 0xff hace referencia a todos los endpoints activos (endpoint broadcasting). Ya que los endpoints 0xf1-0xfe están reservados, es posible tener un total de 240 aplicaciones en los endpoints 0×01-0xf0.

Una vez establecidos los endpoints para las aplicaciones, se han de definir los descriptores. Como mínimo, el descriptor 'simple' ha de estar presente y disponible para las tareas de descubrimiento de servicio. Existen otros tipos de descriptores que contienen información acerca de la aplicacion, del servicio, etc. Pueden contener información sobre el endpoint al que hacen referencia, el perfil o la versión, pero también sobre el tipo de alimentación del dispositivo, el nivel de la batería, el fabricante, número de serie y hasta un icono o la dirección de este para representar el nodo en PCs, PDAs etc.

2.4 Sistemas Operativos.

Para las Redes de sensores inalámbricas, existen multitud de sistemas operativos con variadas características. Sin embargo, no todos son buenos para satisfacer las necesidades y las restricciones que nos imponen las propias redes. Es por esta razón, por la que muchos sistemas se descartan rápidamente.

Así pues, es posible reducir este número de sistemas operativos en unos pocos, que se detallan a continuación. Estos sistemas presentan capas de abstracciones para independizar al programador de los niveles inferiores (hardware). El estudio se basará en cómo manejan las tareas y eventos que ocurren en cada nodo.

2.4.1 Contiki.

Se trata de un Sistema Operativo de código abierto, multitarea, fácilmente portable, desarrollado para su uso en ordenadores que van desde los 8 bits, hasta los sistemas

integrados basados e microcontroladores, incluidos motes en redes inalámbricas. El nombre de Contiki, proviene de la balsa empleada por el explorador y escritor noruego Thor Heyerdahl's en su viaje a través del océano Pacífico. La mayoría de sus funciones y su base, fue desarrollado por Adam Dunkels, perteneciente al Swedish Institute of Computer Science.

A pesar de la multitarea incorporada y la pila TCP/IP, Contiki sólo necesita unos pocos kilobytes de código y de unos cientos de bytes de RAM. Un sistema completo con una interfaz gráfica de usuario (GUI) necesita alrededor de 30 kB de memoria RAM. Contiki está desarrollado para sistemas embebidos con pequeñas cantidades de memoria. Una configuración típica en Contiki, consume 2 kB de RAM y 40 kB de ROM.

Contiki se compone de eventos impulsados por el núcleo, que es la parte superior de los programas de aplicación son cargados y descargados en en tiempo de ejecución. Los procesos de Contiki, usan protothreads, se trata de mecanismos de poco consumo que proveen y permiten una programación lineal o de tipo hilo.

Se ejecuta en una gran variedad de plataformas que van desde los microcontralodores integrados como el MSP430 y la AVR para ordenadores antiguos.

Contiki usa la comunicación de tipo paso de mensajes a través de eventos, así como una interfaz gráfica opcional, ya sea directamente del subsistema gráfico de apoyo a nivel local o de los terminales conectados con red virtual de VNC o Telnet. Una instalación completa de Contiki incluye las siguientes características:

- Núcleo multitarea
- Multitarea opcional por aplicación preventiva
- Protothreads
- TCP/IP de red, incluyendo IPv6
- Ventanas del sistema y una GUI
- Red de visualización remota utilizando Visual Network Computing
- Un navegador web (decía ser el más pequeño del mundo)
- Personal del servidor web
- Cliente simple telnet.
- Protector de pantalla

Este sistema operativo será el que se utilice en el presente proyecto para la programación de los motes que se usarán en el desarrollo de este trabajo.

2.4.2 PalOS.

Es un sistema operativo desarrollado por la UCLA (Universidad de California). En la fase de inicialización del programa, cada tarea registra una tarea de eventos en la programación del sistema. Si la tarea 1 desea hablar con la tarea 2, lleva a cabo una petición (post) de un evento en la cola de eventos de la tarea 2, usando una funcionalidad del Scheduler (organizador o programador) del sistema, para que luego la tarea 2 capture ese evento al preguntar al Scheduler si tiene algún evento para él. Para un correcto funcionamiento de esta estructura de software, es necesario que un "timer" maneje la periodicidad con que una tarea registra eventos. La forma en que se implementa es a través de una tarea "timer". Esta posee tres colas:

1. Cola Nexo, encargada de interactuar con las demás tareas (recibe el envío de otras tareas).
2. Cola Delta, en la cual se ordenan los distintos eventos dependiendo del tiempo de expiración.
3. Eventos Expirados, donde se van colocando para su posterior ejecución.

2.4.3 SOS.

Fue desarrollado en la Universidad de UCLA específicamente en el “Networked and Embedded Systems Lab (NESL)”. Implementa un sistema de mensajería que permite múltiples hilos entre la base del sistema operativo y las aplicaciones, las cuales pasan a ser módulos que pueden ser cargadas o descargadas en tiempo de ejecución sin interrumpir la base del sistema operativo.

El principal objetivo de SOS es la reconfigurabilidad, es decir, la capacidad dinámica de agregar o eliminar módulos, permitiendo la construcción de software mucho más tolerante a fallos. Esto presenta dos grandes ventajas: uno es el hecho de poder realizar descargas de forma fácil, el otro es la capacidad de anular el funcionamiento de algún módulo defectuoso, de algún nodo que pertenece a la red. A continuación en la figura 2.22 se muestra un esquema de la arquitectura de SOS:

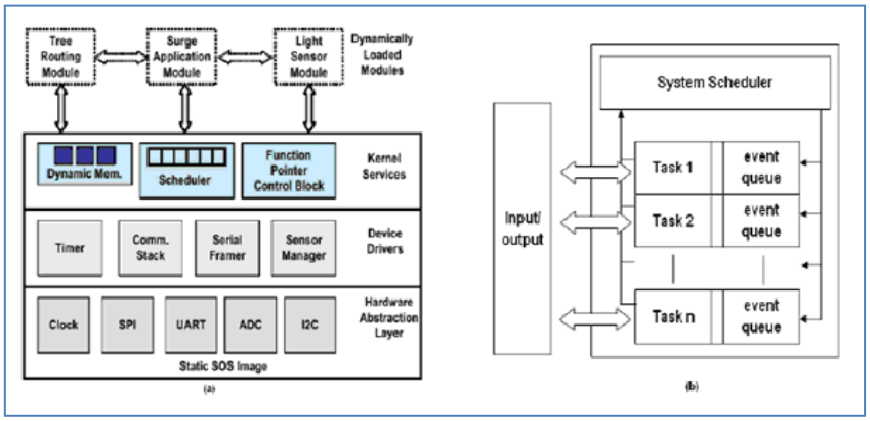


Fig 2.7 *Capas funcionales de sos y esquema de palos*

Además de las técnicas tradicionales usadas en el diseño de sistemas embebidos, las características del kernel de SOS son:

- Módulos cargados dinámicamente.
- Programación flexible de prioridades.
- Subsistema para manejo de memoria dinámica.

Las capas de abstracción de hardware y drivers son de la misma forma que para el sistema PalOS.

2.4.4 TinyOS.

TinyOS es un sistema operativo para trabajar con redes de sensores, desarrollado en la Universidad de Berkeley. TinyOS, es un SO libre y de código abierto,basado en componentes del sistema operativo y la orientación por plataforma de las Redes Inalámbricas de Sensores.
Se trata de un sistema operativo en el que se utiliza el NesC como lenguaje de programación. Está destinado a ser incorporado en Smartdust. TinyOS comenzó como una colaboración entre la Universidad de California con Intel y Crossbow, que ha crecido hasta tratarse de un gran consorcio internacional conocido como la Alianza TinyOS.

Puede ser visto como un conjunto de programas avanzados, el cual cuenta con un amplio uso por parte de comunidades de desarrollo, dada sus características de ser un proyecto de código abierto. Este “conjunto de programas” contiene numerosos algoritmos, que permitirán generar enrutamientos, así como también aplicaciones pre-construidas para

sensores. Además soporta diferentes plataformas de nodos de sensores, arquitecturas bases para el desarrollo de aplicaciones.

Los programas de TinyOS se construyen a partir de componentes de software, algunas de cuyas abstacciones son del hardware actual, los componentes están conectados entre sí por medio de interfaces. TinyOS proporciona interfaces y componentes para las abstracciones comunes, tales como la comunicación de paquetes, el enrutamiento, la detección, actuación y almacenamiento.

2.5 Hardware utilizado para las WSN.

2.5.1 Introducción.

En este apartado se explica el componente fundamental de toda red de sensores: sus nodos llamados Motes, los cuales disponen de un simple sensor o varios de ellos o incluso en algunos modelos incluyendo receptores de GPS. En el siguiente apartado se comienza con la descripción de la unidad principal: el mote, seguido de las diversas soluciones comerciales que los fabricantes basándose en este estándar han ido sacando al mercado, Posteriormente se describirán los principales kits de desarrollo existentes para introducirse en las WSN basadas en el estándar IEEE 802.15.4.

2.5.2 Motes.

Se exponen a continuación las características de los motes comerciales más ampliamente utilizados.

2.5.2.1 TelosB.

Los motes TelosB son una plataforma para aplicaciones en redes de sensores de muy bajo consumo y alta recolección de datos. Llevan integrados tanto los sensores como la radio, antena y microcontrolador y pueden ser fácilmente programados. En la Ilustración 2.16 se muestra un diagrama de bloques donde es posible encontrar las interconexiones entre estos elementos.

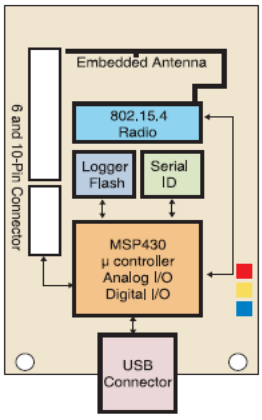
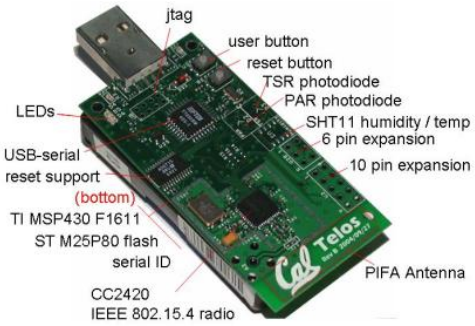


Fig 2.8 *Diagrama de bloques del mote TelosB*

Las operaciones de baja energía en los TelosB se llevan a cabo gracias al microcontrolador MSP430 F1611. Este procesador RISC de 16 bits consume muy poca energía tanto en el estado activo como durante el modo sleep. Para reducir al máximo este consumo, permanece en modo sleep durante la mayoría del tiempo. Se activa tan rápido como puede para procesar, enviar y una vez terminado entonces vuelve a sleep. Utiliza un controlador USB

del fabricante FTDI para comunicarse con el ordenador y lleva el módulo de radio CC2420, el cual envía y recibe bajo el estándar IEEE 802.15.4, el cual provee una fiable comunicación inalámbrica para WSNs. La radio tiene la posibilidad de enviar datos a muy alta frecuencia. El microcontrolador se comunica con la antena a través del puerto SPI y puede apagarlo para el modo de baja energía. La antena interna "Invertid –F micro strip" es una antena pseudo-omnidireccional que puede alcanzar los 50 metros dentro de un edificio y los 125 en el exterior.

En la Ilustración 2.17 se puede observar el reducido tamaño de este mote, donde el mayor espacio se usa para el conector USB necesario para su programación y para el alojamiento de las baterías que irían en la parte posterior.



6

Fig 2.9 *Mote TelosB*

Las características esenciales del mote TelosB son:

- Transmisor Chipcon inalámbrico de 250Kbps 2.4GHz IEEE 802.15.4
- Puede interactuar con otros dispositivos IEEE 802.15.4
- Microcontrolador MSP430 de 8MHz (10Kb deRAM y 48 Kb de Flash)
- ADC, DAC, supervisor de voltaje y controladora DMA integrada
- Antena, sensores de humedad (0-100% RH), temperatura(-40 ºC – 123,8 ºC) y radiación (320 nm - 730 nm)
- Muy bajo consumo (1,8 mA en estado activo y 5,1 μA en estado sleep).
- Rápido en despertar del modo sleep (<6 μs)
- Hardware para encriptación y autenticación de la capa de enlace
- Programación y recogida de datos por USB
- 16 pines para soportar una expansión y conector de antena opcional
- Conector de antena SMA
- Ayuda de TinyOS: enrutamiento de malla e implementación de las comunicaciones

Los sensores de humedad/temperatura están fabricados por Sensiron AG, producidos con procesadores CMOS y emparejados con un dispositivo ADC de 14 bits. Los coeficientes para estos sensores se guardan en la EEPROM. La precisión del sensor de temperatura es de ±0.5 º C y el de humedad de ±3.5% RH.
Esta plataforma consigue un bajo consumo de potencia permitiendo una larga vida a las baterías además de tener un tiempo mínimo en el estado de wakeup, otro de los objetivos dentro de las estrategias de bajo consumo.

Los TelosB se alimentan con 2 baterías del tipo AA, aunque si está conectado mediante el puerto USB para programación o comunicación, la alimentación la proporciona el ordenador. También proporcionan la capacidad de añadir dispositivos adicionales mediante los dos conectores de expansión de los que disponen, estos pueden ser configurados para controlar sensores analógicos, periféricos digitales y displays LCD.

2.5.2.2 Motes MICAz.

Usando la banda de frecuencias libres de 2.4GHz y cumpliendo la especificación del IEEE 802.15.4, Crossbow Technology empresa fundada en 1995 y especializada en redes de sensores, nos ofrece un mote que puede ser usado para formar redes inalámbricas de sensores de bajo consumo. En la Ilustración 2.18 podemos ver el aspecto exterior de este Mote, donde al igual que en modelos de otros fabricantes el mayor espacio es el ocupado por las 2 baterías del tipo AA.

Fig 2.10 *Mote Micaz*

Las principales características del mote Micaz son:

- Transceptor cumpliendo la especificación IEEE 802.15.4.
- Funciona en la banda ISM (Industrial, Scientific and Medical) desde 2.4GHz hasta 2.4835 GHz
- Utiliza la técnica de modulación conocida como DSSS (Direct Sequence Spread Spectrum - Espectro ensanchado por secuencia directa) la cuál es resistente a las interferencias de radiofrecuencia y provee de esta manera seguridad en los datos de sus transmisiones.
- Puede llegar alcanzar tasas de datos de hasta 250 Kbps
- Preinstalación de TinyOS.
- “Plug and Play” de las diferentes placas de sensores ofrecidas por crossbow. Entre estas destacan placas de entradas analógicas, digitales, soporte de I2C, SPI y UART (Universal Asynchronous Receiver-Transmitter).

Entre sus placas de expansión se encuentra la MIB510CA, la mostrada en la Ilustración 2.19, esta placa proporciona un puerto serie para poder conectarlo al PC, al fin que sirva de estación base para trasladar los datos de la red al PC, así como para las funciones de programación.

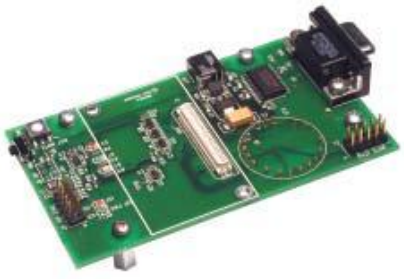


Fig 2.11 *Placa de expansión MIB510CA para MICAz*

2.5.3 IMOTE.

Originalmente diseñado por Intel, fue adquirido por la compañía Crossbow en 2007 y es la empresa que actualmente lo distribuye. Este mote se diferencia de la mayoría de los existentes en el mercado por usar un procesador XScale, funcionando a 416MHz, con una potencia superior a los actuales Atmel ATmega128, que tienen una velocidad de 16MHz. La memoria de este mote también es superior al resto de motes en el mercado siendo de decenas de megas, mientras que otros motes tienen montados unos cientos KB.
Es por esto que este mote es muy indicado para sensores que conlleven más complejidad computacional, como por ejemplo, sensores inerciales, de vibración y detección de movimientos sísmicos. Este mote consta de estas características por haber sido orientado al ámbito industrial más que al medioambiental donde los requerimientos de los sensores suelen ser menores. Las principales características del IMOTE son:

- Procesador Marvell PXA271 XScale de 13 – 416 MHz
- Co-procesador Marvell, Radio MMX DSP.
- 256K de SRAM, 32 MB de FLASH y 32 MB de SDRAM.
- Dispone de indicador de estado LED multicolor.
- Antena Integrada de 2.4 GHz, con capacidad de alcance de hasta 30 metros, para conseguir una distancia mayor se puede utilizar un conector de SMA.
- Permite comunicación con dispositivos USB.
- Gran cantidad de E/S estándar, 3XUART, 2XSPI, I2C, SDIO, GPIOs.
- Algunas de sus aplicaciones son: monitorización y análisis industrial, monitorización sísmica y de vibraciones.
- Es una plataforma avanzada con diseño modular.
- El procesador tiene dos modos de funcionamiento en bajo consumo: “sleep” y “deep sleep.”
- El procesador cuenta con numerosos temporizadores, entre ellos un reloj en tiempo real.
- Usa el transceptor de radio CC2420 IEEE 802.15.4.
- Velocidad de transmisión de datos a 250 Kb/s con 16 canales a 2.4 GHz

En la Ilustración 2.10 se muestra su placa y en la 2.11 un diagrama de bloques de sus componentes.

Fig 2.12 *Vista exterior del IMote*

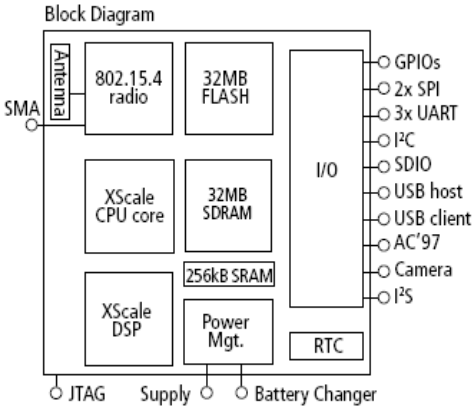


Fig 2.12 *Diagrama de bloques del IMOTE*

2.5.4 Waspmote.

La empresa Libelium (Spin-off de la Universidad de Zaragoza) lanzó en el 2009 su plataforma Waspmote, en el cuál centraron sus esfuerzos en conseguir una plataforma con muy bajo consumo, llegando a los 0.7 uA en modo hibernación. Esta plataforma usa los módulos de radio XBee. En la Ilustración 2.21 se muestra el mote con su módulo de radio, entrada USB, tarjeta SIM, lector de tarjetas MMC/SD y el conector de expansión.

Fig 2.14 *Waspmote en placa de desarrollo*

Esta plataforma tiene dos modelos diferentes de radio: el XBee y el XBee Pro que le permite trabajar en las siguientes frecuencias y protocolos:

- Frecuencias: 2.4Ghz, 900 MHz, 868MHz.
- Protocolo: 802.15.4, ZigBee.
- Potencia: 1mW, 100mW.

Estas radios permiten obtener rangos de operación de hasta 7 kilómetros en la banda de 2.4GHz, 24 km en la de 900MHz y 40 km en la banda de 868MHz. Estos módulos llevan incorporados un acelerómetro de 3 ejes en su placa. Además están basados en una arquitectura modular con lo cual es sencillo ampliar sus capacidades. Siendo algunos de los módulos ampliables más utilizados los de GPS, GPRS, lectores de tarjetas SD, Sensores para gases, diferentes sensores de luminosidad, vibración, nivel de líquidos y módulos de entrada salida genéricos como ADC, entradas digitales, etc.

2.5.5 Otros nodos comerciales.

Node	Picture	CPU	Memory	I/O Sensors	Radio
CSIRO Fleck		Atmega128L, 8MHz	512K external memory	Temperature, Light, Screw terminal for 4X digital i/o and 2X analog	Nordic 903
NICTOR				2 Serial (RS-232) Ports, 4 Digital Inputs, 2 Digital Outputs, 2 AnalogInputs	2.4GHz
BTnode	 Product Brief(pdf)	Atmel ATmega128L(AVR RISC 8 MHz @ 8 MIPS)	64+180 Kbyte SRAM, 128 Kbyte Flash ROM, 4 Kbyte EEPROM	UART, SPI, I2C, GPIO, ADC, Clock, Timer, LEDs Standard Molex 1.25mm Wire-to-Board and Hirose DF17 Board-to-Board connectors	Chipcon CC1000 operating in ISM Band 433-915 MHz)
EYES	 Product Information(pdf)	MSP 430F149 (5 MHz @ 16 Bit)	60 Kbytes of program memory , 2 Kbytes of data memory, 4 Kbyte EEPROM	UART, AD and I/O, JTAG interface and sensor board with compass, accelerometer, temperature sensor, light sensor, pressure sensor, microphone nad push button lines	RFM TR1001 hybrid radio transceiver
Rockwell Wins-Hidra Nodes		SrongARM 1100 (133MHz)	4MB Flash 1MB SRAM	seismic(geophone), acoustic, magnetometer, accelerometer, temperature and pressure	Connexant's RDSSS9M (100Kbps)
Sensoria WINS NG 2.0		SH-4 processor (167 Mhz)		GPS and imaging	Dual 2.4 Ghz FH
Sensoria WINS 3.0		Intel PXA255 (scalable from 100 to 400 MHz)	64MB SDRAM 32MB Flash	GPS, USB(2 host ports, 1 device port), RS-232 serial(5 generic + 1 Linux console),Audio in/out(1 stereo input, GPS, USB(2 host ports, 1 device port),PCMCIA/CardBus(1 external slot),	Dual embedded 802.11b modules

Fig 2.15 *Otros nodos comerciales*

2.6 . Kits de desarrollo

Por ser los kits de desarrollo un elemento importante en las primeras fases de desarrollo de un proyecto así como para evaluar la idoneidad de un mote para la aplicación a desarrollar, a continuación se describirán brevemente los diferentes kits de desarrollo inalámbricos basados en el estándar 802.14.5 de los principales fabricantes. Como ya se ha especificado en en el capítulo anterior, el kit que se usará en este proyecto es el ofrecido por la empresa Meshnetics, concretamente el kit "ZigBit Amp Development Kit", el cual se hablará mas detalladamente en el capítulo siguiente.

2.6.1 Microchip.

La casa microchip ofrece, entre otros, el PICDEM Z como kit de desarrollo inalámbrico, bajo el nombre de “PICDEM Z Demonstration Kit” con número de referencia DM163027.
El kit, mostrado en la Ilustración 2.24, está formado por los siguientes elementos:

- Dos placas de desarrollo con dos transceptores RF de radio MRF24J40MA
- Dos placas de desarrollo con dos RF transceptor de radio MRF24J40MA
- Código fuente de la pila de protocolo ZigBee.
- CD-ROM con las guías de usuario.

Fig 2.16 *Kit de desarrollo PICDEM Z*

Este kit de desarrollo se presenta con las siguientes características:

- Usa el Microchip’s MRF24J40 transceptor, como módulo de radio, usando la banda de 2.4 GHz, implementando el estándar IEEE 802.15.4.
- Dispone de un microcontrolador de la familia PIC 18. Concretamente esta placa utiliza el PIC18LF4620 MCU con 64 KB de memoria Flash.
- Presenta una pila de protocolo ZigBit preinstalada, soportando RFD (Reduced function device), FFD (Full Function Device) y Coordinador.
- Conector modular de 6 pines para interactuar directamente con MPLAB mediante ICD 2, y permitir la programación desde este software.
- El software ZENA™ para análisis de la red inalámbrica creada.

- Las placas de pruebas del kit están equipada con sensor de temperatura (Microchip TC77), LEDs e interruptores.
- Incorporan interface de comunicación RS-232 y regulador de tensión de 9V a 3.3V.
- Texas Instruments
- Texas Instruments lanzó al mercado a principios de 2008 el kit denominado “eZ430-RF2480”.
- Este kit, mostrado en la Ilustración 2.25, dispone de:
- 3 Placas de comunicaciones con módulo de radio CC2480.
- 2 Placas AAA eZ430 con alimentación por batería. (Sin USB).
- 1 Placa de emulador con USB, eZ430 (Para la programación y servidor host).

Fig 2.17 *Placa AAA eZ430*

Fig 2.18 *Placa eZ430*

El kit eZ430-RF2480 es un sistema microcontrolador con soporte USB basado en el MSP430 y usando el popular módulo de radio en la frecuencia de 2.4Ghz de TI, CC2480. Tiene todo lo necesario para evaluar este módulo de radio en poco tiempo y con el mínimo esfuerzo.

El kit utiliza una versión libre del entorno de desarrollo “IAR Embedded Workbench Kickstart “el cuál es usado para la familias de microcontroladores MSP430, y nos permite escribir, compilar y depurar nuestras aplicaciones. Las principales características de este kit son las siguientes:

- ZASA ,ZigBee Accelerator Sample Application, aplicación de demostración para PC ya desarrollada, para la monitorización de la temperatura y el nivel de la batería a través de los sensores que lleve incorporados los motes.
- Código de ejemplo para demostrar la configuración, la selección de red y la comunicación entre motes.
- 5 pines GPIO (Pines de propósito general de entrada/salida).
- Alta integración y muy bajo consumo.
- Dos pines de entrada/salida digital generales conectados a dos LEDs rojo y verde para una visualización rápida de esas salidas.
- Pulsador generador de interrupción para aplicaciones de control.
- Área mínima al tener la antena integrada en el propio chip.

La placa de comunicaciones CC2480 se ha diseñado para cumplir las especificaciones Z-Accel de TI, que imponen un sistema de comunicaciones de radio basado en el protocolo de puerto serie, aislando así la parte de radio como un sistema de caja negra con un API bien definida. La placa con USB eZ430, nos proporciona una conexión serie a través del USB con el PC, además de recibir la alimentación por medio de este. El resto de placas de eZ430 que se

alimentan a baterías nos permite tener movilidad con ellos y situarlos donde la aplicación lo requiera.

2.6.2 Crossbow.

La empresa Crossbow, líder en el mercado de las redes inalámbricas de sensores, tiene entre otros en el mercado el “WSN Imote2 Builder Kit”, basado en el IMote2. La característica más resaltables de este motes es su sistema operativo, que por defecto tiene instalado está basado en la tecnología “Microsoft Framework .NET” y por tanto su metodología de programación se realiza con el software Microsoft Visual Studio. El contenido del kit se puede ver en la Ilustración 2.15 , el cual está compuesto de:

- 3 Motes IMOTE2 pre-programados con la plataforma de desarrollo Microsoft .NET Micro Framework SDK.(IPR2410CA)
- 2 Placas de sensor. (ITS400CA)
- Copia de evaluación de Microsoft Visual Studio 2005.

Fig 2.19 *IMOTE2 de Crossbow*

2.6.3 Discovery Kit.

Zolertia ofrece el Dicovery Kit Z1 Este kit viene preparado para la instalación del sistema operativo TinyOs y Contiki, además de tener soporte para 6LoWPan. Una caracterisitca destacar del Zolteria Z1, el mote del kit, es la incorporación de los Phidget Port, lo que permite la fácil conexión de varios sensores a la placa a través de estos conectores. El kit Zolteria Discovery Kit Z1, incluye: 3 módulos Z1 con sus adaptadores para las baterías, 2 cables USB y el pack de baterías necesarias.

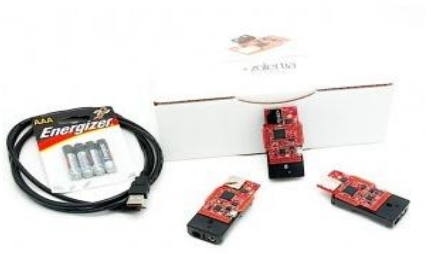


Fig 2.20 *Contenido de “Discovery Kit”*

El módulo Z1 está equipado con dos sensores digitales integrados en la misma placa (acelerómetro de 3 ejes y sensor de temperatura), es posible ver un detalle de este módulo en Ilustración 2.18.

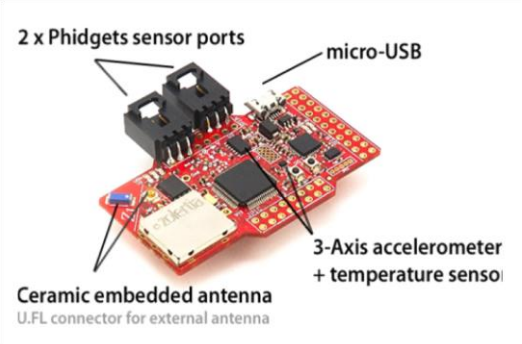


Fig 2.21 *Placa de desarrollo "Discovery Kit"*

Las principales características de este kit de desarrollo son:

- Implementación del protocolo 802.15.4
- Conector RP-SMA.
- Potenciómetro para entrada analógica.
- 2 Puertos (5V y 3V regulados)
- Conector de expansión de 54 pines para GPIO, buses digitales, entradas y salidas analógicas e interrupciones.
- ADC, UART, I2C, SPI
- Puerto Ziglet, está basado en I2C y permite la conexión de más de dos sensores a la placa de desarrollo.
- Conector USB
- LED Tricolor.

2.6.4 XBee y XBeePro Starter development kit.

XBee/XBee Pro es el más conocido kit de desarrollo bajo el estándar 802.15.4 presentado por la casa Digi (anteriormente Maxstream). Este kit destaca por su configuración por defecto de fábrica que te permite usarlo como un simple reemplazo de una conexión serie, además de ya traer el adaptador cruzado de DB9, para poder realizar las pruebas de alcance del la radio con solo conectarlo a la placa alimentada por batería.

El contenido del kit, el cual se puede ver en la Ilustración 2.29, está compuesto por dos módulos XBee y dos placas de interface con comunicación serie. Aunque también podemos encontrar placas de interface con comunicación a través de USB.

Fig 2.22 *Kit de desarrollo XBee/ XBee Pro*

Contenido detallado del kit de desarrollo:

- Placa de desarrollo con conector RS232
- Cable RS232
- Placa de desarrollo con conector USB
- Cable USB
- Adaptador de corriente 9V 1A
- Conector de batería 9V
- Adaptador Serial loopback. Ver Ilustración 2.20
- Guía de inicio
- CD

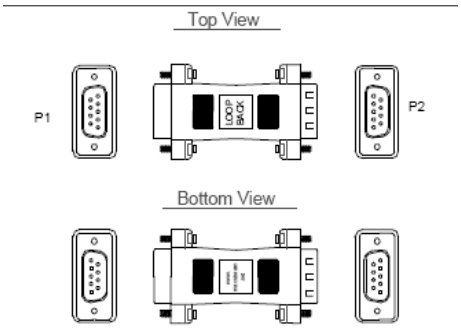


Fig 2.23 *Adaptador Serial Loopback*

Estas placas de interface basadas en el módulo XBee (ver figura 2.21) contienen LEDs que permite ver la calidad de la señal de transmisión (comportamiento por defecto), monitorización de salidas y/o entradas, pulsadores para las entradas digitales, conectores de expansión que permiten tener acceso a cada uno de los pines del modulo, así como del adaptador de corriente, y los conversores de adaptación de nivel de las señales del puerto serie. En el caso de la placa con puerto RS232, el adaptador convierte señales TTL del módulo XBee a señales RS232 y en la placa USB convierte las señales TTL serie a un puerto Serie-USB emulado, se pueden ver ambas placas en la Ilustración 2.22 .

Los módulos XBee son soluciones embebidas de radiofrecuencia que implementando el protocolo 802.15.4, permiten conectividad inalámbrica entre módulos. Con la implementación del estándar 802.15.4 se consigue tener un protocolo rápido para la comunicación en redes punto a punto y en redes peer-to-peer. Están diseñados para tener una alta confiabilidad en las transmisiones, requiriendo poco tiempo entre transmisiones y tiempos de comunicación predecibles.

Fig 2.24 *Módulo XBee*

Las principales características del módulo XBee son:

- No necesita configuración para empezar a funcionar como transmisor RF.
- Configuración de pines común a varios módulos de radio frecuencia.
- Alta tasa de transferencia de datos de hasta 250 Kbps hacia dispositivos finales.
- Transmite en la banda de 2.4GHz.
- Soporta modo hibernación para ahorrar energía. (Sleep mode).

Los módulos XBee y XBee-PRO son compatibles pin a pin, y su mayor diferencia es el precio y su poder de transmisión.

Fig 2.25 *Placa de desarrollo RS232*

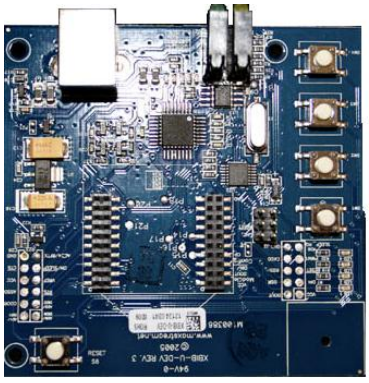


Fig 2.26 *Placa de desarrollo USB*

CAPÍTULO 3

DESCRIPCIÓN DEL SISTEMA

En este capítulo se hará hincapié en el kit usado en este proyecto, donde se hablará de la empresa desarrolladora, los productos variados que ofrece y un extenso estudio de las tarjetas a usar. Se comentará la manera de usar el software que incluye el kit para la gestión de la red ZigBee y se finalizará con un breve análisis del programador a usar.

3.1 ZigBit Development Kit.

MeshNetics es un proveedor tecnológico líder en el mercado especializado en control inalámbrico y sensores inalámbricos 802.15.4/ZigBee. MeshNetics ayuda a sus colaboradores y clientes a acelerar el tiempo de lanzamiento de los productos desarrollando y desplegando conjuntamente soluciones M2M para todo tipo de proyectos.

Los módulos ZigBee y el software de MeshNetics permiten añadir conectividad inalámbrica a los productos y soluciones de OEMs e integradores de sistema. Los módulos inalámbricos IEEE802.15.4/ZigBee producidos con los módulos ZigBit de MeshNetics resultan ser los mejores en su clase, con las mejores prestaciones de la industria, la mayor duración de las baterías, y los circuitos de menor tamaño. Estas características, unidas a la sencillez de integración en los sistemas, proporcionan a los desarrolladores una reducción considerable de los tiempos de lanzamiento y un ahorro importante en costes.

MeshNetics basa su estrategia a largo plazo en estándares abiertos y es miembro activo de la Alianza ZigBee y del la Fundación OPC. Todo el software de red ZigBee de MeshNetics está disponible para su descarga gratuita.

3.2 Módulos ZigBit.

ZigBit es un módulo 802.15.4/ZigBee de bajo consumo y alta sensibilidad. Encierra una funcionalidad impresionante en una superficie de 2,5 cm^2. Basado en el transceptor Atmel, líder en la industria, el ZigBit combina unas prestaciones de radio superiores con una excepcional facilidad de integración. El módulo ZigBit elimina la necesidad de desarrollos RF costosos en tiempo y dinero, y reduce el tiempo de lanzamiento para un amplio rango de aplicaciones inalámbricas. Existen diferentes tipos de módulos que ofrece esta empresa, a saber:

- ZigBit con Puerto RF Balanceado.
- ZigBit con Antena Chip Dual.
- ZigBit Amp.
- ZigBit 900.

Las características comunes de estos módulos son los siguientes:

- Bajo consumo: inferior a 6 µA en modo sleep.
- Amplio rango: hasta 100m en interior y unos 1.000m en exterior sin amplificar (con antena externa).
- El footprint más pequeño de la industria: 18,8x13,5x2,8mm.
- Software libre: BitCloud stack, SerialNet AT, OpenMAC.
- Dos opciones de antenas: salida RF balanceada y antena de chip dual.
- Amplio rango de temperatura: -40 a +85 °C.
- Kits de desarrollo de uso sencillo que disponen de diseños de referencia.
- Soporte técnico para cuestiones hardware y software.

El módulo que usa el kit de este proyecto es el denominado ZigBit Amp. A continuación se hablará más detalladamente de éste módulo en el apartado siguiente.

3.2.1 Descripción del módulo ZigBit Amp.

ZigBit es un módulo RF amplificado 802.15.4/ZigBee a 2,4GHZ con puerto RF balanceado. Su diseño RF único alcanza una rara combinación de rendimiento puntero con un bajo consumo, y su pequeño footprint de 5cm2 facilita la integración.

El módulo ZigBit se basa en la plataforma, líder en la industria, AVR Z-link de Atmel, que combina una sensibilidad de recepción de -101dBm y 20dBm de potencia de transmisión. El amplificador de baja potencia maximiza el rango, al mismo tiempo que mantiene el consumo al mínimo. El link budget mejor en su clase de 124dB garantiza al ZigBit Amp un rango mucho mayor que otros módulos con link budgets menores.

Fig 3.1 *Módulo ZigBit*

Las características clave del módulo ZigBit Amp son las siguientes:

- El mejor rango en campo en su clase: Aproximadamente 4 km.
- Alta sensibilidad de recepción (-104 dBm)
- Link budget superior (124 dB link budget)
- Muy bajo consumo (10 µA en modo sleep)
- Tamaño ultra compacto (28.0x13.5x2.0 mm^3)
- Diseño rápido con conector U.FL en placa
- Protocolo 802.15.4/ZigBee que se caracteriza por proporcionar una malla de red auto-organizada, auto-mantenida y con auto-recuperación (auto-healing).

Especificaciones:

- Banda de frecuencia: 2.400 - 2.483 GHz
- Número de canales: 16
- Ratio de datos: 250 kbps

- Sensibilidad (PER 1%): -104 dBm
- Máxima potencia de salida: +20 dBm
- Máxima potencia del transmisor: -2 - 20 dBm
- Alimentación: 3.0 - 3.6 V
- Consumo de recepción: 23 mA
- Consumo de transmisión: 50 mA
- Consumo en modo Sleep: 10 µA
- Memoria flash: 128 kB
- RAM: 8 kB
- EEPROM: 4 kB
- Rango de temperaturas de operación: -40 a +85 °C

Interfaces externos soportados:

- USART/SPI, I2C, 1-wire
- UART con control CTS/RTS
- JTAG
- 9 GPIOs libres (hasta un total de 25 GPIOs)
- 2 líneas de interrupción (IRQ) libres
- 4 líneas ADC

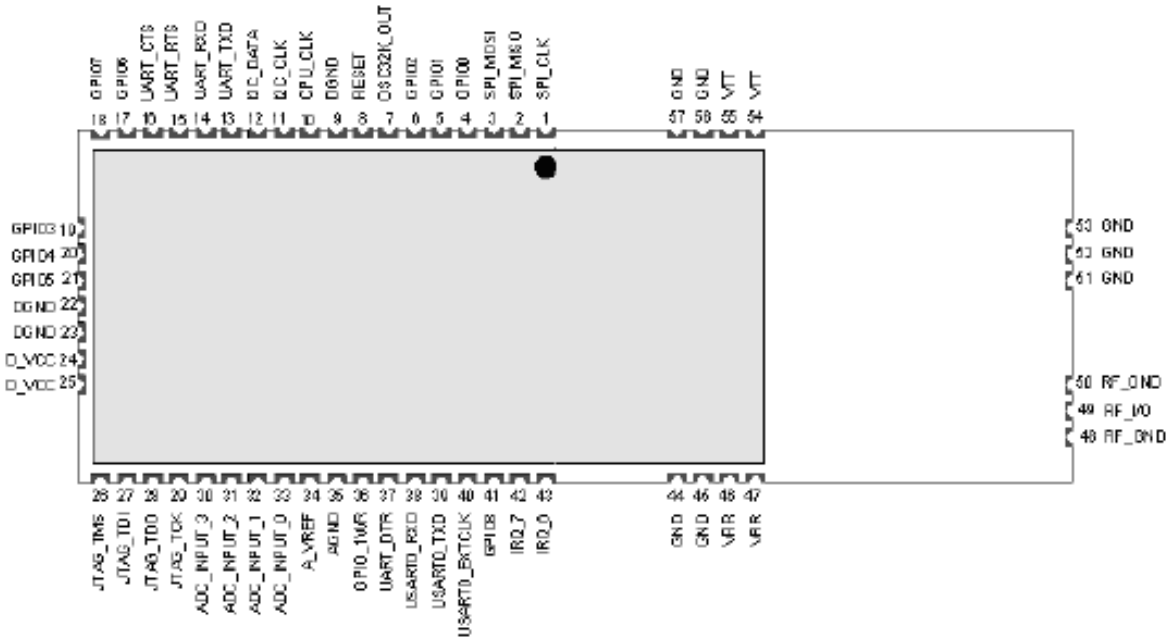
Opciones de software:

Los módulos ZigBit son compatibles con las tres configuraciones de pila disponibles desde MeshNetics:

- BitCloud Stack Es una robusta pila de software 802.15.4/ZigBee PRO, diseñada para redes sensores, control, monitorización y aplicaciones de adquisición de datos. Soporta topologías mesh y árbol.
- SerialNet es usado para la programación de módulos por medio de comandos serie AT. SerialNet emula la funcionalidad estándar de eZeeNet para que los desarrolladores puedan controlar la pila de software eZeeNet sin necesidad de programar el MCU directamente.
- OpenMac es la implementación en código abierto de la capa MAC IEEE 802.15.4 para expertos y entusiastas del software embebido. Soporta topologías de red estrella y punto a punto.

Datasheet:

ATZB-A24-U0 Pinout



ATZB-A24-UFL Pinout

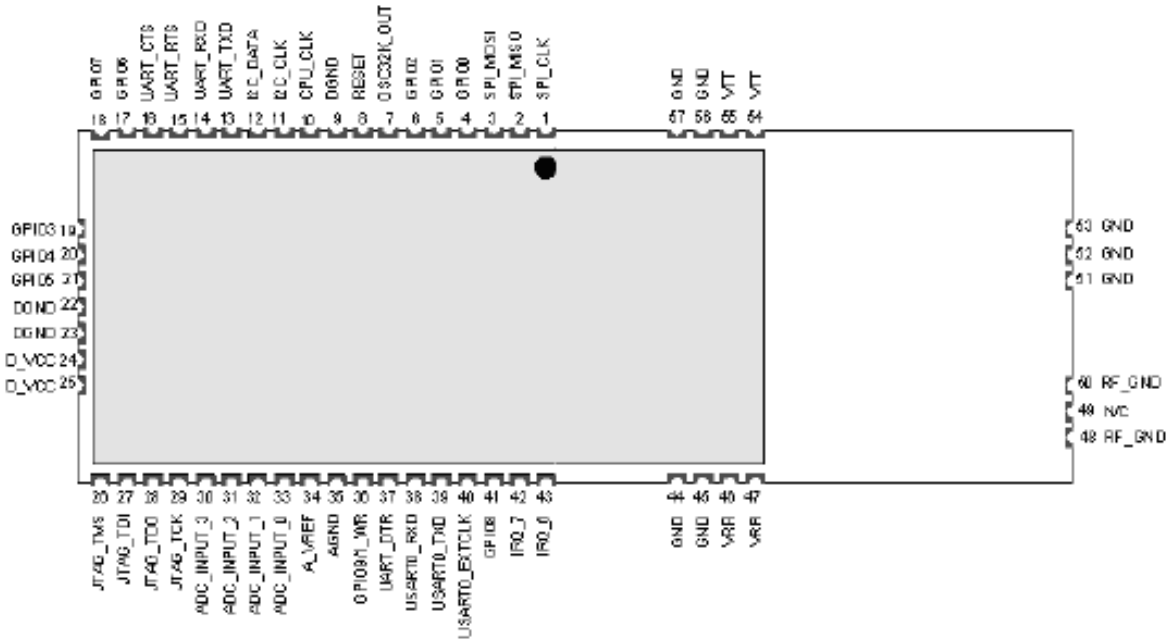


Fig 3.2 *Datasheet*

Connector Pin	Nombre del Pin	Descripción	I/O
1	SPI_CLK	Reserved for stack operation	O
2	SPI_MISO	Reserved for stack operation	I/O
3	SPI_MOSI	Reserved for stack operation	I/O
4	GPIO0	General purpose digital input/output 0	I/O
5	GPIO1	General purpose digital input/output 1	I/O
6	GPIO2	General purpose digital input/output 2	I/O
7	OSC32K_OUT	32.768 kHz clock output.	O
8	RESET	Reset input (active low).	I
9, 22, 23	DGND	Digital ground	
10	CPU_CLK	RF clock output. When module is in active state, 4 MHz signal is present on this line. While module is in the sleeping state, clock generation is stopped also.	O
11	I2C_CLK	I^2C serial clock output	O
12	I2C_DATA	I^2C serial data input/output	I/O
13	UART_TXD	UART receive input	I
14	UART_RXD	UART transmit output	O
15	UART_RTS	RTS input (Request To Send) for UART hardware flow control. Active low.	I
16	UART_CTS	CTS output (Clear To Send) for UART hardware flow control. Active low.	O
17	GPIO6	General purpose digital input/output 6	I/O
18	GPIO7	General purpose digital input/output 7	I/O
19	GPIO3	General purpose digital input/output 3	I/O
20	GPIO4	General purpose digital input/output 4	I/O
21	GPIO5	General purpose digital input/output 5	I/O
24, 25	D_VCC	Digital supply voltage (V_{DD})	
26	JTAG_TMS	JTAG test mode select	I
27	JTAG_TDI	JTAG test data input	I
28	JTAG_TDO	JTAG test data output	O
29	JTAG_TCK	JTAG test clock	I
30	ADC_INPUT_3	ADC input channel 3	I
31	ADC_INPUT_2	ADC input channel 2	I
32	ADC_INPUT_1	ADC input channel 1	I
33	BAT	ADC input channel 0. Used for battery level measurement. This pin level equals to VCC / 3.	I
34	A_VREF	Output/Input reference voltage for ADC	I/O
35	AGND	Analog ground	
36	GPIO_1WR	1-Wire Interface	I/O
37	UART_DTR	DTR input (Data Terminal Ready) for UART. Active low.	I
38	USART0_RXD	UART/SPI receive pin	I
39	USART0_TXD	UART/SPI transmit pin	O
40	USART0_EXTCLK	UART/SPI external clock	I
41	GPIO8	General purpose digital input/output 8	I/O
42	IRQ_7	Digital input interrupt request 7	I
43	IRQ_6	Digital input interrupt request 6	I
44, 46, 48	RF_GND	RF analog ground	
45	RFP_IO	Differential RF input/output.	I/O
47	RFN_IO	Differential RF input/output.	I/O

Fig 3.3 *Descripción de los pines*

3.3 Placas de desarrollo.

Las placas de sensores MeshBean están basadas en los módulos OEM ZigBit de 2,4GHz con antena PCB y un conector SMA externo (también disponible la versión con chip antena integrado). Cada placa soporta conectores de extensión estándar e incluye sensores de temperatura y luminosidad y otros dispositivos periféricos.

El módulo ZigBit con su ultra reducido tamaño y excepcional rendimiento RF dota a la placa de conectividad inalámbrica y la hace funcionar como un nodo de la red ZigBee. La placa MeshBean puede configurarse para operar como un coordinador de red o un router, tanto por DIP-switch como a través de comandos AT.

Alimentado por dos baterías AA, la placa de evaluación MeshBean incluye 2 de los sensores más frecuentemente usados - temperatura e iluminación. La placa presenta 3 LEDs de estado, botón de encendido/apagado y dos botones programables.

Las placas de sensores MeshBean están basadas en los módulos RF ZigBit, ZigBit Amp y ZigBit 900 combinados con una antena, que puede ser: integrada en la PCB, externa con conector SMA o chip integrado en el propio módulo. Alimentadas por dos baterías AA, las placas de desarrollo MeshBean / MeshBean Amp / MeshBean 900 incluyen 2 de los sensores más frecuentemente usados - temperatura e iluminación, así como otros dispositivos periféricos. La placa también soporta conectores de extensión para permitir un fácil acceso a los interfaces del módulo.

La alta capacidad de integración de los módulos ZigBit permite la conectividad inalámbrica de la placa y posibilita que funcione como un nodo en la red ZigBee. Las placas MeshBean pueden ser configuradas para funcionar como coordinador de red o como router mediante la configuración de sus DP-switch o por medio de comandos AT. ZigBit Amp es una versión amplificada del módulo ZigBit, se caracteriza por su mayor potencia de salida y sensibilidad, lo que resulta en un mayor alcance operativo. ZigBit 900 es la versión a 868/915MHz de los módulos RF ZigBee, se caracteriza por su mayor línea de visión y alta capacidad de penetración en paredes para su utilización en edificios

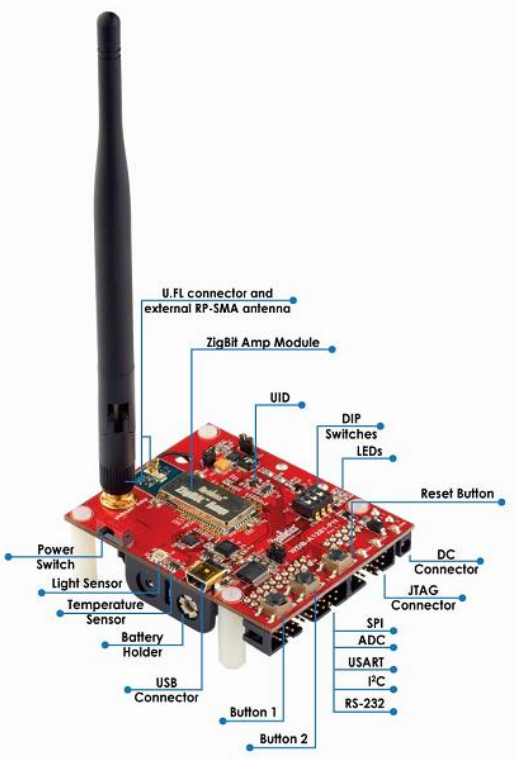


Fig 3.4 *Placa ZigBit Amp*

Características principales

- Placa PCB de dos capas con módulo ZigBit OEM.
- Entradas opcionales que incluyen interruptores DIP, 3 botones, 4 conectores ADC y sensores de luz y temperatura I2C.
- 3 LEDs de salida.
- Entradas y salidas a través de conectores USB y JTAG, USART, 1-Wire y 9 pins GPIO.
- Alimentación por USB, 2 baterías AA o red.

3.4 Kit de desarrollo.

Poderoso conjunto de herramientas que facilitan el desarrollo de soluciones de redes inalámbricas de sensores reduciendo costes y tiempos de lanzamiento. Con estos kit podrá:

- Desarrollar nuevos productos con conectividad inalámbrica listos para su lanzamiento.
- Crear aplicaciones a medida basadas en API BitCloud Stack.
- Desarrollar prototipos y probar sus dispositivos.
- Disfrutar de asistencia en todos los estados de desarrollo del producto, incluida la integración, interconexiones de sensores, selección de componentes, diseño de antenas y más.
- Disponer de soporte técnico adicional por e-mail.

A continuación se muestran los tipos de kits disponibles:

- Kit de Desarrollo ZigBit
- Kit de Desarrollo ZigBit Amp
- Kit de Desarrollo ZigBit 900

3.4.1 Kit de desarrollo ZigBit Amp.

Como ya se sabe, en este proyecto se usará el Kit de desarrollo ZigBit Amp. Este kit es un extenso conjunto de herramientas que soportan el rápido diseño, realización de prototipos y despliegue de soluciones inalámbricas basadas en la innovadora plataforma hardware ZigBit Amp. El kit incluye todo lo necesario para crear sistemas y aplicaciones inalámbricas listas para ser introducidas en el mercado, incluyendo:

- Plantilla de aplicaciones para desarrollo y personalización de aplicaciones de bootstrap.
- Soporte de programación embebida para aplicaciones OEM con acceso directo a la pila software BitCloud Stack a través de su API ANSI C.
- Conjunto completo de herramientas de evaluación, incluyendo medidas de distancia, interface de configuración serie, y aplicación de demostración de red de sensores inalámbricos.
- Soporte completo de sensores de terceras partes seleccionados por el desarrollador con código fuente para la capa de abstracción de hardware (HAL).
- Prácticos drivers de referencia y APIs para periféricos UART, I2C, ADC y 1-wire.
- Entorno de desarrollo intuitivo de Atmel, incluyendo la funcionalidad de depuración embebida con el JTAG de depuración in-circuit de Atmel.

Contenido del Kit de Desarrollo ZigBit

- 2 x placas de desarrollo MeshBean Amp.
- 2 x módulos OEM ZigBit Amplificados montados en placas MeshBean Amp.
- 2 x cables USB.
- 2 x cables de interfaz externo.
- 1 x CD de software y documentación.

Fig 3.5 *Contenido del Kit*

A continuación se muestra el diagrama funcional de la tarjeta.

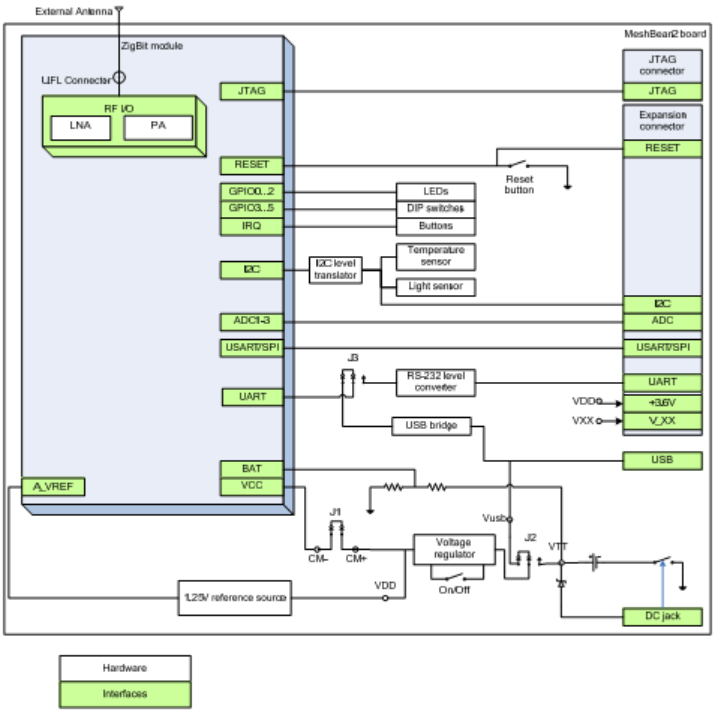


Fig 3.6 *Diagrama funcional de la placa*

A continuación se muestra detalladamente las conexiones del slot de expansión y del conector Jtag.

Pin	Name	I/O	Description
1	UART_RTS	Input	Request to Send Pin. RS-232 level.
2	UART_TXD	Input	Transmit Data Pin (meaning that the host device will transmit data to this line). RS-232 level.
3	UART_CTS	Output	Clear To Send signal from the module. Active low. RS-232 level.
4	UART_RXD	Output	Receive Data Pin (meaning that the host device will receive data from this line). RS-232 level.
5	GND		Digital/analog ground
6	GND		Digital/analog ground
7	I2C_CLK	Input	I^2C clock. It is connected to the I2C_CLK pin of the module via low-voltage level translators. For details, refer to ZigBit Amp datasheet [1].
8	I2C_DATA	Bidirectional	I^2C data. It is connected to the I2C DATA pin of the module via low-voltage level translators. For details, refer to ZigBit Amp datasheet [1].
9	+3.6V	Output	Output of internal voltage regulator. Normally, the voltage is 3.6 V.
10	V_XX	Output	ZigBit Amp supply voltage
11	RESET	Input	Reset Pin. Active low. This pin is connected in parallel to the RESET button on the board.
12	USART_TXD	Output	This is Transmit Data Pin for USART0 interface of the ZigBit Amp module. It is connected directly to the USART0_TXD pin of the module. Digital logic level. For details, refer to ZigBit Amp datasheet [1].
13	USART_RXD	Input	This is Receive Data Pin for USART0 interface of the ZigBit Amp module. It is connected directly to the USART0 RXD pin of the module. Digital logic level. For details, refer to ZigBit Amp datasheet [1].
14	USART_CLK	Input	This is Clock Data Pin for USART0 interface of the ZigBit Amp module. It is connected directly to the USART0 EXTCLK pin of the module. Digital logic level. For details, refer to ZigBit Amp datasheet [1].
15	GND		Digital/analog ground
16	ADC_INPUT1	Input	ADC input. This pin is connected directly to the ADC_INPUT_1 pin of the module. For details, refer to ZigBit Amp datasheet [1].
17	ADC_INPUT2	Input	ADC input. This pin is connected directly to the ADC INPUT 2 pin of the module. For details, refer to ZigBit Amp datasheet [1].
18	ADC_INPUT3	Input	ADC input. This pin is connected directly to the ADC_INPUT_3 pin of the module. For details, refer to ZigBit Amp datasheet [1].
19	GND		Digital/analog ground
20	GND		Digital/analog ground

Fig 3.7 *Pines del slot de expansión*

Pin	Name	Description
1	JTAG_TCK	Scan clock
2	JTAG_GND	Digital ground
3	JTAG_TDO	Test data output
4	JTAG_VCC	Controller supply voltage
5	JTAG_TMS	Test mode select
6	JTAG_RST	Reset controller; active low
7	N_Cont	Not connected
8	N_Cont	Not connected
9	JTAG_TDI	Test data input
10	JTAG_GND	Digital ground

Fig 3.8 *Pines del conector Jtag*

El conector Jtag es compatible con el ATmega JTAGICE mkII. A continuación se detallan los jumpers que contiene las tarjetas:

Posición del Jumper	Descripción
J1 montado	Esta posición es durante el funcionamiento normal
J1 abierto	En esta posición, el módulo Amp ZigBit está sin energía mientras que las partes restantes de la placa se alimentan. Esta posición se utiliza para medir el consumo de corriente del módulo Amp ZigBit.
J2 pines 2 y 3 unidos	La tarjeta se alimenta vía USB
J2 pines 2 y 1 unidos	La tarjeta se alimenta por la batería externa
J3 pin central unido con el pin RSR 232	La conexión en serie es por medio del conector Jtag
J3 pin central unido con el pin USB	La conexión en serie es vía USB

Fig 3.9 *Descripción de los Jumpers*

3.5 Software.

MeshNetics ofrece software embebido y configuraciones de la pila ZigBee que soporta auto-mantenimiento y auto-organización de redes con topología mesh y posibilita desarrollos e implementaciones sencillas de aplicaciones de redes inalámbricas de sensores. A continuación se mostrará los tipos de software que soporta este kit.

3.5.1 BitCloud Stack y Kit de Desarrollo Software.

BitCloud es una completa pila de software embebido de nueva generación de MeshNetics. La pila proporciona una plataforma de desarrollo de software para ejecutar aplicaciones fiables, escalables y seguras, en módulos ZigBee. BitCloud está diseñado con el objetivo explícito de soportar un extenso ecosistema de aplicaciones diseñadas por el usuario, dirigidas a diversos requerimientos, y permitir un amplio espectro de software personalizado. En el ámbito de las aplicaciones fundamentales, se incluye la automatización de edificios residenciales y comerciales, mediciones automáticas, seguimiento de activos, y automatización industrial.

BitCloud cumple por completo con los estándares ZigBee y ZigBee PRO para sensores y controladores inalámbricos. Proporciona una colección ampliada de APIs que, mientras mantiene el 100% de compromiso con el estándar, ofrece una funcionalidad extendida y diseñada teniendo en cuenta la facilidad de uso y la comodidad para el desarrollador. Como avezados expertos en tecnología ZigBee, MeshNetics ha creado la herramienta BitCloud para reducir dramáticamente la curva de aprendizaje del desarrollador, eliminar la innecesaria complejidad y descubrir tanta potencia de la plataforma hardware ZigBit como sea posible. La pila incorpora el valor de tres años de experiencia en diseño de sistemas inalámbricos, trabajo en campo, y observaciones reales de los usuarios.

El público objetivo de BitCloud son los diseñadores de sistemas, programadores embebidos e ingenieros de hardware que evalúen, construyan prototipos y lancen soluciones y productos inalámbricos basados en la plataforma ZigBit. BitCloud se entrega como un kit de desarrollo de software, que incluye (1) amplia documentación, (2) surtido estándar de librerías que contienen múltiples componentes de la pila, (3) Aplicaciones de ejemplo en código fuente, así como (4) una completa colección de drivers de periféricos (también en código fuente) para las plataformas soportadas.

Características clave:

- Compromiso completo con ZigBee y ZigBee PRO
- API C de uso sencillo y comandos serie AT disponibles
- Lo último en fiabilidad de datos con enrutamiento real mesh
- Soporte de redes grandes (cientos de dispositivos)
- Optimizado para consumo extremadamente bajo (duración de las baterías de 5-15 años)
- Extensa API de seguridad
- Capacidad de actualización del software por el aire
- Herramientas de desarrollo flexibles y sencillas de manejar

Arquitectura de la pila:

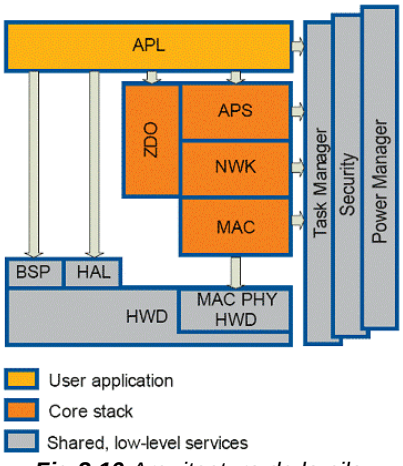


Fig 3.10 *Arquitectura de la pila*

- ZDO & APS: Proporciona un lote de APIs Device Object completamente conforme con ZigBee para habilitar el control de la red (comenzar, reinicio, formación, unión), y de la alimentación (dormir, despertar). Define los tipos de Perfiles de Dispositivo, y los comandos de descubrimiento de de dispositivos y servicios. Proporciona APIs para reconocimiento y transmisión unicast, multicast, y broadcast.
- BSP & HAL: Hardware Abstraction Layer (HAL) incluye un completo repertorio de APIs para su utilización en los recursos hardware del módulo (EEPROM, app, sleep, y watchdog timers) y controladores de referencia para diseños rápidos e integración sin problemas con un surtido de periféricos externos (IRQ, I2C, SPI, UART, 1-wire). Board Support Package (BSP) incluye un completo lote de drivers para controlar periféricos estándar (sensores, chip UID), colocados en una MeshBean de desarrollos.
- Task Manager: Proporciona una API para programar tareas en una cola con prioridad, optimizada para entornos de pila ZigBee multicapa y demandas de protocolos críticos en el tiempo.

Entorno de desarrollo:

	Windows 2000/XP/Vista	Linux
IDE, Depurador	AVR Studio 4.13 + Service Pack 2	AVArICE 2.40, GDB 6.30, DDD debugger 3.3.10
Compilador/Herramientas	WinAVR 20070525	AVR GCC 3.4.5, GNU make 3.80
Emulador	AVR AT JTAG ICE MKII (USB)	AVR AT JTAG ICE MKII (RS-232)

Fig 3.11 *Entorno de desarrollo*

Disponibilidad y soporte:

BitCloud se encuentra disponible como parte del Kit de Desarrollo ZigBit y el Kit de Desarrollo ZigBit Amp de MeshNetics. Los usuarios con el paquete completo de soporte tienen derecho a un año de actualizaciones gratis de software BitCloud, consultoría de diseño de aplicaciones profesionales, y acceso online al centro de servicio al cliente con una línea directa a los expertos software y hardware de MeshNetics. Se anima a todos los actuales clientes eZeeNet a cambiar a BitCloud. MeshNetics proporcionará asistencia para portar aplicaciones ZigBee a BitCloud bajo los términos de un acuerdo de soporte.

3.5.2 MeshNetic OpenMac.

Implementación, en código abierto, de la subcapa de control de acceso al medio (MAC) IEEE802.15.4. Forma la base del software MeshNetics y soporta topologías de redes peer-to-peer y estrella.

OpenMac ha sido portado a varias plataformas basadas en AVR, incluido los propios módulos OEM ZigBit de MeshNetics. Se suministra la versión completa con código de ejemplo y controladores de dispositivos.

Objetivos MeshNetics OpenMAC:

Permitir a usuarios, que no requieran la funcionalidad completa de la pila software MeshNetics eZeeNet, desarrollar aplicaciones WSN a medida sobre módulos ZigBit.
Permitir a usuarios avanzados modificar internamente OpenMAC para adecuarlo a necesidades específicas. Comenzar a desarrollar aplicaciones con la completa documentación de aplicaciones de ejemplo.

Facilitar una API C eficaz para aquellos desarrolladores no familiarizados con los lenguajes de programación TinyOS o nesC. (tecnologías del núcleo de OpenMAC).

Proporcionan un diseño de referencia para ser portado a plataformas hardware análogas. OpenMAC es el código de producción y se incluye en el software MeshNetics eZeeNet, con una amplia ayuda y documentación

IEEE 802.15.4 Certification:

El software MeshNetics OpenMAC ha sido certificado por la National Technical Systems (NTS) de acuerdo con las directrices ZigBee Alliance para asegurar la interoperatibilidad con el software MAC certificado de otros fabricantes.

OpenMAC y pila software eZeeNet:

La capa MAC se ocupa de todo el acceso al canal físico de radio y es responsable de las siguientes tareas:

- Sincronizar a las balizas.
- Soportar asociaciones y des-asociaciones de sensores.
- Soportar seguridad de dispositivos.
- Mediar en el acceso a radio compartido por múltiples iguales.
- Manejar y mantener el mecanismo GTS.
- Facilitar un enlace fiable entre dos nodos MAC de dos dispositivos

Aplicaciones:

- Control remoto punto a punto.
- Reemplazar al cable para enlaces punto a punto y estrellas.
- Redes single-hop con un gran número de terminales.
- Aplicaciones con duros saltos en latencia de transmisión de datos y/p sincronización distribuida.

3.5.3 SerialNet.

En lo alto de la pila software de IEEE 802.15.4/ZigBee PRO BitCloud Stack, SerialNet es la herramienta ideal para desarrollar dispositivos inteligentes de red sin la necesidad de programar directamente los módulos ZigBit. Usando la plataforma ZigBit como una radio controlada con comandos serie AT, los usuarios de SerialNet pueden diseñar prototipos de aplicaciones WSN en un tiempo menor que el requerido si fuese necesaria la programación embebida.

Los comandos SerialNet AT se basan en ASCII y lenguaje comprensible para el hombre, los que asegura la facilidad de depuración y ampliación. Un procesador host puede controlar un módulo ZigBit cargado con SerialNet sobre UART o cualquier otro interfaz serie. Sin embargo, no se requiere un procesador host en cada nodo de la red. Pueden enviarse comandos SerialNet AT por la red y ser interpretados remotamente en cualquier dispositivo ZigBit que tenga habilitado SerialNet. Además, los comandos SerialNet AT son también adecuados para facilitar la utilización de redes a través del aire.

Características clave:

- No se requiere experiencia en programación embebida.
- Perfiles privados con interfaces de comandos serie de fácil uso.
- Amplio juego de comandos con mensajes estándar de diagnóstico.
- Redes con Topología mesh o anillo.
- Forma altamente estructurada de acceso a la mayor parte de la funcionalidad de la pila de software BitCloud Stack.

- El juego de comandos SerialNet AT se construye sobre la extensión inalámbrica del juego de comandos V.250. Actualmente, el lenguaje de comandos AT contiene unos 50 comandos y 40 registros-S. SerialNet se incluye como una imagen firmware dentro del paquete del Kit de de Desarrollo ZigBit.

Los comandos AT se enmarcan en las siguientes categorías:

- Red, administración de nodos y parámetros de red.
- Transmisión de datos.
- Control genérico
- Control de interfaz de host.
- Control de hardware.
- Administración remota.

Los cuatro primeros grupos son mapeados directamente a las funciones BitCloud API correspondientes. Los dos últimos grupos contienen sub-colecciones de funcionalidad BitCloud. El juego de comandos AT también contiene comandos dedicados a reinicios seguros del módulo y restauración de parámetros de fábrica. La función control de hardware permite configurar/leer/escribir pins GPIO, leer A/D y administrar otros periféricos. Las funciones de administración remota son habilitadas por comandos AT opcionalmente protegidos que se envían de un nodo ZigBit a otro. Los nodos de destino ejecutan las secuencias de comandos AT recibidas, como si fuese a través de un puerto serie.

3.6 Introducción a la aplicación WSNDemo.

Esta demo para PC permite visualizar de una manera gráfica los valores recibidos de los sensores preinstalados en las tarjetas (luz, temperatura y batería), además de varios parámetros de conexión, como pueden ser la potencia de la señal, la dirección física de cada tarjeta, etc. Con esta demo se muestra el funcionamiento de la red ZigBit en un código basado en el API de "eZeeNet Software". Esta aplicación cuenta con firmware embebido, compatible con funciones del coordinador, del router y del dispositivo final (end device), como está establecido en el protocolo de comunicación ZigBee.

Se recuerda que esta aplicación es una demo básica para la monitorización de las tarjetas y que no permite la visualización de nuevos sensores instalados. De aquí se explica la razón principal de este proyecto. Se ha dado la necesidad de crear una práctica aplicación en Java para la visualización de nuevos sensores, además de los preinstalados en las tarjetas. Al final de este capítulo se muestra la manera de modificar el código de esta aplicación para añadir nuevos sensores.

3.6.1 Instalación del WSNDemo.

Para instalar esta aplicación hay que ejecutar el paquete que se encuentra dentro de la carpeta software llamado MeshNetics_ZigBit_Amp_Development_Kit_Complete(v2.3.0).exe. Una vez instalado, dentro de la carpeta que se encuentra por defecto en C:\Archivos de programa\MeshNetics\ZDK_Amp_Complete\Evaluation Tools\WSNDemo (WSN Monitor) se encuentra el instalador del monitor. Una vez hecho esto se graban en las tarjetas a utilizar la imagen del código de la aplicación con el AVR Studio, como se mostró en capítulos anteriores.

3.6.2 Principios de funcionamiento.

Gracias a la aplicación de la Demo WSN, los dispositivos Amp MeshBean se organizan en un conjunto de nodos que constituyen una red inalámbrica. Los leds de las tarjetas indican el estado actual y las actividades que se están llevando a cabo. Los datos que se obtienen de

los sensores que se encuentran en cada dispositivo son enviados al coordinador y son mostrados en el panel del monitor WSN, como son la temperatura, la luz y el nivel de la batería.

El dispositivo final está preparado sobre todo para dormir. Su consumo de energía es muy bajo, y se despierta cada 10 segundos para realizar las funciones. Durante el período de sueño, se puede forzar el dispositivo para despertar pulsando el botón SW1.

El router envía los datos cada 1 segundo. Mediante la interfaz uart, el coordinador transmite los paquetes recibidos, junto con sus propios datos de los sensores, para la aplicación WSN.

En tiempo real, el monitor WSN visualiza la topología de la red en forma de árbol. También muestra los parámetros del nodo como las direcciones del nodo, la información de los sensores y la calidad de la señal de vínculo. Dentro de la aplicación WSN se puede cambiar la máscara del canal de red, los tiempos de espera de nodo y se puede restablecer cualquier nodo de forma remota.

El procedimiento para cargar este programa es el mismo visto anteriormente. Se abre el archivo WSNDemoApp.aps que se encuentra en la carpeta de programas AVR. Se compila y a continuación se programan todos los dispositivos con la misma imagen. Una vez hecho esto hay que configurar los interruptores de cada dispositivo siguiendo la tabla mostrada a continuación:

1	2	3	Descripción
ON	ON	X	La tarjeta se configura como Coordinador
ON	OFF	X	La tarjeta se configura como Router
OFF	OFF	X	La tarjeta se configura como Dispositivo final

Fig 3.12 *Configuración de los switch*

Como se ha comentado, puede saberse la actividad de cada dispositivo por la acción de sus leds. A continuación se muestra una tabla que muestra dichas actividades:

	Estado de los leds		
Estado del dispositivo	Led rojo	Led amarillo	Led verde
Búsqueda de red	OFF	OFF	PARPADEANDO
Dentro de la red			ON
Mensaje recibido		PARPADEANDO	ON
Mensaje transmitido	PARPADEANDO		ON
Cambio de la MAC	PARPADEANDO	PARPADEANDO	PARPADEANDO
Función dormir	OFF	OFF	OFF
Indicación tipo dispositivo	Todos los leds parpadean una vez en el router, dos veces en el dispositivo final y tres veces en el coordinador		
Dirección de MAC perdida	ON		

Fig 3.13 *Estado de los leds*

El monitor WSN es una aplicación para el programa WSN Demo que se utiliza para mostrar topología WSN e información sobre la red WSN. Puede observarse en la siguiente imagen.

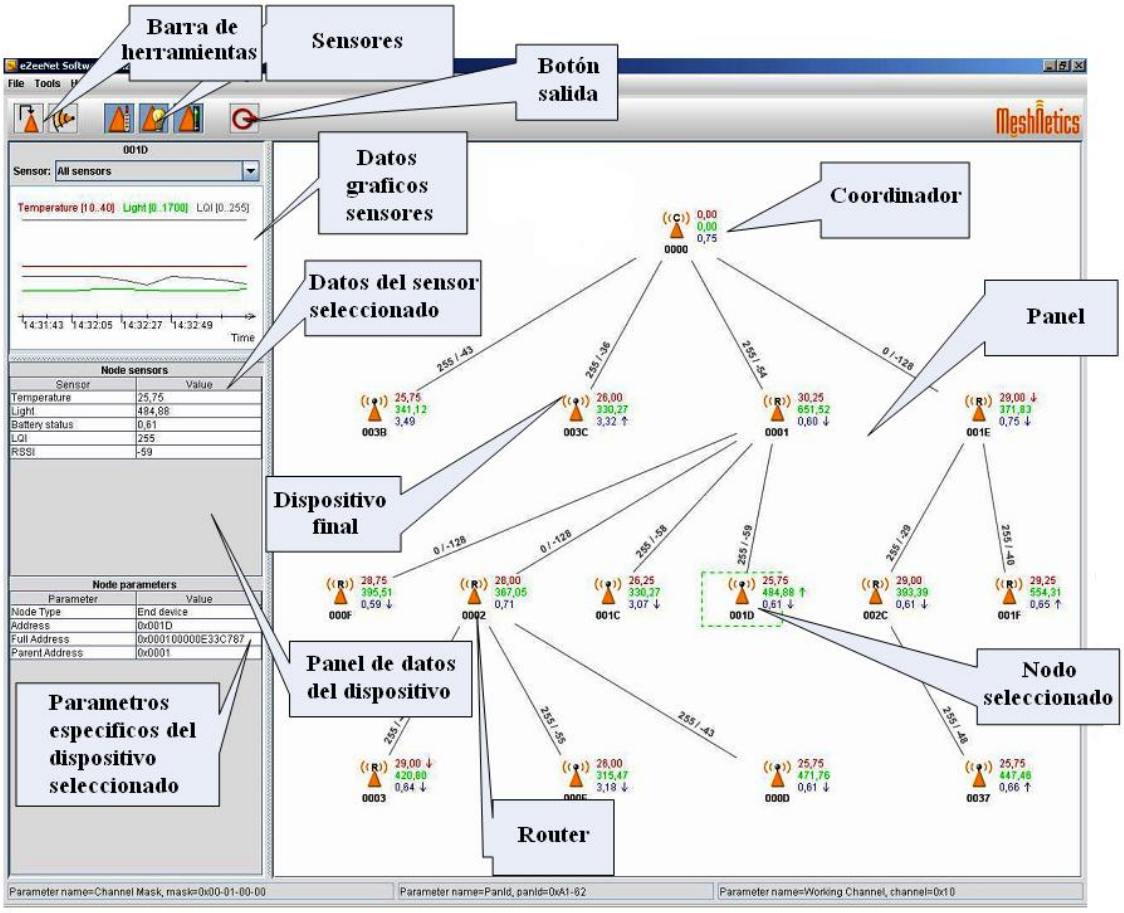


Fig 3.14 *Ventana principal de la aplicación*

El panel muestra la topología de la red, los datos de los sensores, el panel de gráficos, el panel de datos de los sensores y la barra de herramientas. Además muestra el árbol de la creación de redes en tiempo real. Ayuda a controlar la formación y evolución de la red como la participación de los nodos. Cada uno de los nodos que se muestra es representado por un icono junto con su nombre y los datos del nodo.

El panel de datos del nodo muestra los datos procedentes de cada uno de los nodos del sensor seleccionado. Se representa en gráficos y en forma de tabla. Hay otros parámetros que también pueden ser observados en forma de tabla para cada nodo.

3.6.3 Configuración de la aplicación.

Una vez visto el entorno de la aplicación se procede a la conexión del coordinador con el PC por el puerto USB, configurando previamente el pin J3. A continuación, se ejecuta el monitor WSN. En el inicio, el programa intentará utilizar el puerto COM por defecto para conectar con el coordinador. Hay que establecer un puerto COM adecuado a través de menú Herramientas / Opciones como se observa en la imagen. Si el icono no aparece se debe reiniciar el programa.

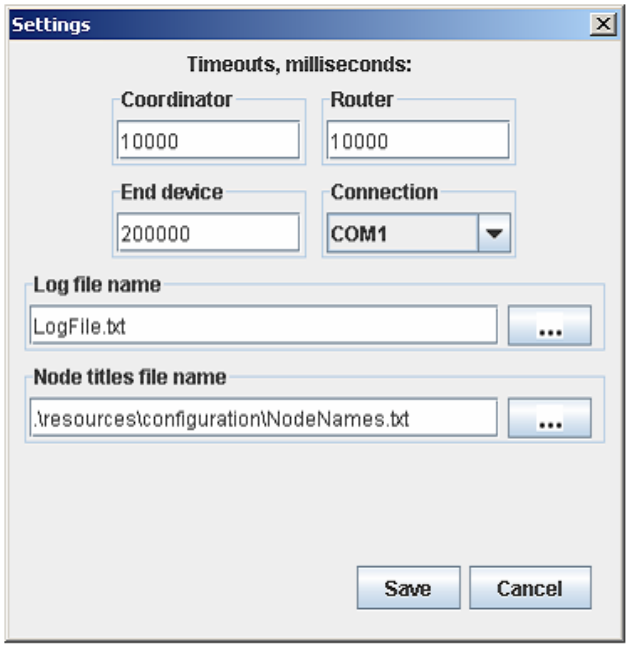


Fig 3.15 *Panel de herramientas*

Observando la topología de árbol y el funcionamiento de los controles de interfaz gráfica de usuario, el usuario puede seleccionar cualquiera de los nodos para ver los datos en tres formas diferentes:

- Tabla de texto.
- Gráfico.
- Los datos de los sensores en el diagrama de la topología árbol. Estos valores adjuntan unas flechas que indican los aumentos o disminuciones relativas.

El panel muestra las lecturas de temperatura y luz, así como el nivel de batería para cualquier nodo seleccionado. Además, estos valores se muestran en el gráfico de datos del sensor y ver su evolución en el tiempo.

3.7 ATmega128.

Se trata del microcontrolador que contiene el programador JTAGICE mkII, el cual se usará para programar las tarjetas. El ATmega128 es un potente microcontrolador que proporciona una gran solución flexible y rentable para muchas de las aplicaciones de control. El ATmega128 AVR está respaldado por un conjunto completo de programas y herramientas incluidas en el sistema: C compiladores, ensambladores de macro, programa depurador/simuladores, en el circuito Emuladores, y kits de evaluación.

El microcontrolador Atmega128 combina un conjunto de 32 instrucciones de registro de propósito general y control de periféricos que están directamente conectadas a la Unidad Aritmética Lógica (ALU). La arquitectura resultante es más eficiente y diez veces más rápida que los microcontroladores RISC convencionales.

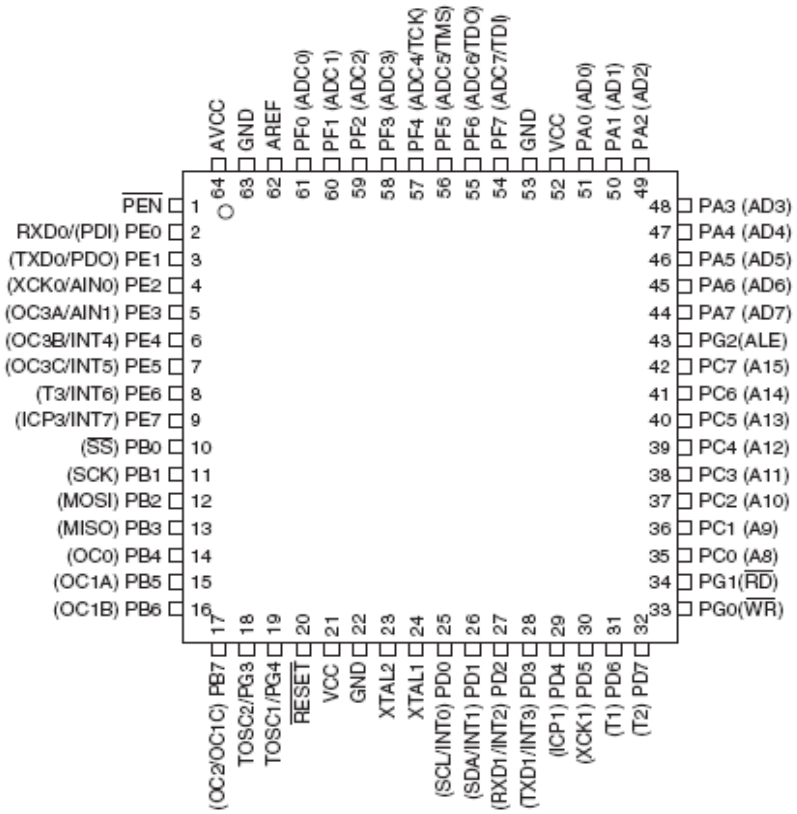


Fig 3.16 *Microcontrolador ATmega128*

El ATmega128 ofrece las siguientes características:

- Alto rendimiento y bajo consumo con el microcontrolador AVR de 8 bits.
- Avanzada arquitectura RISC
    - 133 instrucciones de alto alcance, la mayoría de un simple ciclo de reloj
    - 32 x 8 Registros de trabajo de propósito general + Registros de control periféricos
    - Operaciones completamente estáticas
- Alta resistencia de los segmentos de la memoria no volátil
- 128K Bytes de Sistema autónomo programable de la memoria del programa Flash
- 4K Bytes EEPROM
- 4K Bytes SRAM Interna
- Escribir / Borrar ciclos: 10000 Flash/100000 EEPROM
- Tiempo de retención de datos: 20 años a 85 ° C/100 años a 25 ° C
- Arranque Opcional con código de la sección independiente de bloqueo Bits
- Hasta 64K Bytes de espacio de memoria externa opcional
- Programación de Software Security Lock
- En SPI Interface para el Sistema de Programación
- Interfaz JTAG (estándar IEEE 1149,1 Compliant)
- Capacidades de Fronteras Según el Estándar JTAG
- chip Apoyo Debug extensible
- Programación de Flash, EEPROM, fusibles y Lock Bits a través de la interfaz JTAG

3.8 Conclusiones.

Los productos de MeshNetics pueden presumir de encontrarse entre los líderes de la tecnología más emergente del mercado M2M. Sus magníficas prestaciones, unidas a una increíble sencillez de integración, los convierten en la mejor solución para aquellos proyectos industriales que necesiten comunicación inalámbrica. Con estos módulos basados en ZigBee, cualquier desarrollador podrá crear, de forma sencilla, rápida y cómoda, redes de sensores inalámbricos con total confianza y seguridad. Por esto y por mucho más se ha escogido este kit para ser estudiado.
Con ZigBee y MeshNetics se abre una multitud de posibilidades en soluciones M2M para:

- Automatización de edificios
- Equipos de calefacción, ventilación y aire acondicionado (HVAC)
- Lectura automatizada de medidas
- Mantenimiento predictivo
- Transporte
- Seguimiento de recursos

CAPÍTULO 4

APRENDIZAJE DEL USO DEL KIT DE DESAROLLO

En este capítulo se mostrará un pequeño tutorial del uso del kit de desarrollo ZigBit de la empresa MeshNetics. Se comenzará por una breve introducción al uso del entorno de programación de ATMEL llamado AVR STUDIO. Seguidamente se mostrará las maneras útil de programar las tarjetas y el software implementado en ellas. Además, se mostrará la manera de modificar el código para añadir nuevos sensores a la red ZIgBee.

4.1 Introducción al entorno de programación AVR Studio.

Antes de comenzar con el estudio del kit ZigBit es necesario conocer el entorno de programación que se utilizará para programar las tarjetas para luego ir familiarizándose con la pila Meshnetics eZeeNet ZigBee que ofrece este kit. Es recomendable usar el programador ATMEGA por la facilidad y estabilidad que ofrece a la hora de programar las memorias flash y EEPROM de las tarjetas por medio de unas imágenes con formato HEX. AVR Studio es el software necesario para escribir el código para programar concretamente un ATEMEGA 1281. Hay ciertas demos que ofrece este kit para poder experimentar con las tarjetas. Estas demos están en el formato indicado, por lo que es necesario saber cómo instalarlas en las tarjetas.

4.1.1 Instalación del AVR STUDIO.

Lo primero que se necesita para programar un ATMEGA es el software para escribir el código (AVR STUDIO 4 y WinAVR), el cual lo distribuye ATMEL completamente gratis y se puede descargar desde los siguientes links:

- http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
- http://winavr.sourceforge.net

El primer link es para descargar el AVR STUDIO4 y el segundo es para WinAVR. De todas formas, en la carpeta software del CD pueden encontrarse los instaladores. Una vez descargado, he instalado en el ordenador, es necesario instalar el paquete SP1. A continuación se instala el WinAvr. Este programa es un conjunto de herramientas de software libre para compilar, verificar y cargar programas escritos en lenguaje C a microcontroladores AVR de arquitectura RISC.

4.1.2 Crear nuevos proyectos.

Una vez instalado todo el software necesario para programar se ejecuta el AVR Studio. Al iniciarlo aparecerá en pantalla el asistente de proyectos. Si se presiona el botón "New Project" se podrá iniciar un nuevo proyecto. Si no aparece el asistente, podrá abrirse en "Project -> New project" del menú principal de AVR Studio.

En la ventana que aparece a continuación (ver Figura 2.1), se selecciona "AVR GCC" dentro del recuadro "Project type", se introduce el nombre del proyecto en "Project name". Si se quiere crear automáticamente un archivo fuente inicial, se selecciona la casilla "Create initial

file" y se introduce un nombre para el archivo (sin extensión) en "Initial file". Para poder crear una carpeta con el nombre del proyecto se tendrá que seleccionar la casilla "Create folder". Finalmente, presionando el botón "..." junto a "Location" se escoge un directorio para el nuevo proyecto. Se presiona "Next >>" para continuar a la siguiente pantalla.

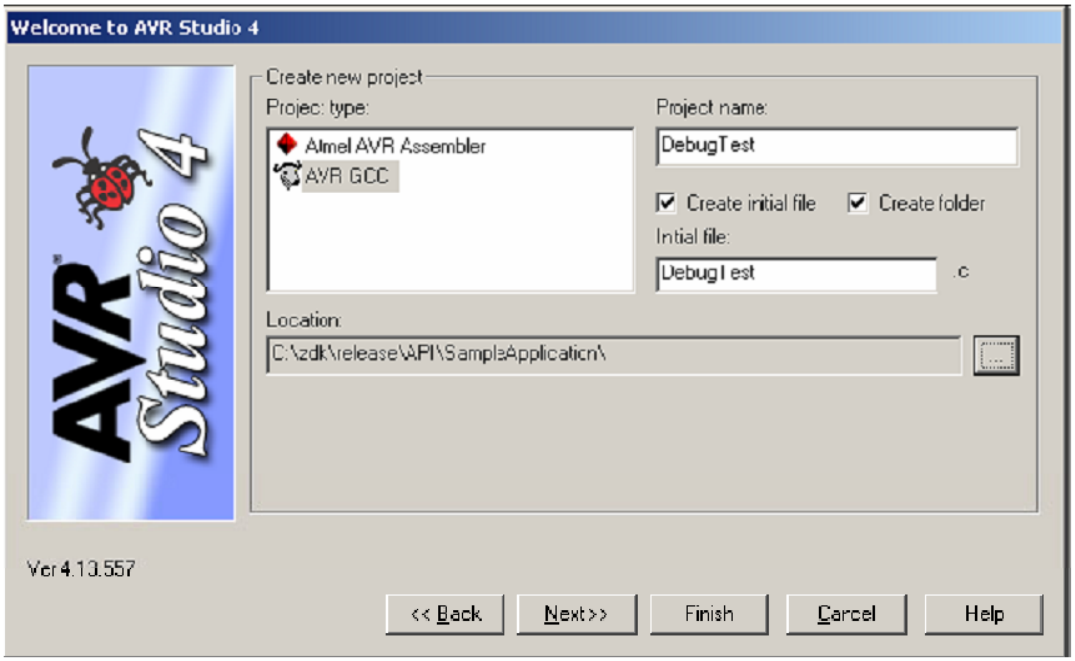


Fig 4.1 *Creación de un nuevo proyecto*

En la siguiente pantalla que aparece (ver Figura 3.2) te da la opción de escoger el programador a usar. En este proyecto se ha usado el siguiente: se escoge el "JTAGICE mkII" dentro de "Debug platform" y el "ATmega1281" dentro de "Device". Se pulsa "Finish" para cerrar el asistente.

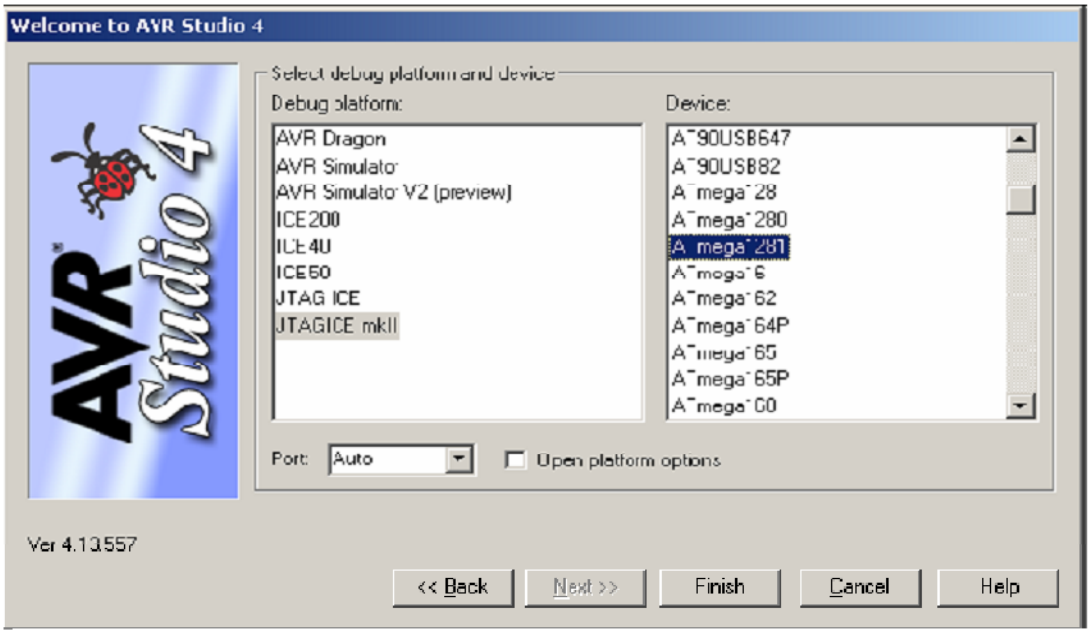


Fig 4.2 *Selección de la plataforma de depuración y del dispositivo*

Una vez instalado el AVR Studio e iniciado un nuevo proyecto, es necesario realizar la siguiente operación para poder usar la pila de software embebido BitCloud de MeshNetics. Primero se instala el paquete de software que ofrece este kit de investigación llamado "MeshNetics_ZigBit_Amp_Development_Kit_Complete(v2.3.0)". Se puede encontrar tanto en internet como en la carpeta de software del proyecto. Con este ejecutable se instalaría en el equipo un conjunto de herramientas, así como unos ejemplos de aplicación para las tarjetas y guías de utilización del kit, además de las librerías necesarias para el control de las tarjetas.
Una vez instalado bastaría solo copiar la carpeta "BitCloud" en la carpeta del proyecto iniciado. Esta carpeta se encuentra por defecto en "C:\Archivos de programa\MeshNetics\ZDK_Amp_ Complete\ BitCloud". De esta manera se dispondrá de todo un conjunto de librerías que más tarde serán útiles a la hora de crear las aplicaciones.

4.1.3 Uso del Makefile.

La colección de herramientas de AVR Studio requiere el uso del llamado makefile, un archivo de texto plano que se usará para llevar la gestión de la compilación de programas. El makefile está ordenado en forma de reglas, especificando qué es lo que hay que hacer para obtener un módulo en concreto y definir qué otros módulos, librerías, fuentes o programas son necesarios para construir un programa.

Para ahorrar tiempo y trabajo en la creación de futuras aplicaciones, es posible usar un makefile ya compilado. Para ello, debe introducirse en la carpeta del proyecto de la siguiente manera: Se abre el AVR Studio, dentro de "Project -> Configuration" en el menú, se selecciona la casilla "Use external makefile". Se pulsa el botón "..." junto al cuadro de texto y se escoge un makefile (normalmente localizado en la misma carpeta que el proyecto AVR Studio). Pulsando "OK" se cerrará la ventana.

Cada aplicación de ejemplo que ofrece este kit contiene un makefile especifico. Se ha comprobado que usando uno de ellos, concretamente de la aplicación "Blink", el cual se puede encontrar en el software del kit instalado, puede usarse como referencia para crear futuras aplicaciones. A continuación se muestra una sección del makefile extraído de la aplicación "Blink".

```
# Components path definition -------------------------
COMPONENTS_PATH = ../BitCloud/Components

# Application makerules including --------------------
include $(COMPONENTS_PATH)/../lib/MakerulesBcAll

# Project name ---------------------------------------
PRJ_NAME = NOMBRE DEL PROYECTO

# Application parameters -----------------------------
CFLAGS += -DBLINK_PERIOD=1000
CFLAGS += -DMIN_BLINK_PERIOD=100
CFLAGS += -DMAX_BLINK_PERIOD=10000
```

Fig 4.3 *Fragmento del makefile "Blink"*

Para hacer uso de este makefile es necesario colocar el nombre del proyecto dentro de la casilla que está en negrita y guardarlo en la carpeta del proyecto. Una vez hecho esto se siguen los pasos descritos anteriormente para usar este makefile. Una cosa a tener en cuenta de este makefile es que hace uso de una carpeta llamada “objs”, por lo que se tendrá que crear una nueva carpeta con el nombre "objs" en la carpeta de proyecto. Si no se hiciese daría problemas de compilación.

4.1.4 Construcción de la imagen.

A la hora de crear el programa, se puede escribir el código fuente en AVR Studio o mediante cualquier editor de texto apropiado. La estructura general del archivo es la misma que la que usaría en cualquier otro proyecto basado en C (cabeceras, definiciones, código).

Una vez se haya terminado de escribir el código, se puede construir la imagen ejecutable seleccionando "Build" del menú o presionando F7. Se puede hacer uso también del comando "Rebuild all" en proyectos largos para asegurar que se compilan todos los cambios del código. La ventana "Build" (en la parte inferior de la pantalla) mostrará la salida del compilador avr-gcc (ver Figura 3.3).

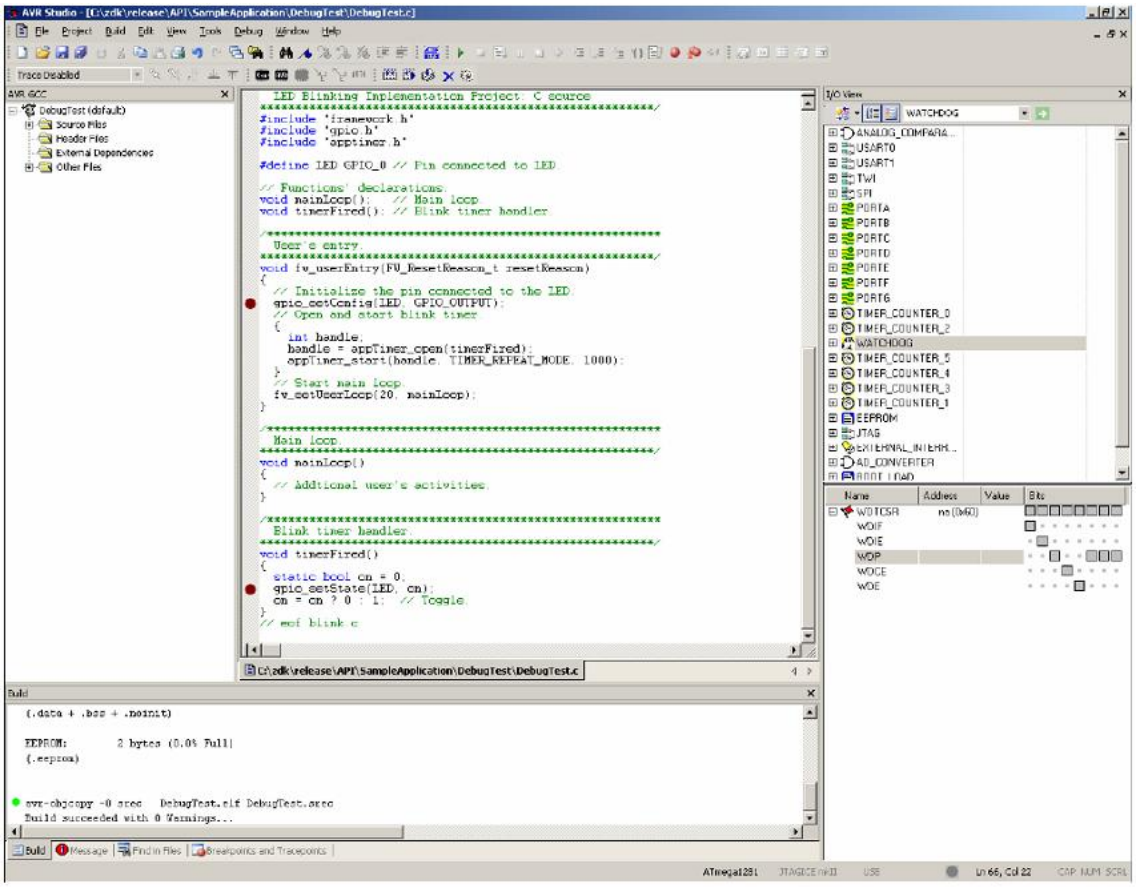


Fig 4.4 *Construcción de la imagen*

Si el código no contiene errores, se generará un archivo "DebugTest.elf" en la misma carpeta en la que se encuentra los archivos del proyecto (a menos que especifique un directorio de salida diferente en el makefile). Esta es la imagen ejecutable que utilizará el depurador. Además se generará otro archivo imagen de tipo “nombreproyecto.hex”. Esta última imagen es la que se usará para cargar el programa en la EEPROM de la tarjeta.

4.1.5 Depuración de la imagen.

Para comenzar la sesión de depuración para la imagen que acaba de construir, primero se conecta la tarjeta MeshBean al dispositivo JTAGICE y se alimenta los dispositivos (pueden ser alimentados por USB). Para probar las características de depuración de AVR Studio, se sitúa un breakpoint (pulse F9) en cualquier línea de código de la que esté seguro que va a ser ejecutada.

Ahora, se selecciona "Debug -> Start debugging" de la aplicación AVR Studio o se pulsa Ctrl-Alt-Shift-F5. Este comando se encuentra disponible únicamente después de ejecutar el comando "Build". Si se reinicia el AVR Studio, se deberá construir de nuevo la imagen. AVR Studio comenzará a programar el dispositivo con la imagen construida, indicando el progreso con la barra de progreso en la parte inferior de la ventana. Después de que la programación esté completa, saltará la siguiente ventana (ver Figura 3.4).

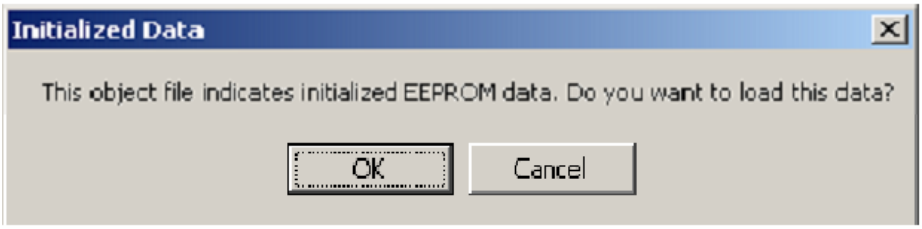


Fig 4.5 *Confirmación de carga de datos a la EEPROM*

EZeeNet utiliza una porción de la EEPROM para almacenar sus parámetros. Se pulsa el botón OK para continuar. En algunas ocasiones, AVR Studio interrumpirá su ejecución y mostrará la ventana de desensamblado con contenidos inválidos (ver Figura 3.5).

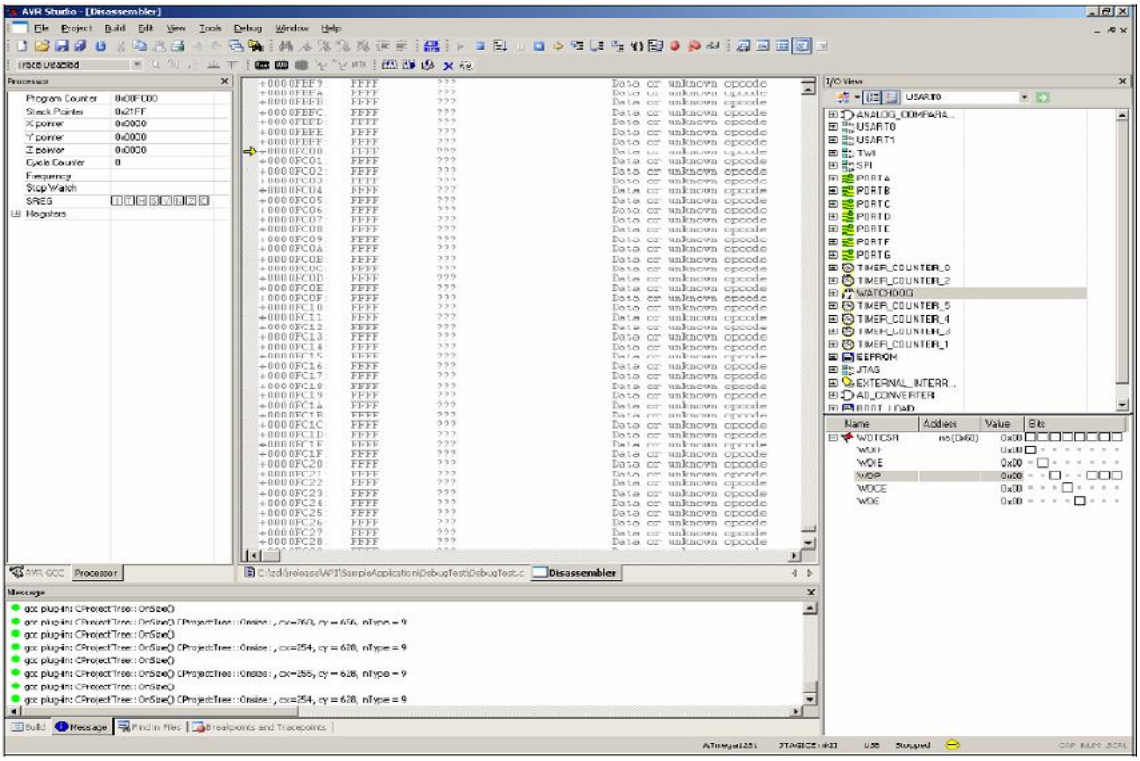


Fig 4.6 *Ventana de desensamblado*

Se cierra esta ventana y se presiona F5 para continuar la ejecución. AVR Studio pausará la ejecución en el primer breakpoint del código (ver Figura 3.6).

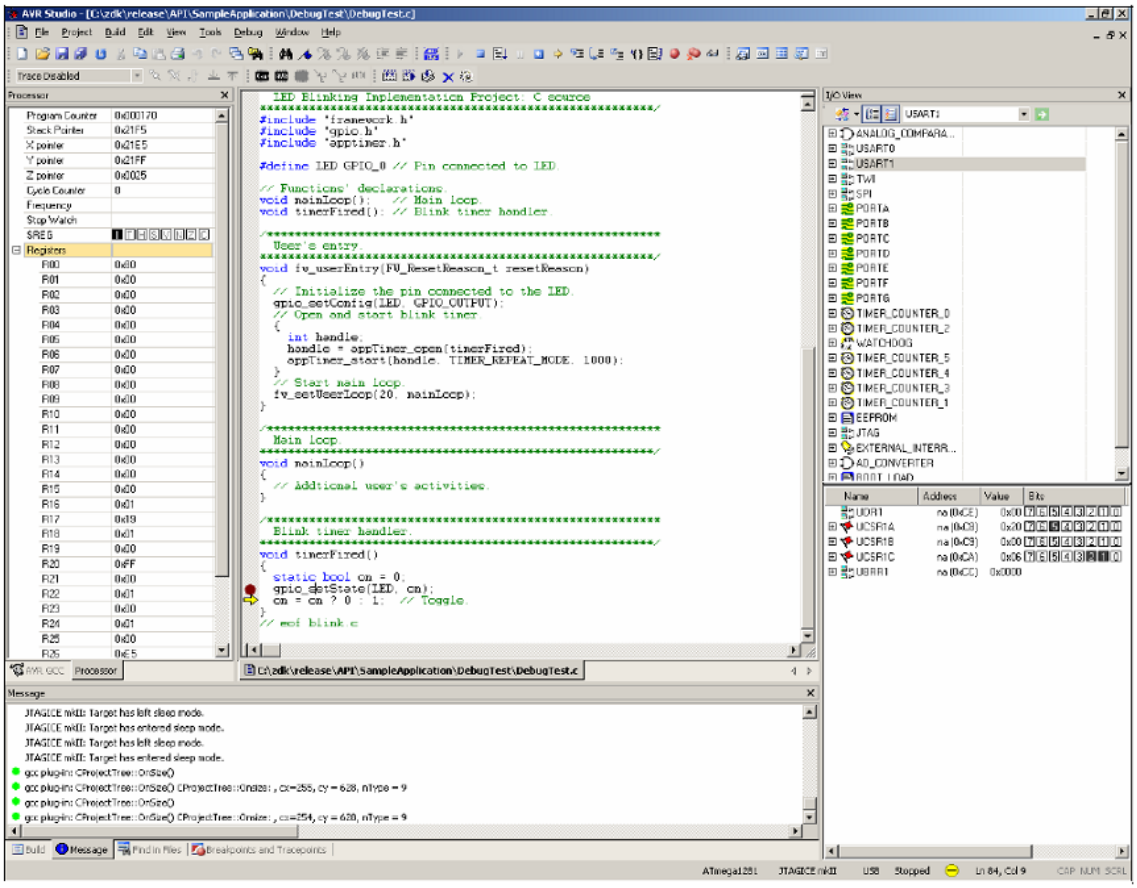


Fig 4.7 *Depurando*

En el menú View, se puede seleccionar ventanas adicionales de depuración: procesador, E/S, desensamblador, vigilancia de variables, memoria, registros, etc. Pulsando con el botón derecho del ratón en el editor de código, se podrá disponer también de unas pocas opciones de depuración, como añadir y eliminar breakpoints y expresiones de vigilancia.

Se continua con la ejecución en cualquier momento pulsando F5. Al pulsar Shift-F5, se reiniciará la sesión de depuración. Para detener el depurador, si se selecciona "Debug -> Stop debugging" del menú o se pulsa Ctrl-Shift-F5. Este comando está sólo disponible en modo stop, es decir, se deberá pausar primero la ejecución y después detenerlo completamente.

4.1.6 Carga de la imagen en las tarjetas.

Esta es una manera sencilla y rápida de cargar la imagen en la EEPROM de la tarjeta sin tener que depurar nuestro programa.

Se empieza por conectar primero el programador en el ordenador y la tarjeta al programador (ver Figura 3.7).

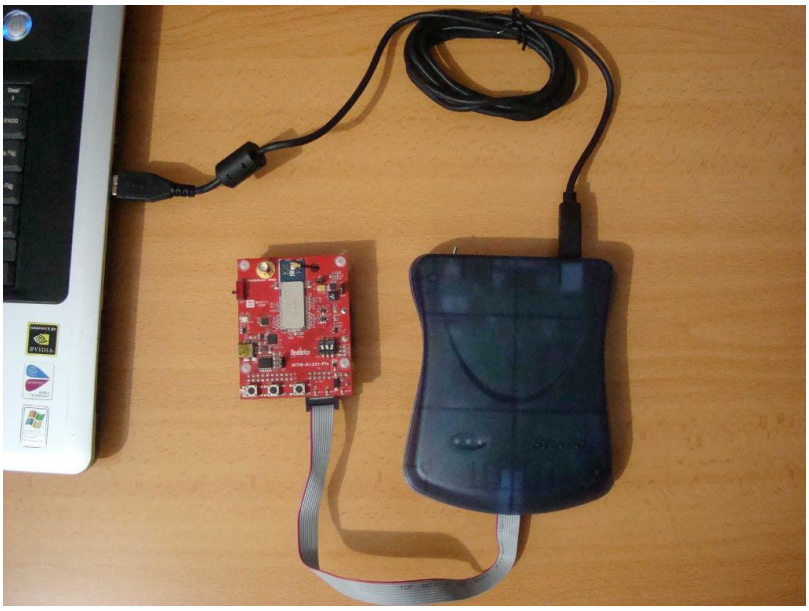


Fig 4.8 *Conexión del programador*

Hay que asegurarse que el jumper J3 está en modo de conexión RS 232 (ver Figura 3.7).

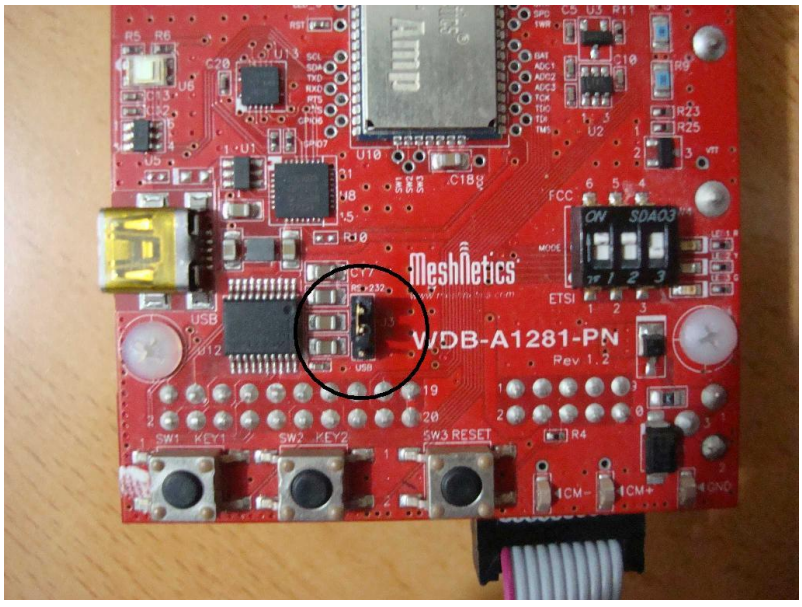


Fig 4.9 *Selección del Jumper J3 en modo RS 232*

Se enciende el programador y la tarjeta. Luego se pincha en AVR Studio "Tools -> Program AVR -> Auto Connect". Aparecerá en pantalla la siguiente ventana (ver Figura 3.8).

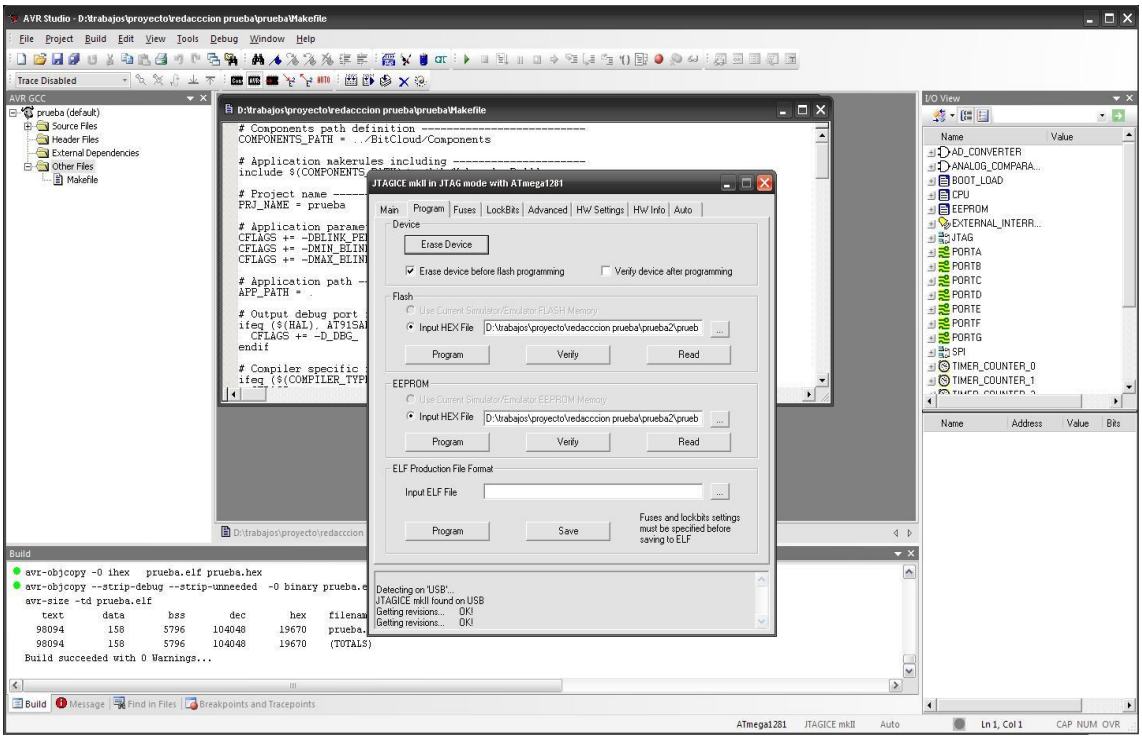


Fig 4.10 *Ventana del programador*

Se selecciona la pestaña "Main" y en la sección "Device and Signature Byte"s se escoge el programador que se usará. En esta caso es el modelo "ATmega1281".
Se pestañea en "Program" y en el recuadro encabezado con el nombre “Flash” se pulsa el botón "..." para seleccionar la imagen creada “nombreproyecto.hex”. Se pulsa "Program" y listo. Si no ha habido ningún error de escritura saldrá una verificación correcta en el recuadro inferior.

4.2 Librerías esenciales para compilar.

Una vez aprendido como se usa el programa y las formas que tiene de cargar la imagen en la tarjeta se mostrará a continuación las librerías esenciales para manejar la tarjeta.. Solo se analizará aquellas usadas por las aplicaciones que se crearán a continuación. No obstante, existe una herramienta que se puede encontrar en la documentación del software instalado de MeshNetics. Esta herramienta responde al nombre de "BitCloud Stack Documentation".

4.2.1 Libreria "zdo.h".

La primera que se estudiará es “zdo.h”. Esta librería proporciona un lote de APIs Device Object completamente conforme con ZigBee para habilitar el control de la red (comenzar, reinicio, formación, unión), y de la alimentación (dormir, despertar). También ddefine los tipos de Perfiles de Dispositivo, y los comandos de descubrimiento de de dispositivos y servicios. Proporciona APIs para reconocimiento y transmisión unicast, multicast, y broadcast. Por ello es necesario e imprescindible incluir esta librería junto con las siguientes funciones:

```
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams) {
        nwkParams = nwkParams;
}
void ZDO_WakeUpInd(void) {}
```

La primera de ellas indica los parámetros de actualización de red. La función debe ser definida por una aplicación. Se llama cuando el dispositivo ha resuelto el conflicto de "adress/panid" o cuando se reincorporó a la red. Tiene como parámetro "nwkParams" y describe el motivo de la actualización de la red y los nuevos parámetros. La segunda indica que una aplicación de temporizador de apagado automático ha disparado. Si está en modo dormir, se despierta y está listo para trabajar. La aplicación puede continuar sus actividades.
Viendo el significado de cada función parece que en la aplicación no serían necesarias, pero está comprobado que si no se colocasen, daría error de compilación. Es posible que el motivo sea por el Makefile que se está usando, o que por estar trabajando bajo la pila Zigbee, son unas funciones indispensables.

4.2.2 Librería "Task Manager.h".

La siguiente librería "Task Manager.h" proporciona una API para programar tareas en una cola con prioridad, optimizada para entornos de pila ZigBee multicapa y demandas de protocolos críticos en el tiempo. Las funciones que se encuentran en esta librería son:

- Void APL_TaskHandler (void): Esta función es el “manejador de tareas”. Procesa las tareas de la aplicación. Su primera función se destina solo para establecer los parámetros en la pila. Para poder iniciar una nueva tarea se debe de utilizar. Es decir, esta función equivaldría a la función main () de cualquier aplicación. Realmente no haría falta usar esta función para este sencillo programa, pero se aplicará para ver su funcionamiento.
- Void SYS_PostTask (SYS_TaskId_t taskId): es una función de control de flujo que entra en el código de las aplicaciones a través del manejador de tareas. Es decir, sirve para dar prioridad a ciertas llamadas en nuestro proceso. En la aplicación se utilizará esta función como un bucle. Al poner APL_TASK_ID entre los paréntesis comenzará de nuevo por el manejador de tareas. El controlador de flujo de tarea se ejecuta en la aplicación sólo cuando todas las tareas de mayor prioridad se han completado.

4.2.3 Librería "leds.h".

Esta librería es la que gestiona el uso de los leds, llamada leds.h. De esto hay poco que decir. Las funciones que tiene esta librería son muy simples:

- BSP_OpenLeds (void): Abre el módulo de leds para su uso.
- BSP_CloseLeds (void): Cierra el módulo de leds.
- BSP_OnLed (uint8_t id): Pone en uso el led que se encuentra entre paréntesis. En la tarjeta se dispone de tres leds de colores rojo, amarillo y verde, de los cuales el nombre sería LED_RED, LED_GREEN y LED_YELLOW respectivamente.
- BSP_OffLed (uint8_t id): Apaga el led que se encuentra entre paréntesis.
- BSP_ToggleLed (uint8_t id): Cambia al estado opuesto del led que se encuentra entre paréntesis. Pasa a encender el led si está apagado e inversamente.

4.2.4 Librería "adc.h".

Se trata de la librería más interesante a estudiar, pues es la encargada de gestionar el periférico "ADC" por el que se conectarán los sensores analógicos. A continuación se mostrarán las funciones esenciales para su control:

- HAL_AdcParams_t: Describe la estructura de los parámetros de la interfaz ADC. En esta función se puede establecer la resolución de conversión en 8 o 10 bit; la frecuencia de muestreo (simple rate) en 4800 sps, 9600 sps, 39 ksps, etc; el voltaje de referencia AREF , AVCC, INTERNAL_1d1V o INTERNAL_2d56V. Las dos primeras son voltajes de

referencia externas y las otras internas. Se cuenta también con un puntero al dato de la aplicación (BufferPointer), asi como la cantidad de muestras que se realizarán (selecctionsAmount) y un callback en la función.

- HAL_OpenAdc_t(HAL_AdcParams_t *param): Función que iniciará el Adc con los parámetros definidos en HAL_AdcParams_t.
- HAL_ReadAdc_t(HAL_AdcChannelNumber_t channel): Esta función se encargará de leer el canal que está definido entre paréntesis. Hay que recordar que solo se dispone de tres canales Adc mas el de la batería.
- HAL_CloseAdc_t(void): Cierra el Adc.

4.2.5 Librería "uart.h".

Para la conexión con el ordenador se debe de incluir una librería que permitiese esta acción .Esta librería permite el control del puerto de la placa para una conexión serie con el bus de datos. Las funciones que se usarán son las siguientes:

- HAL_UartDescriptor_t: Describe los parámetros de la estructura de la interfaz de Uart. En esta función se establece el canal de Uart (chanel 0 o chanel 1), el modo (mode), es decir, si será síncrona o asíncrona, la tasa de baudios (baudrate), el control de flujo (flowControl), la longitud del dato (dataLength), bit de paridad (parity), el bit de parada (stopbits) y todo un conjunto de parámetros que caracterizan los datos transmitidos y recibidos (rx y tx).
- HAL_OpenUart (HAL_UartDescriptor_t *descriptor): Esta función realiza la configuración de los registros de Uart y de la RTS, CTS y los pines DTR con los parámetros establecidos en UartDescriptor por medio de un puntero a dicha función.
- HAL_CloseUart (HAL_UartDescriptor_t *descriptor) : Libera el canal y los pines de Uart.
- HAL_WriteUart (HAL_UartDescriptor_t *descriptor, uint8_t * buffer, uint8_t length): Escribe un número de bytes al canal Uart. La función txCallback se utilizará para notificar cuando haya finalizado la transmisión. Hace un puntero a la estructura UartDescriptor_t y al buffer del dato de la aplicación, e indica la cantidad de bytes de transferencia.
- HAL_ReadUart (HAL_UartDescriptor_t *descriptor, uint8_t * buffer, uint8_t length): Lee un número de bytes de la UART y los coloca en el búfer. La estructura interna es la misma que la función anterior.

4.3 Aplicaciones de prueba desarrolladas.

Una vez aprendido como se usa el programa y conocido las librerías útiles a utilizar, ya se está preparado para hacer la primera aplicación. A continuación se mostrarán unas sencillas aplicaciones para las tarjetas con el objetivo de familiarizarse con las librerias y así llegar a la cuestión principal, que es conseguir conectar un sensor por el canal "ADC" y saberlo gestionar.

4.3.1 Parpadeo Leds.

El primer programa que se hará va a tratar del uso de las librerías básicas de esta pila. En él se logrará hacer parpadear los led que se encuentran en la tarjeta.

Se crea un nuevo proyecto como se ha descrito anteriormente con el nombre “Parpadeo_led”. Hay que recordar que en la carpeta donde se guarden las aplicaciones creadas debería de encontrarse la carpeta Bitcloud con todas sus librerías. En esta aplicación se usará el Makefile mostrado anteriormente. Para ello debe colocarse el nombre de la aplicación dentro del Makefile como el trozo de código que se muestra a continuación y listo (ver Figura 3.9).

```
# Project name ------------------------------------------------------
PRJ_NAME = Parpadeo_led
```

Fig 4.11*Trozo del Makefile de Parpadeo_led*

Se guarda en la carpeta que se ha generado al crear el proyecto. Se exporta el Makefile con el procedimiento descrito anteriormente y listo. Faltaría por crear la carpeta “objs” para que no diera problemas de compilación.

Se procederá ahora a escribir las primeras líneas de código en la ventana de proyectos. Se escribe las librerías que se utilizarán en esta aplicación. A continuación se muestra el código del programa (ver Figura 3.10).

```
#include "leds.h"
#include "zdo.h"
#include "taskManager.h"
static int led = LED_GREEN;
static int count = 1000;
void APL_TaskHandler(void)
{
      BSP_OpenLeds();
      BSP_OnLed(led);
      if(--count==0){
            count = 1000;
            BSP_OffLed(led);
            if (led==LED_GREEN) led = LED_RED;
            else led = LED_GREEN;
      }
      SYS_PostTask(APL_TASK_ID);
}
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams){
      nwkParams = nwkParams;
}
void ZDO_WakeUpInd(void){}
```

Fig 4.12 *Código de la aplicación Parpadeo_led*

Se ha creado una variable "count" de valor 1000 que hará de temporizador para poder apreciar mejor el parpadeo. Se puede observar que después de inicializarse el módulo de leds comienza a encenderse el led verde y hasta que no se termina el temporizador no se encenderá el led rojo. Después de haber hecho un ciclo se ejecuta la función SYS_PostTask y se inicia de nuevo la función APL_TaskHandler como si se tratara de un bucle.

Las funciones ZDO_MgmtNwkUpdateNotf y ZDO_WakeUpInd como se ha explicado antes, no tienen ninguna función especial en este programa, pero tal como se ha desarrollado el Makefile, son necesarias incluirlas, por el contrario daría error de compilación.

Se ha visto que con esta simple aplicación es muy sencilla la utilización de funciones para manejar el entorno de la tarjeta.

4.3.2 Lectura de la batería.

Se ha visto que con esta simple aplicación es muy sencilla la utilización de funciones para manejar el entorno de la tarjeta. Se prosigue ahora a realizar una aplicación en la que incluya más elementos de la tarjeta como la batería. Como se sabe, la tarjeta tiene un cargador de baterías en la parte posterior de dos pilas AA en la que se puede recoger datos de la misma por medio de una entrada del slot de 20 pines al convertidor analógico/digital. Al igual que la

tarjeta tiene tres entradas ADC, existe otra por la cual ya está integrada en la tarjeta que conecta directamente con la batería. De esta manera, si se consigue controlar el ADC de la batería, se sabrá manejar los sensores que más adelante se conectarán a la tarjeta por medio de este sistema. La tarjeta tiene una ranura de expansión que soporta las interfaces UART, I2C, SPI, ADC, GPIO, IRQ, etc. Por su simplicidad, se usará solo la interfaz ADC.

La función del siguiente programa es utilizar los leds con el motivo de indicar el estado de carga de la batería. A continuación se muestra el código:

```
#include "adc.h"
#include "leds.h"
#include "zdo.h"
#include "taskManager.h"

static HAL_AdcParams_t adcParam;//definimos adcParam de tipo
HAL_AdcParams_t
static uint8_t batteryData;
static void batteryCallback();
static int count = 100;

void APL_TaskHandler(void)
{
    BSP_OpenLeds();
    adcParam.bufferPointer = &(batteryData);
    adcParam.callback = batteryCallback;
    adcParam.resolution = RESOLUTION_8_BIT;
    adcParam.sampleRate = ADC_4800SPS;
    adcParam.selectionsAmount = 1;
    adcParam.voltageReference = AREF;
    HAL_OpenAdc(&adcParam);
    HAL_ReadAdc(HAL_ADC_CHANNEL0);
}

static void batteryCallback(){
      if (--count==0){
            count = 100;
            if (batteryData>170){
                  BSP_ToggleLed(LED_GREEN);
            } else if ((batteryData<170)&&(batteryData>85)){
                  BSP_ToggleLed(LED_YELLOW);
            } else if (batteryData<85){
                  BSP_ToggleLed(LED_RED);
            }
      }
      HAL_CloseAdc();
      SYS_PostTask(APL_TASK_ID);
}

void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams){
      nwkParams = nwkParams;
}

void ZDO_WakeUpInd(void)
{
}
```

Fig 4.13 *Código de la aplicación "Lectura de batería"*

Los parámetros que se han establecido son por defecto para una lectura correcta de la batería. El proceso del programa es el siguiente: Se abre el canal 0 que corresponde a la batería con los parámetros que se han establecido. Al leer el canal, el dato se guarda en la variable batteryData y más adelante hace un callback a la función batteryCallback. En este algoritmo hay un contador que mantendrá encendida el led correspondiente durante un tiempo determinado. Como el dato que se recibe es de 8 bits, se ha estimado para un valor inferior a 85 (un tercio de 256) como batería baja, con lo que se encendería el led rojo. Para un valor entre 85 y 170 sería batería media, que corresponde con el led amarillo. Y para un valor mayor de 170 sería para una batería llena, con el led verde.

Se ha observado con este programa lo sencillo que es obtener datos del puerto Adc. Si se quisiera leer directamente desde un sensor bastaría solo conectarlo al puerto de expansión y seleccionar el canal correspondiente.

4.3.3 Envio de datos.

Hasta ahora se ha visto como es posible el control de una parte de las tarjetas como son los leds. También se ha logrado leer el estado de la batería por medio de un canal "ADC". Pues bien, la siguiente aplicación trataría precisamente de conectar un sensor al puerto y recibir el dato por la pantalla del ordenador. Para la conexión con el ordenador se debe de incluir la librería "uart.h" vista anteriormente.

El sensor que se le va a conectar es un simple potenciómetro variable en el canal 3 del puerto de expansión que corresponde al pin nº 18. El potenciómetro se alimenta por el pin nº 9 que proviene de la batería y el GND del pin nº 5. Se ha usado el cable que viene con el kit. El circuito es un simple divisor de tensión. A continuación se muestra el esquema eléctrico y una imagen del sensor con la conexión de la tarjeta.

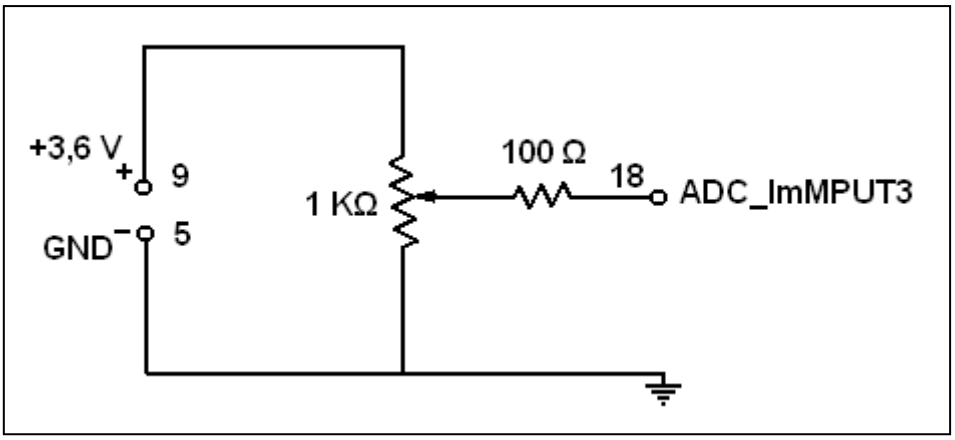


Fig 4.14 *Esquema eléctrico*

Fig 4.15 *Montaje del sensor*

Como se observa, el conector es lo que se introduce en la ranura de expansión de la tarjeta. A continuación se expone el código de la aplicación:

```
#include "uart.h"
#include "adc.h"
#include "leds.h"
#include "zdo.h"
#include "taskManager.h"

static HAL_AdcParams_t adcParam;
static uint8_t sensorData;
static void sensorCallback();
static int count = 1000;

static HAL_UartDescriptor_t uartDescriptor;
static void readByteEvent(uint8_t readBytesLen);
static void writeConfirm();
static char buffer[255];

void APL_TaskHandler(void)
{	BSP_OpenLeds();
	uartDescriptor.tty		= UART_CHANNEL_1;
	uartDescriptor.mode		= ASYNC;
	uartDescriptor.flowControl	= UART_FLOW_CONTROL_NONE;
	uartDescriptor.baudrate		= UART_BAUDRATE_38400;
	uartDescriptor.dataLength	= UART_DATA8;
	uartDescriptor.parity		= UART_PARITY_NONE;
	uartDescriptor.stopbits		= UART_STOPBIT_1;
	uartDescriptor.rxBuffer		= NULL;
	uartDescriptor.rxBufferLength	= 0;
	uartDescriptor.txBuffer		= NULL;
```

```
uartDescriptor.txBufferLength       = 0;
            uartDescriptor.rxCallback           = readByteEvent;
            uartDescriptor.txCallback           = writeConfirm;
            HAL_OpenUart(&uartDescriptor);
           adcParam.bufferPointer     = &(sensorData);
           adcParam.callback                    = sensorCallback;
            adcParam.resolution                 = RESOLUTION_8_BIT;
            adcParam.sampleRate                 = ADC_4800SPS;
            adcParam.selectionsAmount           = 1;
            adcParam.voltageReference           = AREF;
            HAL_OpenAdc(&adcParam);
            HAL_ReadAdc(HAL_ADC_CHANNEL3);
}
static void readByteEvent(uint8_t readBytesLen){
}
static void writeConfirm(){
}
static void sensorCallback(){
            if (--count==0){
                        count = 1000;
                        if (sensorData>170){
                                    BSP_OffLed(LED_YELLOW);
                                    BSP_OffLed(LED_RED);
                                    BSP_ToggleLed(LED_GREEN);
                        } else if ((sensorData<170)&&(sensorData>85)){
                                    BSP_OffLed(LED_GREEN);
                                    BSP_OffLed(LED_RED);
                                    BSP_ToggleLed(LED_YELLOW);
                        } else if (sensorData<85){
                                    BSP_OffLed(LED_YELLOW);
                                    BSP_OffLed(LED_GREEN);
                                    BSP_ToggleLed(LED_RED);
                        }
                        sprintf(buffer, "Sensor=%d\n\r",sensorData);
                        HAL_WriteUart(&uartDescriptor, &buffer, sizeof(buffer));
            }
            HAL_CloseAdc();
            SYS_PostTask(APL_TASK_ID);
}
void ZDO_MgmtNwkUpdateNotf(ZDO_MgmtNwkUpdateNotf_t *nwkParams){
            nwkParams = nwkParams;
}void ZDO_WakeUpInd(void) {}
void ZDO_SleepInd(void) {}
```

Fig 4.16 *Código de la aplicación "Envío de datos"*

Después de cargar el programa se conecta al PC por medio del cable USB. Hay que recordar de tener el jumper J3 en la posición adecuada para la conexión serie con la tarjeta. Solo faltaría ejecutar el hyper Terminal de Windows y configurar la conexión como se muestra en la siguiente figura.

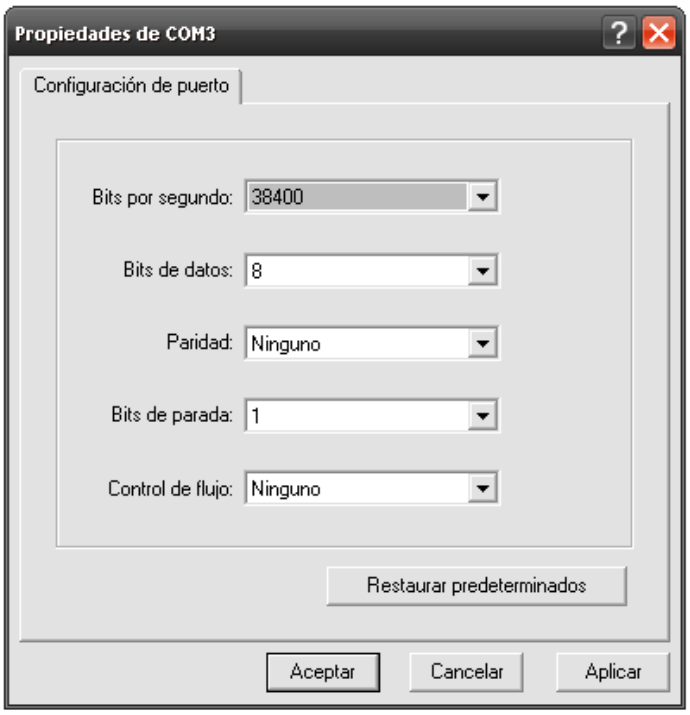


Fig 4.17 *Configuración de conexión*

Por último se muestra un ejemplo de los datos recogidos del sensor (Ver Figura 3.15).

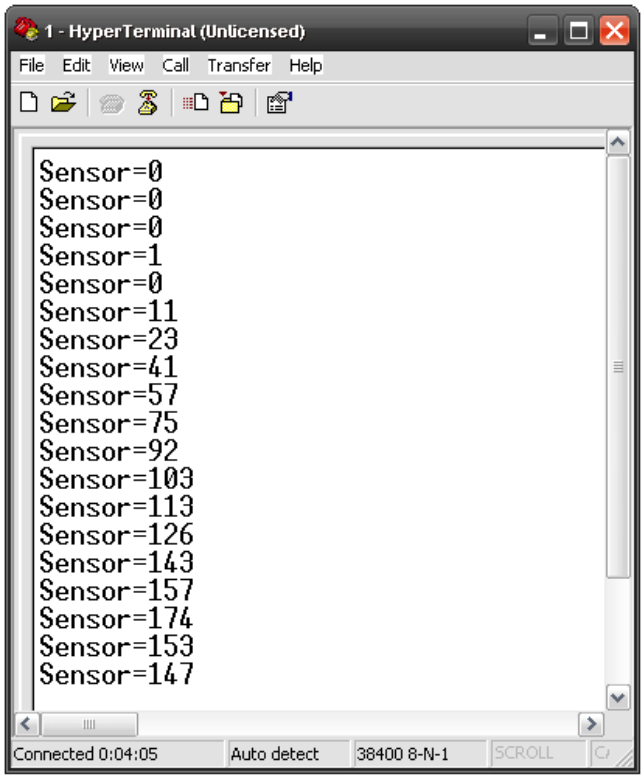


Fig 4.18 *Hyper Terminal de Windows*

De esta manera se observa lo sencillo que es enviar datos por el puerto de comunicación. Pero para que sea realmente útil y necesario para este proyecto es una comunicación wireless con la tarjeta y el ordenador.

4.4 Estudio del código implementado.

A continuación se hará un pequeño análisis sobre el código basado en el protocolo ZigBee que maneja las tarjetas. Este programa se compone principalmente por varios archivos estructurado en cinco códigos:

- WSNDemoApp.c.
- WSNCoordinator.c.
- WSNRouter.c.
- WSNEndDevice.c.
- WSNSensorManager.c.
- WSNUARTManager.c.

Se pueden encontrar en la siguiente dirección C:\Archivos de programa\MeshNetics\ ZDK_Amp_Complete\Sample Applications\WSNDemo. Todo el código está escrito en un lenguaje basado en programación orientado a eventos.

4.4.1 WSNDemoApp.c.

Este sería el código principal del programa, el cual se encargaría de configurar y gestionar prácticamente toda la tarjeta, así como crear una red de comunicación entre los dispositivos.

De las librerías que están incluidas, faltaría por comentar que configServer.h describe la interfaz pública y los parámetros de configuración del servidor. Básicamente se trata de una interfaz destinada a la escritura y lectura de los parámetros de la pila. Se pueden redefinir los parámetros a través del fichero MakeFile o a través de las variables de entorno. Por otro lado se tiene la librería aps.h que describe la interfaz de la pila pública APS, que como se sabe, proporciona el nivel más alto de la creación de redes relacionadas con la API. Y por último, tanto la librería sliders.h como buttons.h, gestionan los interruptores y botones respectivamente.

Una vez definido todas las variables y las funciones, se hace a la llamada a la función principal que gestiona toda la aplicación app_TaskHanler.

Por la inicialización de la variable appState, comenzaría con el estado APP_INITING_STATE.
En este estado se inicializarán los dispositivos. Sus funciones serán:

- Configura el PANID de la red y lo guarda en appMessage.
- Inicializa los botones
- Lee los interruptores. Dependiendo de su posición, detecta si es coordinador, router o dispositivo final y lo guarda en appMessage.
- Configura el ExtAddr , que se trata de la dirección extendida del nodo, y lo guarda en appMessage.
- Se lee el identificador único CS_UID_ID que servirá para la configuración del servidor y se guarda en un campo del mensaje.
- Se desactiva la función dormir en el coordinador y en el router, y se activa en el dispositivo final.
- Se habilitan los leds.
- Si el dispositivo es coordinador se habilita la UartManager.
- Se configuran los parámetros fijos del mensaje. Entre ellos se encuentra sensorsize que será importante a la hora de añadir más datos de sensores al mensaje de envío. Se trata

de una función que utiliza un sizeof que calcula el espacio de memoria que ocuparán los datos de los sensores en el mensaje de envío.

- Se inicia el temporizador de inicio de red y comienza el parpadeo del led rojo.
- Cambia el valor de la variable appState a APP_STARTING_NETWORL_STATE y se vuelve a app_TaskHanler, esta vez dejando paso al estado nombrado y comenzando por el evento APP_PROCESS.

En este estado se inicia la red de comunicación y se realizan las siguientes funciones:

- Se activa el temporizador del parpadeo de los leds.
- Se confirman los parámetros de red y se ejecuta un callback a una función del sistema que inicia la red para el coordinador y se une con el router. A continuación se produce otro callback que lo lleva al evento APP_NETWORK_STARTING_DONE donde se indica que si la conexión ha sido realizada, se realicen las siguientes funciones:
- finalizar el temporizador del parpadeo.
- Llamar a la variable de estado APP_IN_NETWORK_STATE.
- Encender el led rojo.
- Guardar los parámetros de red en el mensaje.
- Y por último iniciar las subrutinas particulares de cada dispositivo dependiendo si es coordinador, router o dispositivo final. Estas subrutinas se comentarán en su código correspondiente.

Al ejecutarse el break, se sale del evento y comienza en el estado "APP_IN_NETWORK_STATE". A partir de este estado comienza la subrutina correspondiente a cada dispositivo, ejecutando el estado "APP_PROCESS" del TaskHandler de todos los códigos.

Falta por comentar que existe un estado llamado" APP_NETWORK_STATE_UPDATED" el cual actualiza el estado de la red de comunicación y si ésta falla vuelve a restablecer la comunicación.

4.4.2 WSNCoordinator.c.

Este código va dirigido al dispositivo que se ha establecido como coordinador, el cual no se ejecutará hasta que la comunicación entre los dispositivos sea correcta. El código comienza por el estado INITIAL_DEVICE_STATE como se ha comentado antes. Aquí hace las siguientes tareas:

- Para el temporizador del dispositivo, lo configura y ejecuta el callback de la función, el cual ejecuta el evento APP_TIMER_FIRED. Aquí su única función es llamar a la variable de estado appDeviceState en READING_SENSORS_STATE. En este estado se hacen las siguientes tareas:
- Obtener los valores de Lqi y Rssi, que determinan la calidad del enlace y se guarda en el mensaje.
- Inicia el SensorManager, obtiene los valores de los sensores y se guarda en el mensaje. Estas función se encuentra definida en WSNSensorManager.c y se explicarán más adelante.

Una vez leído los sensores y guardado los datos, desde WSNSensorManager.c se ejecuta un callback que inicia el estado "APP_READING_DONE":

- Cambia la variable de estado appDeviceState a "SENDING_DEVICE_STATE".
- Cierra la lectura de los sensores y el SensorManager.

Una vez hecho esto se pasa al estado "SENDING_DEVICE_STATE" que es donde se inicia el envío del mensaje. Las tareas son:

- Parpadeo del led verde.
- Envío del mensaje por la uart.
- Cambio de la variable estado appDeviceState a "STARTING_TIMER_STATE", donde se inicia de nuevo el temporizador del dispositivo.

Una vez terminado todo este proceso, el coordinador está a la espera a que se ejecute el estado "WAITING_DEVICE_STATE" que será llamado por los dispositivos router y dispositivo final y comenzará un nuevo ciclo de lectura de sensores y envío del mensaje.

4.4.3 WSNRouter.c.

Este código va dirigido al dispositivo que se ha establecido como router y como sucede en el coordinador, no se iniciará hasta que la comunicación sea correcta. Aquí comienza por una función de inicialización llamada desde el programa principal cuya tarea es ejecutar el estado REINITIAL_DEVICE_STATE si el dispositivo se encuentra en estado de espera. De un modo u otro se ejecuta el estado INITIAL_DEVICE_STATE. Esta parte es prácticamente igual que en el coordinador, con la diferencia que en este momento se rellenan unos parámetros del mensaje request, entre los que se encuentra los datos de los sensores, el nodo que corresponde, el tipo de dispositivo, el short adress... Los siguientes procesos son también iguales:

- Inicio de los sensores
- Lectura y escritura en el mensaje.

Pero esta vez el mensaje no se envía por medio de la uart, sino que se hace mediante una función del sistema que envía el mensaje de los parámetros definidos anteriormente a otro nodo por la red. Una vez realizado esto, por medio de un callback, activa una función de confirmación del sistema que cambia al estado APP_SENDING_DONE. En este estado se comprueba si se ha enviado bien el mensaje. Si no ha sido así, se vuelve al estado inicial para una nueva ejecución de procesos. Este estado acaba por apagar el led verde y pasar al estado WAITING_DEVICE_STATE a la espera de una nueva orden.

4.4.4 WSNEndDevice.c

Este código pertenece al dispositivo final. Aquí comienza igual que en los demás, por la subrutina ejecutada desde el programa principal en donde nos lleva al estado inicial, el cual pasa al dispositivo en modo de suspensión, rellenan los parámetros del mensaje request como en el router y procede a activar los sensores, leer los datos de ellos y guardarlo en el mensaje request. El proceso es el mismo que en el router, pero con la diferencia que el dispositivo pasa al estado dormir una vez enviado el mensaje al nodo, y se despierta cuando los temporizadores lo indican, continuando con las actividades. Se sabrá en qué estado se encuentra al despertar por los leds. Si se encienden todos es que se encuentra en el estado inicial y si es solo el led rojo, significa que está en el estado de red y a continuación se pone a leer los sensores.

4.4.5 WSNSensorManager.c

Este código va dirigido a la gestión de los sensores. En él se podrá abrir los sensores, recibir los datos de los mismos y escribirlo en el mensaje que más tarde se enviará. Esta vez no se trata de una programación basada en eventos, sino que mas bien es una serie de funciones que se ejecutan en un momento dado por los demás códigos. Esas funciones son las siguientes:

- "Void appStartSensorManager()": Inicia los sensores. Esta función es necesaria ejecutarla antes de una lectura de los sensores.

- "Void appGetSensorData(void (*sensorsesGot)(void))": En esta función es un conjunto de callbacks que va obteniendo el dato de cada sensor y lo guarda en el mensaje sucesivamente. Comienza por hacer un callback a la función de donde se obtiene el valor de la temperatura y lo guarda. Este a su vez hace un callback para la obtención del dato de la luz y finalmente se vuelve a hacer otro callback para recoger la lectura de la batería. Por último se ejecuta el callback que se encuentra entre paréntesis de esta función que confirma a los dispositivos la finalización de la lectura de los sensores para proseguir con la ejecución de los estados, concretamente el estado APP_READING_DONE, como se había comentado antes en el coordinador.
- "Void appStopSensorManager()": Cierra la gestión de los sensores.

Como se observa, es muy sencillo la obtención de los datos de los sensores. Si se quisiera añadir nuevos sensores debería gestionarse dentro de este código.

4.4.6 WSNUARTManager.c

Este código corresponde a la gestión de envío de mensajes que provienen desde el coordinador al PC por medio de la interfaz uart. El código comienza por definir la estructura de las siguientes funciones:

- "UartMessage_t": está compuesto por un array con un tamaño en correlación al mensaje de envío y una variable definida como size que indicará el tamaño del mensaje.
- wsn2uart: Compuesto por una variable "UartMessage_t" llamada "uartMessageQueue [MAX_UART_MESSAGE_QUEUE_COUNT]" que indica la capacidad máxima de la cola de mensajes que se enviarán por la uart. Un booleano que indicará si ya no hay mas mensajes de envío, y unas variables definidas como la cabeza, talla y tamaño del mensaje.

Esta programación también se trata de unas funciones que se ejecutan en un momento concreto por el coordinador. A continuación se muestra cada una de ellas:

- "Static void readByteEvent(uint8_t readBytesLen)": muestra el recuento de bytes recibidos por la Uart. Se guarda en la variable "readBytesLe".
- "static void writeConfirm()": Indica que se ha recibido la confirmación de escritura en la Uart y está listo para enviar un nuevo mensaje.
- "static void sendNextMessage()": Envía el siguiente mensaje de la cola. Es ejecutado por la función "writeConfirm".
- "appStartUARTManager()": Se ejecuta para iniciar la interfaz Uart con unos parámetros definidos que describen la estructura de la interfaz. Dentro se encuentran dos callbacks que ejecutan las funciones "readByteEvent" o "writeConfirm" si se han recibido o transmitido datos por la uart.
- "void appSendMessageToUart(AppMessage_t *newMessage)": Esta función se encarga de poner el mensaje en la cola de envío. Previamente modifica el mensaje colocando unos bytes de inicio de trama y final de trama.

4.5 Añadir un nuevo sensor.

Una vez entendido el código de este programa principal se procederá a la investigación de añadir un nuevo sensor. Como se ha observado en el código, el único archivo que gestiona la parte de los sensores es WSNSensorManager.c. Por lo tanto es el único archivo que se modificará. Pero además, existe otro archivo que no se ha tenido en cuenta, donde sus funciones son inicializar todas las librerías del código, especificar el periodo de envío de

mensajes, además de darle una estructura definida a los mensajes que se crean en las tarjetas para ser enviadas entre otras funciones. Se trata de un archivo llamado "WSNDemoApp.h".

El sensor que va ser utilizado será el que se ha visto anteriormente, por lo tanto, la manera que se obtendrán datos de él será de la misma manera vista anteriormente. En un principio se consiguió cambiar el valor que se obtiene de un sensor del dispositivo por el del potenciómetro. Existían problemas con el cambio del sensor de batería por el del potenciómetro, debido a unos conflictos que tenían lugar a la hora de obtener una lectura de la batería junto con el potenciómetro. Esto ocurría porque el sensor de la batería tiene dos formas de recoger datos. Por un lado se puede obtener mediante llamadas al sistema BSP, y por otro lado mediante la interfaz adc. Esta última manera se comentó anteriormente en la aplicación “lectura de la batería”.

No obstante, se ha llevado un estudio más meticuloso en el código de las tarjetas y se ha conseguido modificarlo de tal forma que pueda leerse el sensor añadido junto con todos los sensores que contienen las tarjetas, sin ningún tipo de conflictos entre librerías. Esto significa que podrán añadirse tantos sensores como se desee. Siempre y cuando se trate de sensores analógicos conectados a los pines de expansión correspondientes a las entradas adc y con la electrónica de acordonamiento adecuada. Existe más formas de añadir otros tipos de sensores con otras interfaces (IRQ, I2C, SPI, UART, 1-wire), pero en este proyecto se ha centrado en un solo tipo.

Continuando con las modificaciones pertinentes, la imagen siguiente muestra la sección del código "WSNDemoApp.h" que se debe modificar.

```
typedef struct
{
  uint8_t     messageType;
  uint8_t     nodeType;
  ExtAddr_t   extAddr;
  ShortAddr_t shortAddr;
  uint32_t    softVersion;
  uint32_t    channelMask;
  PanId_t     panID;
  uint8_t     workingChannel;
  ShortAddr_t parentShortAddr;
  uint8_t     lqi;
  int8_t      rssi;
  //additional field
  uint8_t boardType;//1
  uint8_t sensorsSize;//1
  struct {
    uint32_t battery;
    uint32_t temperature;
    uint32_t light;
    uint32_t sensorAdc3;
  } meshbean;
} PACK AppMessage_t;
```

Fig 4.19 *Parte del código del archivo modificado WSNDemoApp.h*

Como se observa, este trozo de código pertenece a la estructura de los mensajes que se envían. Cada sensor que añada deberá aparecer en esta sección. El código perteneciente al potenciómetro sería "uint32_t sensorAdc3". Si en la misma tarjeta se quiere añadir otro sensor más se añadirá por ejemplo "uint32_t sensorAdc2".

Ahora se debería de modificar el archivo WSNSensorManager.c. A continuación se mostrará el código modificado para la lectura del canal adc, concretamente el tercero.

```
/*****************************************************************
*****
  WSNSensorManager.c
  Copyright (c)Meshnetics.

  History:
    13/06/07 I. Kalganova - Modified
*****************************************************************
******/

#include <types.h>
#include <taskManager.h>
#include <sensors.h>
#include <WSNDemoApp.h>
#include <adc.h>

extern AppMessageRequest_t  appMessage;
// Prototipes
static void temperatureDataReady(bool result, int16_t temperature);
static void lightDataReady(bool result, int16_t light);
static void batteryDataReady(uint16_t battery);
static void sensorLeido(uint16_t sensor);


static void (*callback)(void);
static void HAL_OpenAdcCompleted();
static int16_t resultado;
static bool openADC;
static bool bateriaLeida;
static HAL_AdcParams_t param;

void appStartSensorManager()
{
param.resolution = RESOLUTION_10_BIT;
param.sampleRate = ADC_9600SPS;//al ser un parametro que se puede
modificar, una velocidad de muestro alto daria problemas
param.voltageReference = AREF;
param.bufferPointer = &resultado;
param.selectionsAmount = 1;
param.callback = HAL_OpenAdcCompleted;

#ifdef _TEMPERATURE_SENSOR_
  openADC = HAL_OpenAdc(&param);
 assert(SUCCESS == BSP_OpenTemperatureSensor(), 0xf007);
#endif // _TEMPERATURE_SENSOR_
#ifdef _LIGHT_SENSOR_
  assert(SUCCESS == BSP_OpenLightSensor(), 0xf008);
#endif // _LIGHT_SENSOR_
#ifdef _BATTERY_SENSOR_
 // assert(SUCCESS == BSP_OpenBatterySensor(), 0xf009);
#endif // _BATTERY_SENSOR_
}
```

Fig 4.20 *Código modificado del archivo WSNSensorManager.c*

```
  void appStopSensorManager()
{
  // No checking for return status  yet
#ifdef _TEMPERATURE_SENSOR_
  HAL_CloseAdc();
  assert(SUCCESS == BSP_CloseTemperatureSensor(), 0xf004);
#endif // _TEMPERATURE_SENSOR_
#ifdef _LIGHT_SENSOR_
  assert(SUCCESS == BSP_CloseLightSensor(), 0xf005);
#endif // _LIGHT_SENSOR_
#ifdef _BATTERY_SENSOR_
  //assert(SUCCESS == BSP_CloseBatterySensor(), 0xf006);
#endif // _BATTERY_SENSOR_
  //BSP_CloseLeds();
}
/
void appGetSensorData(void (*sensorsesGot)(void))
{
    callback = sensorsesGot;

#ifdef OMIT_READING_SENSORS
  callback();
#else
  #ifdef _TEMPERATURE_SENSOR_

    assert(SUCCESS == BSP_ReadTemperatureData(temperatureDataReady),
0xf001);
  #else
    temperatureDataReady(1, 10);
  #endif //_TEMPERATURE_SENSOR_
#endif
}

static void temperatureDataReady(bool result, int16_t temperature)
{
  //save sensor data
  if (result)
    appMessage.data.meshbean.temperature = temperature;
  else
    appMessage.data.meshbean.temperature = 0;
  //callback();

#ifdef _LIGHT_SENSOR_
  assert(SUCCESS == BSP_ReadLightData(lightDataReady), 0xf002);
#else
  lightDataReady(0, 0);
#endif //_LIGHT_SENSOR_
}

static void lightDataReady(bool result, int16_t light)
{
  if (result)
    appMessage.data.meshbean.light = light;
  else
    appMessage.data.meshbean.light = 0;
```

Fig 4.20 *Código modificado del archivo WSNSensorManager.c*

```
#ifdef _BATTERY_SENSOR_
//  assert(SUCCESS == BSP_ReadBatteryData(batteryDataReady),
0xf003);
  if (openADC==0) {
    bateriaLeida = false;
  }

#else
 // batteryDataReady(0);
#endif // _BATTERY_SENSOR_
}

static void batteryDataReady(uint16_t battery)
{
  appMessage.data.meshbean.battery = battery;
    if (openADC==0) {
      bateriaLeida = true;
      }
}
static void sensorLeido(uint16_t sensor){
  appMessage.data.meshbean.sensorAdc3 = sensor;
  callback();
}

static void HAL_OpenAdcCompleted(){
      if (bateriaLeida==false) {
            bateriaLeida = true;
            batteryDataReady(resultado);
      } else sensorLeido(resultado);
}
//eof WSNSensorManager.c
```

Fig 4.20 *Código modificado del archivo WSNSensorManager.c*

Cuando se llama a la función "appStartSensorManager()" se inicializa la interfaz adc con aquellos parámetros y una vez abierto se ejecuta el callback "HAL_OpenAdcCompleted()" definido en los parámetros. El dato obtenido del canal tercero del adc se guarda temporalmente en “resultado” que es transmitido al valor de la batería en este callback.

Se ha descartado la manera preestablecida de obtener el valor de la batería por una lectura mediante la apertura del canal adc 0. Puesto que la batería en sí está conectada a este canal, es posible hacerlo. El motivo es porque existía un porblema a la hora de obtener los datos de la batería por medio de la librería BSP junto con la lectura de los canales adc. No se ha descubierto el motivo, pero esta forma de leer los dos canales (batería y sensor) por medio de la librería adc, es efectiva. En primer lugar se hace una lectura de la batería, usando la misma variable "resultado" y a continuación el canal conectado al sensor. De esta manera si se abriese el monitor WSN con esta modificación, se mostraría el dato del sensor conectado por el adc y el de la batería. Si se desease meter un segundo sensor, como ya se ha dicho, se le añade al archivo WSNDemoApp.h una nueva variable y se modifa el archivo WSNSensorManager.c de tal forma que se puedan leer los canales adc.

4.6 Conclusiones

Las conclusiones que se pueden obtener de este capítulo son:

- Se ha visto en profundidad el manejo del kit propuesto en este proyecto.
- El uso del programa AVR Studio es intuitivo y fácil de utilizar, tanto para cargar los códigos creados como hacerlos funcionar en los dispositivos.
- Al hacer un estudio de las librerías esenciales de la librería BitCloud se demuestra con qué facilidad se pueden manejar las tarjetas, además de implementar un pequeño programa de lectura del sensor vía radio.
- El estudio a fondo del código que gobierna a las tarjetas ha brindado la posibilidad de llevar a cabo el objetivo de añadir un sensor aprovechando el código y sin estropear el protocolo ZigBee ni la comunicación entre las tarjetas.
- Por último, la facilidad del uso que ofrece la librería BitCloud ha sido posible gestionar todos los sensores de la tarjeta y el añadido por el puerto de expansión,

CAPÍTULO 5

APLICACION WSN

En este capítulo se presentará la aplicación en java WSN. Se empezará por una pequeña introducción de la funcionalidad del programa. Se continuará por una breve explicación de la instalación de la herramienta de programación Java y por último se expondrá el diseño final de la aplicación y su funcionamiento.

5.1 Introducción a la aplicación WSN.

En capítulos anteriores se había comentado la necesidad de crear una aplicación que leyese directamente los mensajes que se reciben del coordinador, puesto que la aplicación de prueba que viene con el kit no es posible la visualización de un nuevo sensor. También se mostró la forma de añadir nuevos sensores usando el código de prueba WSNDemo.

Pues bien, esta aplicación se encarga básicamente de leer directamente del puerto del que está conectado la tarjeta con funciones de Coordinador, los mensajes que recibe, e incrustarlo en una base de datos MySQL solo aquella información que puede ser de utilidad para que más tarde se haga uso de ella. Esta información se estructura básicamente en el dato obtenido por el sensor, el nombre de éste y la identificación de la tarjeta que lo contiene.

5.1.1 Instalación del WSN.

Esta aplicación no requiere ninguna instalación. Solo es necesario ejecutar el archivo "star.cmd" que se encuentra en la carpeta WSN que se puede localizar dentro del CD ROM de este proyecto. Lo que sí es esencial es tener instalado los paquetes de Java para su funcionamiento y el driver que controlará el puerto.

Java es una herramienta de programación (como C, C++, BASIC, Pascal o Logo) que sirve para crear aplicaciones informáticas. Los programas Java pueden ser aplicaciones independientes (que corren en una ventana propia). Se trata de un lenguaje "orientado a objetos". Esto significa que los programas se construyen a partir de módulos independientes, y que estos módulos se pueden transformar o ampliar fácilmente, procedimiento que se usará para el desarrollo de la aplicación. Los paquetes de java se pueden encontrar en la página Java.Sun.com con el nombre de Jdk6 o en la carpeta de software de este proyecto. El controlador del dispositivo se auto instala con el kit de MeshNetics. Si el ordenador no lo detectase sería forzoso instalarlo manualmente desde el administrador de dispositivos de Windows con la función "Buscar automáticamente el controlador".

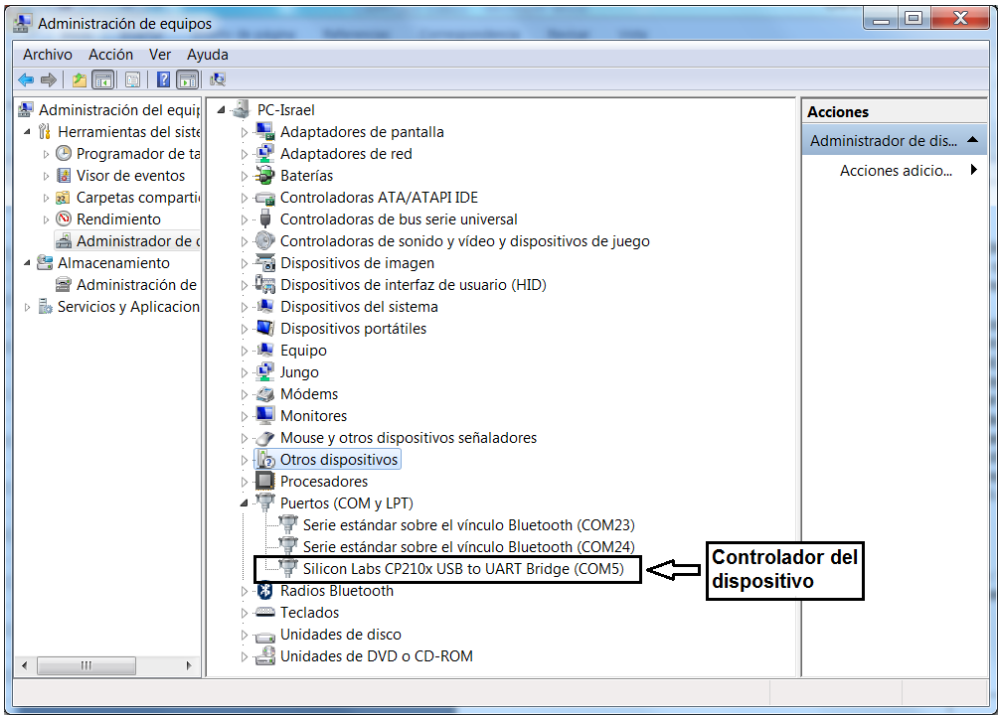


Fig 5.1 *Controlador del dispositivo*

Además, para que exista una conexión directa entre MySQL y la plataforma Java debe de estar instalado el conocido driver "MySQL connector java". Este ejecutable se puede encontrar también en la carpeta de programas del CD ROM del proyecto. Su instalación no requiere ninguna atención especial.

5.1.2 Funcionamiento del programa.

Primeramente tiene que estar conectado la tarjeta Coordinador al puerto serie del ordenador y encendido. Se recuerda que la tarjeta en esta modalidad debe estar bien situado los conmutadores del switch en la posición siguiente:

1	2	3	Descripción
ON	ON	X	La tarjeta se configura como Coordinador
ON	OFF	X	La tarjeta se configura como Router
OFF	OFF	X	La tarjeta se configura como Dispositivo final

Fig 5.2 *Configuración de los switch*

Una vez hecho esto se ejecuta el "star.cmd" de la carpeta WSN. Este archivo es meramente una instrucción ".jar" de la ventana de comandos que inicializa la aplicación java. También podría activarse introduciéndolo manualmente desde el CMD, pero por comodidad es recomendable usar este archivo.

Pues bien, al ejecutarse aparece la ventana de comandos de Windows con la instrucción para abrir la aplicación como se muestra a continuación:

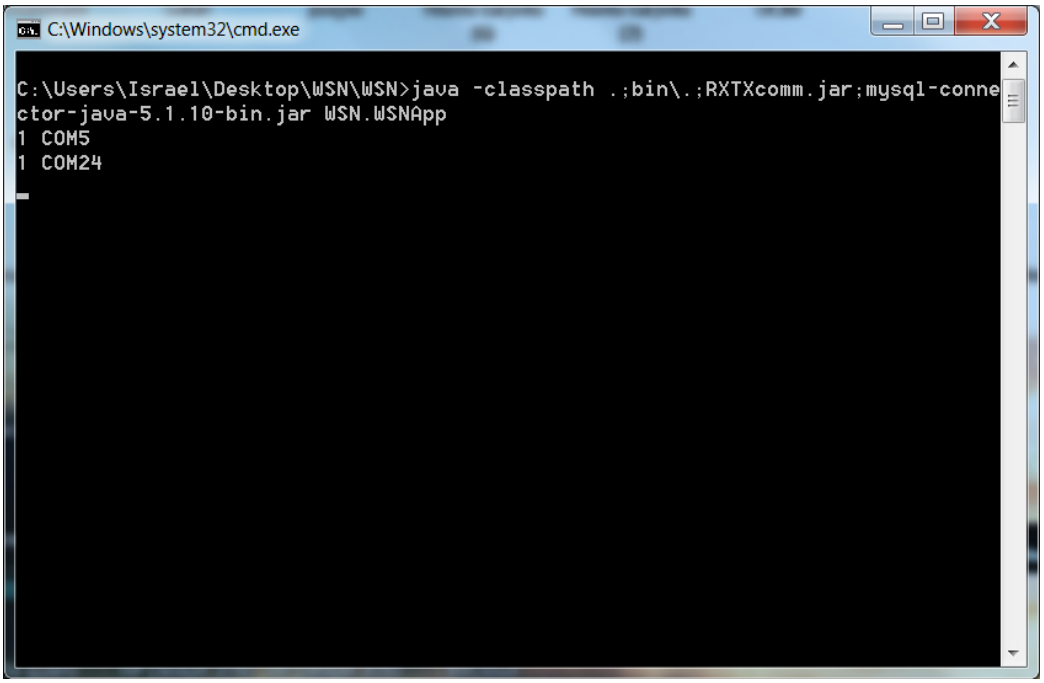


Fig 5.3 *Ventana de comandos*

Como se puede observar en la imagen, aparece la instrucción Java junto con los puertos que ha detectado el programa. Un tiempo considerable después sale la ventana principal de la aplicación:

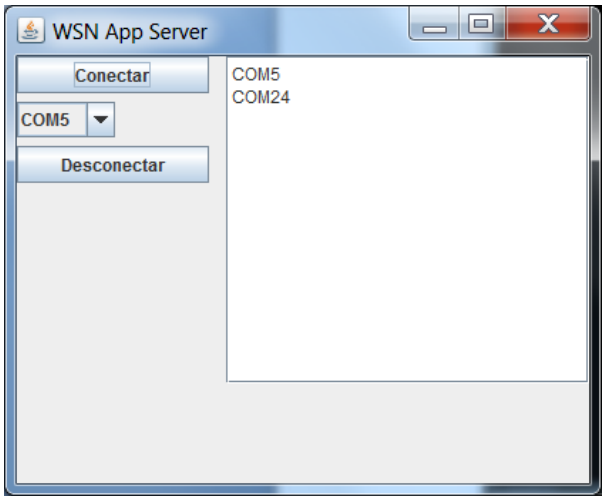


Fig 5.4 *Ventana principal de la aplicación*

En la aplicación pueden encontrarse los botones de conexión y desconexión a la base de datos junto con un desplegable para la selección del puerto correspondiente a la tarjeta. También incluye una ventana de texto en la que se exponen los procesos de funcionamiento del programa. Estos pueden ser:

- Reconocimiento de los puertos abiertos del ordenador.
- Conexión y desconexión del puerto
- Los mensajes recibidos desde el Coordinador.

Para comenzar a recibir mensajes por el puerto, se selecciona el puerto correspondiente y se cliquea en conectar. Si aparece un "Ok" en la ventana de texto significa que la conexión ha sido satisfactoria. En cuestión de segundos comenzarán a emerger en la ventana de texto los mensajes que se reciben por el puerto de conexión.

Fig 5.5 *Mensaje recibido por el puerto*

En el cuadro texto se exponen los valores obtenidos de los sensores y la prueba de que ha sido introducido en la base de datos. Previamente la base de datos debe de estar encendida. Todo lo referente a la instalación y activación de la base de datos se explicará en el capítulo siguiente.

5.2 Desarrollo de la aplicación.

Para programar esta aplicación se decidió usar el entorno del Eclipse. Eclipse es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java. Se trata de un entorno de desarrollo integrado (IDE) en el que se puede encontrar todas las herramientas y funciones necesarias para el trabajo que se realizar. Además posee una atractiva interfaz que lo hace fácil y agradable de usar. Cuenta con un editor de texto donde puedes ver el contenido del fichero en el que se está trabajando, una lista de tareas, y otros módulos similares. El programa puede descargarse desde su página web. Al instalarlo preguntará donde se ubicará la carpeta de trabajos WorkSpace.

5.2.1 Pruebas iniciales.

El siguiente paso será leer los datos que provienen del coordinador a través del puerto serie. En internet se puede encontrar mil maneras de cómo gestionar un puerto. Por su simplicidad y por los ejemplos que vienen se utilizará la implementación RXTXcomm (www. RXTX.org). Al descargarse el fichero viene con tres archivos que solo será necesario copiarlos en la carpeta de ejecución de nuestra próxima aplicación java. Esta implementación permite ver los puertos que se encuentran actualmente abiertos en el PC y leer lo que se transmite por ellos.

A continuación se abre el Eclipse y se crea un nuevo proyecto pinchando en File/New/Java Proyect.

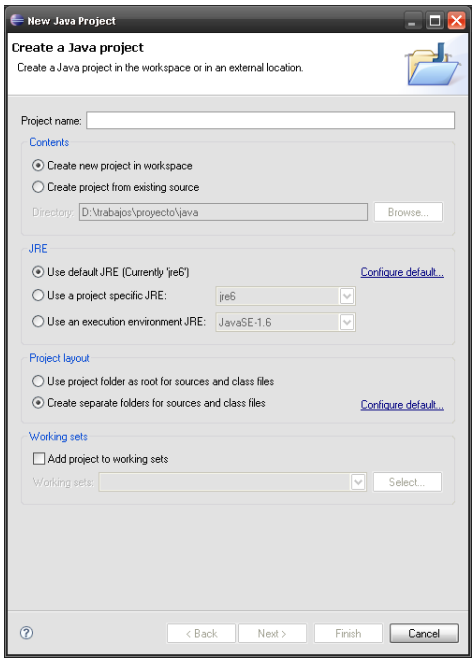


Fig 5.6 *Crear un nuevo proyecto*

En la casilla Proyect name se escribe el nombre del proyecto. En este caso se llamará "Prueba", pues no será el desarrollo final de la aplicación. Mas bien, ha sido creada con la finalidad de transmitir lo que entra por el puerto serie, que luego será usada en el desarrollo final de la aplicación. Al pulsar "Finish" se habrán creado una serie de carpetas y subcarpetas dentro de la carpeta de trabajos del Eclipse (WorkSpace). Ahora se copian los archivos descritos anteriormente en la nueva carpeta de la aplicación. Se actualiza el Eclipse para que aparezcan estos archivos pulsando F5. Solo faltaría por incluir estas librerías en la nueva aplicación. Para ello se pincha botón derecho en el icono “Prueba” que se encuentra en la ventana izquierda “Package Explorer” y se selecciona Build Path/Configure Build Path. Aparecerá una ventana como la que se muestra a continuación:

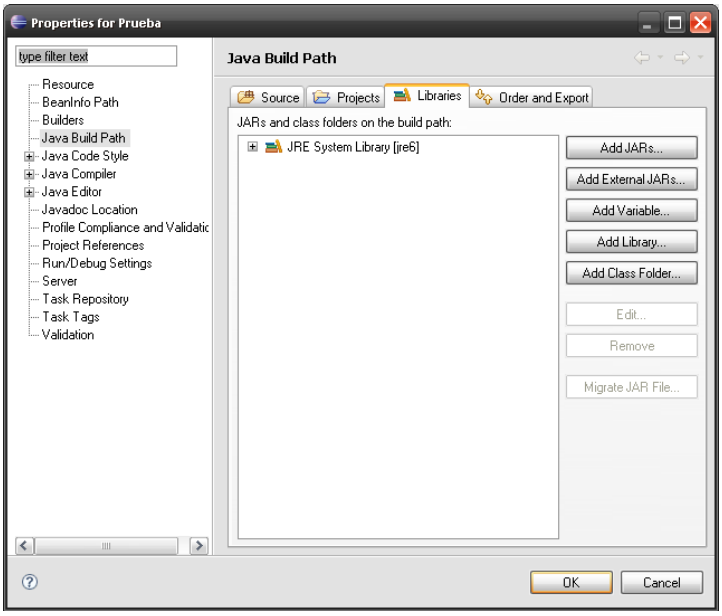


Fig 5.7 *Inclusión de las librerías necesarias para compilar*

Dentro de la pestaña “Libraries” se pincha “Add JARs” y se selecciona la librería “RXTXcomm.jar”. En internet también se pueden encontrar líneas de código que gestionan el puerto COMM. Es la manera más fácil de encontrar solución a la comunicación serie. La imagen que aparece a continuación es la del código final obtenido a partir de varias pruebas realizadas.

```
import java.util.*;
import java.io.*;
import gnu.io.*; // for rxtxSerial library
class PruebaCOMM {
      public PruebaCOMM(){
            Enumeration portList =
CommPortIdentifier.getPortIdentifiers();
            CommPortIdentifier portId;
            InputStream inputStream;
            SerialPort serialPort;
            OutputStream outputStream;
            try { while (portList.hasMoreElements()){
                   portId = (CommPortIdentifier)
            portList.nextElement();
System.out.println(portId.getPortType()+"
      "+portId.getName());
            }
      portId = CommPortIdentifier.getPortIdentifier("COM4");
      serialPort    =    (SerialPort)    portId.open("SimpleReadApp",
      2000);
      inputStream = serialPort.getInputStream();
      serialPort.setSerialPortParams(
                        38400,
                        SerialPort.DATABITS_8,
                     SerialPort.STOPBITS_1,
                     SerialPort.PARITY_NONE);
   serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
   int contadormio =0;
      do{   int dato = inputStream.read();
            if (dato!=-1){
                  contadormio++;
                  System.out.print(dato+" ");
                  if ( (contadormio%47)==0)
                              System.out.println("");
                        }
      } while (true);
      //serialPort.close();

      } catch (Exception e) {
            e.printStackTrace();
      }
            System.exit(-1);
   }
   public static void main(String[] args) {
            System.out.println("Hello World!");
            PruebaCOMM p = new PruebaCOMM();
      }}
```

Fig 5.8 *Código resultante para la conexión serie*

Este código lo que hace es abrir el puerto junto con los parámetros descritos y pintar por pantalla lo que recibe. Lo siguiente sería copiar este código en la nueva clase creada y guardarlo. Se conecta la tarjeta MeshNetics y se comprueba qué puerto se ha abierto con el Administrador de Windows. Se comprueba que es el mismo puerto que está escrito en el código y por último se ejecuta la aplicación. Estos son los datos que se reciben por el puerto:

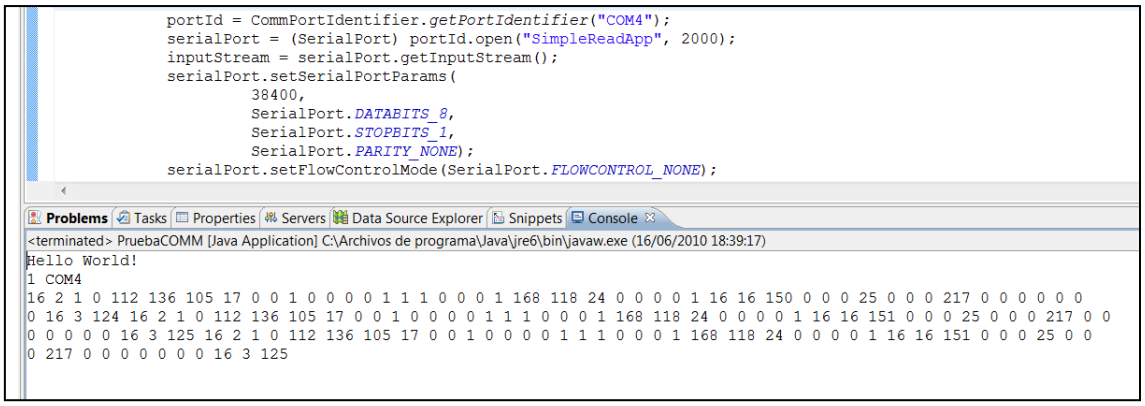


Fig 5.9 *Datos obtenidos por el puerto*

Los datos recibidos por el puerto serie provienen de los mensajes que nos envía el coordinador. Cada cierto tiempo se recibe un nuevo mensaje. Estos mensajes contienen toda la información de la lectura de los sensores, así como todo el conjunto de datos que son necesarios para la creación de una red de comunicación, como la máscara de red, el PANID, dirección, etc. Solo con ver el código fuente se deduce que es cada dato. La composición del mensaje se puede obtener del fichero WSNDemoApp.h. A continuación se puede observar una parte del código de este archivo:

```
BEGIN_PACK
typedef struct
{
  uint8_t     messageType;
  uint8_t     nodeType;
  ExtAddr_t   extAddr;
  ShortAddr_t shortAddr;
  uint32_t    softVersion;
  uint32_t    channelMask;
  PanId_t     panID;
  uint8_t     workingChannel;
  ShortAddr_t parentShortAddr;
  uint8_t     lqi;
  int8_t      rssi;
  //additional field
  uint8_t boardType;//1
  uint8_t sensorsSize;//1
  struct {
    uint32_t battery;
    uint32_t temperature;
    uint32_t light;
      uint32_t sensorAdc3;
  } meshbean;
} PACK AppMessage_t;
```

Fig 5.10 *Parte del código de WSNDemoApp.h*

5.2.2 Estudio de los mensajes recibidos.

A continuación se muestra un mensaje descompuesto:

- 16 2: Bytes de inicio de trama. Cada mensaje que se recibe comenzará con esta numeración. Esta información se puede extraer del código fuente de WSNUARTManager.c Al final del código aparece un algoritmo que añade un inicio de trama, *p++ = 0x10 y *p++ = 0x03 por medio de un puntero *p.
- 1: Tipo de mensaje. Indica si es un mensaje de datos o de error. Normalmente tendrá como valor un 1 indicando que es un mensaje de datos.
- 0: Tipo de nodo: tendrá un valor 0 indicando que el mensaje recibido es del coordinador, 1 si es del router o un 2 si es del dispositivo final.

Los siguientes datos son los que corresponden para establecer una comunicación entre los dispositivos:

- 112 136 105 17 0 0 1 0: Dirección mac.
- 0 0: ShortAddr.
- 0 1 1 1: Versión del sofware.
- 0 0 0 1: Mascara del canal.
- 168 118: panID.
- 24: Canal de trabajo.
- 0 0: parentShortAddr.

Los sucesivos bytes indican la calidad de la señal de transmisión:

- 0: lqi.
- 0: rssi. Son de valor 0 lógicamente porque se trata de un mensaje del coordinador que está conectado al PC.
- 1: tipo de tarjeta
- 16: SensorSize: Este dato corresponde a la extensión de memoria que se reserva dentro de la trama para los datos que se reciben de los sensores. Este parámetro está ya comentado anteriormente en la explicación del fichero WSNDemoApp.h. Se reservan 4 bytes por sensor y como están especificados 4 sensores en el fichero, le corresponde 16 bytes.
- 16: Duplicación. Aparece en el momento que existe algún dato con el valor 16 dentro de la trama. Su finalidad consiste en no confundir con el inicio o final de trama. Si el siguiente dato fuese un 12, habría conflictos en el mensaje. El algoritmo que realiza esta funcíon se puede encontrar en el fichero WSNUARTManager.c
- 150 0 0 0: Valor del sensor de batería
- 25 0 0 0: Valor del sensor de temperatura
- 217 0 0 0: Valor del sensor de luz
- 0 0 0 0: Valor del sensor conectado a la tarjeta
- 16 3: Bytes de final de trama. Tiene la misma finalidad que los bytes de inicio de trama. También se puede ver el algoritmo en WSNUARTManager.c.
- 125: Control de redundancia cíclica (CRC): Sistema de detección de errores en el mensaje.

5.2.3 Estudio del diseño final.

A continuación se mostrará una explicación del desarrollo de la aplicación final, que, como se ha comentado antes, está basada en la descomposición del mensaje descrito.
Estas explicaciones están fundamentadas en los archivos ".java" que pueden encontrarse dentro de la carpeta del programa WSN. Solo se hará hincapié en aquellas partes de código que se han considerado importantes.

5.2.3.1 Fichero de configuración.

La finalidad de este fichero es poder pasar parámetros al programa a través de un fichero de texto llamado "configuracion.properties" (ver figura). Este fichero tiene como estructura un nombre, un "=" y un valor. De esta manera se adjudica un valor a los parámetros configurables del programa. Por ejemplo se tiene como parámetro "mysqlURI", cuyo valor es "jdbc:mysql://localhost:3306/wsn". En este caso el parámetro especifica la cadena de conexión a la base de datos MySql.

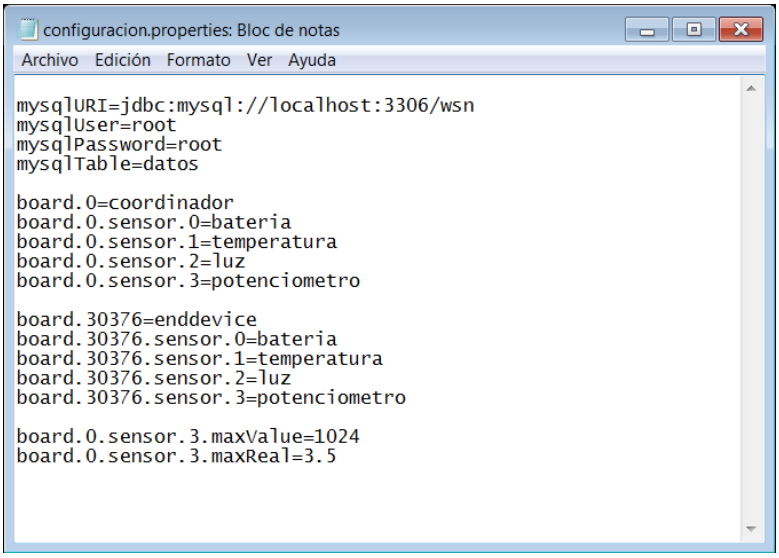


Fig 5.11 *Fichero de configuración*

Los parámetros configurables que se encuentran en este fichero son los parámetros de conexión a la base de datos MySql. Los siguientes parámetros sirven para configurar el nombre de las placas y el nombre de los sensores. Este será el apartado donde se especificarán los nombres correspondientes a cada dispositivo y a cada sensor. La numeración de cada sensor corresponde a la posición de la trama del mensaje.

Por último, los parámetros que aparecen como "maxValue" y "maxReal" corresponden al máximo valor digital y al máximo valor real analógico que reciben los parámetros del sensor conectado al periférico. Si se observa en el código principal de los dispositivos el formato de la trama de los sensores vienen en enteros, entre 0 y un valor arbitrario, puede ser un byte, dos bytes ... El valor máximo no está definido, varía entre 0 y un valor que habrá que especificar. Por otra parte, maxReal tiene como valor máximo 3,5 al ser el sensor un potenciómetro alimentado con la batería de la placa. La finalidad de este apartado es convertir un voltaje que varía de 0 a 3,5 a un valor digital que varía entre 0 a 1024.
Teniendo esta lista de valores configurables se evita estar recompilando el programa cada vez que se modifica algún parámetro.

5.2.3.2 ConexionSerie.java.

Es una clase que funcionará paralelamente al programa principal. Básicamente esta clase se encargará de abrir puerto serie y escuchar al coordinador que está conectado. Es decir, su función será recoger la información que llegará desde el puerto serie en un proceso aparte del programa principal. Cuando recibe un paquete de datos del puerto serie, lo analiza y busca el comienzo y el final de la trama para decodificarla. comprueba que es correcta la trama y si es así la envía a la función principal.

Para detectar y analizar la trama se ha diseñado un "mini" autómata de tres estados. El nombre del primer estado es "buscando trama", significa que está a la espera de recibir algún valor del puerto serie. Cuando empiezan a llegar datos tiene que encontrar un comienzo de trama que correspondería a los valores "16 2" del mensaje. Una vez encontrado el comienzo, se pasa al siguiente estado "dentro de trama". En este estado analizaría el mensaje en busca de los dobles "16" para eliminarlos y el final de trama, que correspondería al "16 3". Estos dobles 16 son insertados en la trama por la aplicación que corre en los dispositivos para evitar confusiones con el inicio y final de trama. De esta manera, si apareciese un 16 en medio de la trama seguido de algún dato con valor "3", se evitaría confundirlo con el final de la trama. Pues bien, en este estado los detectará y los quitará para simplificar el mensaje. Una vez hecho esto, comprueba el CRC. Si todo va bien, pasa al siguiente estado "enviando trama". En este estado construye el mensaje filtrado sin los inicios ni finales de trama, ni los 16 dobles y sin el CRC y lo envía a la aplicación principal.

Además de este mini autómata, la clase "Conexiónserie" está dividido en varios métodos, entre los que se encuentra "obtenPuertos". Este método tiene como finalidad averiguar los puertos que están en uso y devolver un string con todos los puertos disponibles e imprimirlos por pantalla. El siguiente método es " ConexionSerie", que tiene como parámetro el "ReceptorMensajes", es decir, a quien se tiene que enviar el mensaje cuando se encuentra una trama completa y por último "el puerto". Este último indica cual es el puerto que se abrirá.
Además, esta clase tiene como finalidad construir un mensaje utilizando un buffer de datos, que al llegar al estado "enviando trama", invoca al receptor de mensaje y lo envía. "Mensaje" sería en este caso el buffer que se ha creado con este fin.

5.2.3.3 ReceptorMensajes.java.

"ReceptorMensajes" es una clase que actúa como una interfaz cuyo código está formado por dos líneas. Su función principal es crear una especie de contrato que debe ser cumplido por aquella clase que implemente esta interfaz realizando una llamada a la función "nuevoMensaje". Es decir, es una especie de compromiso en la que una clase que implemente "ReceptorMensajes" tiene que tener dentro de sus métodos uno que se llame "nuevoMensaje".
Se puede ver también como la manera de conectar clases entre sí. Como es el caso de las clases "ConexionSerie" y "WSNApp". Cuando se termine de recibir una trama y construir el mensaje, este es enviado a "WSNApp" por medio de esta interfaz, la cual indica cómo debe de ser el mensaje, es decir, como es la llamada.

En la siguiente línea de código se puede observar como el receptor si es diferente a null invoca el método "receptor.nuevoMensaje". Es ahí cuando "ConexionSerie" está enviando el mensaje a "WSNApp" y vuelve al estado inicial de "buscando trama".

```
if (receptor!=null) receptor.nuevoMensaje(mens);
```

Fig 5.12 *Parte del código ReceptorMensajes.java*

5.2.3.4 Mensaje.java.

"Mensaje.java" contiene la estructura que ha sido recibida a través del puerto serie desde la aplicación .c proveniente del dispositivo. Es decir, el código está formado por todas las propiedades privadas como "mensaje tipe", "node tipe", etc. con los mismos tamaños de bytes que proviene de los dispositivos. Además contiene ciertos métodos get y set para cada propiedad privada. De esta manera se pueden establecer y leer todos los valores de cada propiedad con facilidad. Por ejemplo, si se tiene una propiedad privada llamada MessageType, para poder leer su valor hay un "getMessageType" y en la misma propiedad hay un mensaje para establecer el valor, "setMessageType" (ver figura). Así ocurre con todas las partes que forman la estructura de cualquier mensaje que proviene del puerto serie.

```
public int getMessageType() {
            return messageType;
      }
      public void setMessageType(int messageType) {
            this.messageType = messageType;
      }
```

Fig 5.13 *Parte del código Mensaje.java*

Sin embargo hay una propiedad que tiene una función diferente, que es "setBuffer". La trama que llega por el puerto serie está sin estructura. Se trata de una cantidad de bytes sin estructura alguna. La función de "setBuffer" consistiría en trocear esta estructura ya filtrada, es decir, previamente eliminado los dobles 16, comprobado el crc e indicado el inicio y final de la trama, y meter todos aquellos datos en sus correspondientes variables como, mensaje tipe, node tipe... y las que interesantes, que son sensor size y sensor data. Sencillamente lo que hace es coger el buffer, la trama que ya ha llegado de conexsionserie y transformarla en un mensaje con todos sus correspondientes métodos "set and get" que son mas fácilmente programables.

Después de esto, lo único que habría que hacer es construir el mensaje final y enviarlo a la aplicación principal WSNApp con la información preparada, para que ya solo se tenga que apuntar "mensaje.getsensordata" y recibir el dato correspondiente a esa variable.
Una vez dicho esto, se pasará a la explicación de la aplicación principal.

5.2.3.5 WSNApp.java.

La función principal de esta aplicación no es más que recibir un mensaje e insertarlo en la base de datos. Se comenzará por ver la función "inserta mensaje" puesto que es la encargada de obtener las propiedades de la placa como el nombre, su dirección IP, los datos del sensor, etc. de los cuales, algunos se obtendrán desde el fichero de configuración visto anteriormente.

```
String nombrePlaca = msg.getShortAddr()+"";
String param = "board."+msg.getShortAddr();
if (prp.containsKey(param)) nombrePlaca = prp.getProperty(param);
```

Fig 5.14 *Obtención de los valores del fichero de configuración*

Observando esta sección de código se puede apreciar cómo se obtiene el nombre, la dirección IP y la clave del fichero de configuración si existiese. "msg" sería el nombre del mensaje recibido. Si ese fichero de configuración contuviese un parámetro llamado board.0 entonces el nombre de la placa sería el valor de esa propiedad. En las siguientes líneas hay un

bucle que se encarga de obtener los nombres de los sensores a partir de la función getsesorsize.

```
for(int i=0;i<msg.getSensorsSize();i++){
      String nombreSensor = i+"";
      String param2 = param+".sensor."+i;
      if (prp.containsKey(param2))
      nombreSensor = prp.getProperty(param2);
```

Fig 5.15 *Bucle para la obtención de los nombres de las placas*

Sabiendo cómo es la estructura del fichero .c del coordinador, existe un parámetro llamado sensorsize que te dice cuantos sensores tiene esa placa. Entonces haciendo un bucle que recorra el mensaje obtengo el nombre del sensor. Si en el fichero de configuración no estuviera especificado el nombre del sensor se pondría por defecto un número comenzando desde 0. Si el fichero de propiedades no estuviera bien configurado, se insertaría en la base de datos unos números sin sentido ya preestablecidos.

A continuación se procedería a la normalización del valor del sensor con el "maxvalue" y "maxreal" explicado anteriormente solamente si existe los dos valores, es decir, si el fichero de propiedades contiene los dos valores definidos entonces se calcularía una proporción sencillamente.

```
double datoSensor = msg.getSensorsData(i);
String param3 = param2+".maxValue";
String param4 = param2+".maxReal";
if (prp.containsKey(param3)&&(prp.containsKey(param4))){
     double maxValue= Double.parseDouble(prp.getProperty(param3));
     double maxReal = Double.parseDouble(prp.getProperty(param4));
     datoSensor = datoSensor*maxReal/maxValue;
}
resultado = resultado & insertaEnTabla(nombrePlaca, nombreSensor,
datoSensor);
```

Fig 5.16 *Normalización de maxValue y maxReal*

Recapitulando, "insertamensaje" obtiene el nombre de la placa. Para cada sensor obtiene el nombre del sensor, se normaliza el valor del sensor siempre que este definido maxvalue y maxreal para ese sensor y luego se calcula la proporción. A continuación se insertaría todo lo descrito en la base de datos de la siguiente forma. Se define para cada placa y sus sensores un "resultado" que luego insertará en la base de datos.

```
resultado = resultado & insertaEnTabla(nombrePlaca, nombreSensor,
datoSensor);
```

Fig 5.17 *Definición de resultado*

Luego hará una consulta apropiada para comprobar la conexión:

```
private boolean insertaEnTabla(String placa,String sensor,double
dato){
if (this.conn==null) ObtenConexion();
if (this.conn==null) return false;
```

Fig 5.18 *Inserción a BBDD*

"insertaMensaje" llama a "insertaEnTabla", cuyos parámetros son el nombre de la placa, el nombre del sensor y el dato. Se hace una conexión a la base de datos. Si el objeto "conn" está "null" entonces llama a "obten conexión". Si "ObtenConexion" continua estando "null" se ejecuta un "return false", es decir, devolverá un error y no seguirá ejecutando nada hasta que se obtenga una conexión.

Después en las siguientes líneas y suponiendo que se obtiene una conexión se ejecuta unas funciones sql con un lenguaje propio de base de datos. Realiza un "String sql" que contenga los parámetros que van a ser insertados y se ejecuta. Hay que recordar que las propiedades como mysql table, mysql uri, my sql user y my sql password son configurables desde el fichero de configuración. Por lo tanto con la función "prp.getProperty" se obtienen los datos configurables.

Se verá a continuación como funciona "ObtenConexion". Solo hace falta entender que "DriverManager" es el que da el acceso a la base de datos y devuelve una conexión, siendo los parámetros que se pasan, el parámetro my sql uri, my sql user y my sql password, que se encuentran en el fichero de propiedades. Es decir, que para obtener una conexión con la base de datos tenemos que pasarle esos tres parámetros y en vez de estar escrito fijamente en el código dentro del programa se hace una consulta al fichero de configuración.

```
public void ObtenConexion(){
  if (this.conn==null)
  try {
      this.conn=DriverManager.getConnection(prp.getProperty("mysql
      URI"), prp.getProperty("mysqlUser"),
      prp.getProperty("mysqlPassword"));
  }
```

Fig 5.19 *Consulta al fichero de configuración*

Lo único que faltaría por ver es el código introducido por el entorno de desarrollo NetBeans, un programa que está hecho principalmente para el lenguaje de programación Java. Ya se ha explicado la función principal de la aplicación. Ahora se profundizará en el main.

Al final del todo se pueden ver unas "variables declaration" en donde NetBeans pone que objetos tiene el formulario, como un botón, un editor panel, un textfeild... En este apartado no hay que entender todo el código introducido por el entorno. Lo único importante es que al final se hace un main y más adelante se ejecuta "WSNApp().setVisible". Este sería el primer objeto que se crea. Es ahí donde comenzaría el main, y lo primero que hace es crear el entorno del programa WSNApp para luego visualizarlo.

```
public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new WSNApp().setVisible(true);
```

Fig 5.20 *Función para visualizar*

Un poco más arriba hay un apartado que pone "create new form". Ahí se llama al método "initComponents". No es importante como funciona, solo hay que saber que es la función encargada de crear las ventanas con el tamaño diseñado y sus botones de control como el botón para conectar, el de desconectar, un cuadro de texto y el combo box.

Luego se ejecuta "inicializaConfiguracion". Le pasa la variable "prpConfig", con el nombre del fichero "configuracion.properties" y se carga el fichero de propiedades para hacer uso de él.

```
public void inicializaConfiguracion(String fileName){
   this.prp = new Properties();
   try {
      this.prp.load(new FileInputStream(fileName));
       }
   catch (Exception e) {
      System.out.println("No    se    ha    podido    cargar    el
      fichero:"+fileName);
      e.printStackTrace();
   }
}
```

Fig 5.21 *Inicialización de la configuración*

A continuación unas líneas más al principio dentro de la clase "conexiónSerie" encargadas de la gestión del puerto en la función "ConexionSerie.*obtenPuertos*()". Se ejecuta dos veces esta función, una coge los puertos que devuelve "obtenPuertos" y lo pinta en el editor de texto y la otra se encarga de rellenar un combo box del formulario principal, para permitir si hay varios conn, seleccionar que puerto se quiere conectar. En definitiva, la clase "conexiónSerie" indica cuales son los puertos que hay disponibles y con ellos se rellena las opciones del combo box, es decir, del selector.

Lo siguiente que viene después es el código asociado al botón de conectar y el de desconectar. Su funcionamiento es básicamente el control del puerto. Si no se pulsa ninguno, el programa no realizará ninguna función, solo esperará a que se ejecute algún botón. Entonces cuando se ejecuta el botón de conectar se asocia en la clase "jButton1ActionPerformed" un objeto llamado "conexión" que si es igual a null crea una "new ConexionSerie" y le pasa como parámetro "this", que representa a la clase principal e implementa el contrato o la interfaz de receptor de mensaje, y además le pasa otro parámetro, que es el valor seleccionado dentro del combo box, es decir, el puerto que se quiere abrir para empezar a recibir datos desde el coordinador. Al abrirse el proceso la conexión ya no será "null" y podrá proseguir con la ejecución del programa. Luego por muchas veces que se le dé al botón de conectar, simplemente aparecerá que el puerto ya está abierto.

```
if (conexion==null){
    jEditorPane1.setText(jComboBox1.getSelectedItem().toString());
    conexion = newConexionSerie
    (this,jComboBox1.getSelectedItem().toString());
    thread = new Thread(conexion);
    thread.start();
    jEditorPane1.setText("OK");
    } else {
      jEditorPane1.setText("El puerto ya está abierto");
    }
```

Fig 5.22 *Conexión con el puerto*

Al crear la conexión se abre el puerto y ya está preparado para la invocación del método "run", el cual abre un hilo aparte y ejecuta una tarea constantemente. Esta tarea que está siempre en funcionamiento se pone a escuchar el puerto serie y construye mensajes cada cierto tiempo con su apropiada estructura, conforme vayan llegando de los sensores y lo inserta en la base de datos por medio de la clase vista anteriormente "ConexionSerie". Y justo después es pintado por pantalla en el cuadro de texto el resultado de" inserta.mensaje", es decir, el mensaje que ha sido enviado a MySQL.

Finalmente hay un botón de desconectar, que lo único que hace es decirle a "ConexionSerie" que finalice el método ese "run", cerrando el puerto y el proceso que se había creado, de tal forma que cuando se vuelva a dar al botón de conectar, pueda conectarse.

5.3 Conclusiones.

El diseño de una aplicación que gestionase los mensajes enviados por el puerto serie ha dado la posibilidad de hacer un estudio de la estructura de los mensajes que se crean en las tarjetas. Por lo que ha sido posible con diseño final filtrar los mensajes y quedarse con lo más esencial, que son los datos de los sensores. Además, con la implementación de una conexión a la base de datos garantiza el auto guardado de toda la información de interés recibida.

CAPÍTULO 6

BASE DE DATOS

Este capítulo trata todo lo relacionado con la base de datos. Su instalación, la forma de gestionar el programa a usar y un ejemplo para sacarle mayor partido al programa.
Se procederá en primer lugar a una breve introducción al sistema de gestión de base de datos. Más adelante se mostrará un mini tutorial sobre una correcta instalación y se enseñará la forma de reutilizar una tabla ya diseñada para un sencillo uso.

Puesto que la base de datos solo se utilizará para añadir los mensajes de datos recibidos por el coordinador y luego leerlos por la aplicación WSNGui4, únicamente será necesario saber nada más que como encenderlo. No obstante se mostrará un ejemplo para darle uso a la tabla de datos.

6.1 Introducción a Base de datos.

Una base de datos o banco de datos (en ocasiones abreviada con la sigla *BD* o con la abreviatura *b. d.*) es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este proyecto se usará el conocido MySQL.

MySQL es un sistema gestor de bases de datos (SGBD, DBMS por sus siglas en inglés) muy conocido y ampliamente usado por su simplicidad y notable rendimiento. Es una opción atractiva tanto para aplicaciones comerciales, como de entretenimiento precisamente por su facilidad de uso y tiempo reducido de puesta en marcha. Esto y su libre distribución en Internet bajo licencia GPL le otorgan como beneficios adicionales (no menos importantes) contar con un alto grado de estabilidad y un rápido desarrollo.

Se ha optado la utilización de este sistema de base de datos en este proyecto por su simplicidad y fiabilidad, así como las múltiples opciones que ofrece este programa.
Al ser un programa gratuito es fácil encontrarlo por la red. La versión que se utilizará en este proyecto es la 5.1.

Existen múltiples clientes de entorno gráfico que permiten la interacción con un servidor MySQL. Se analizará brevemente los que distribuye la empresa MySQL AB específicamente el paquete de programas relacionado con MySQL Tools 5.0 (MySQL Administrator, MySQL Query Browser, MySQL Migration Toolkit y MySQL System Tray Monitor). Todas estas aplicaciones se pueden descargar del sitio oficial www.mysql.com.
A continuación se describe brevemente la utilidad de cada programa:

- Administrador MySQL (MySQL Administrator) : Consola de administración que permite que un servidor MySQL sea gestionado y lleve a cabo tareas para su mantenimiento.
- Analizador de Consultas (MySQL Query Browser): crea, ejecuta y optimiza consultas SQL de manera visual.
- Migrador de Base de Datos (MySQL Migration Toolkit): Permite que sistemas gestores bases de datos, como Oracle, Microsoft SQL Server y Microsoft Access, puedan migrar sus datos a MySQL.
- Sistema de monitorización (MySQL System Tray Monitor): Es una herramienta que le permite monitorear el estado del servidor local MySQL a través de un indicador que se

encuentra en la bandeja del sistema. También proporciona acceso rápido a las distintas herramientas gráficas de MySQL a través de su menú contextual.

6.1.1 Instalación de MySQL.

En primer lugar se necesitará disponer del programa de instalación MySQL Server 5.1 (a partir de la versión 4.1 valdría). Igualmente se puede descargar desde la misma página oficial. Una vez descargado se ejecuta y se siguen las instrucciones para su instalación:

Fig 6.1 *Programa de instalación MySQL*

Su instalación es muy sencilla. Solo hay que destacar la parte en donde te pide si se quiere hacer una configuración inicial y si se quiere registrar o no. En un principio no es necesario registrarlo, pero hacer una configuración inicial es recomendable hacerla:

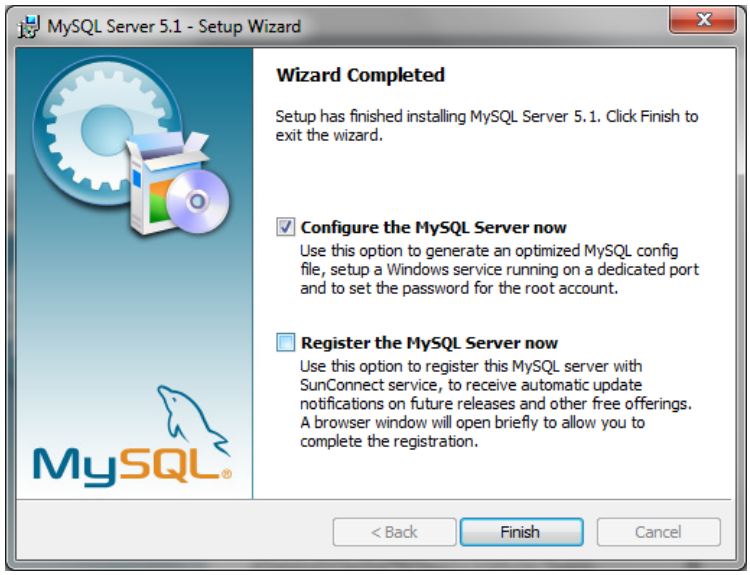


Fig 6.2 *Configuración de MySQL*

Si se realiza la configuración te pide si se quiere hacer una configuración detallada o estándar. Dependiendo del uso que se quiere dar al equipo en el que se instala se marcará una de las tres opciones:

- Developer Machine: Se marcará esta opción si en el equipo donde se ha instalado MySQL Server se utiliza también para otras aplicaciones. MySQL Server utilizará la memoria mínima necesaria.
- Server Machine: Se marcará esta opción si se va a utilizar el equipo para algunas aplicaciones (no demasiadas). Con esta opción MySQL Server utilizará un nivel medio de memoria.
- Dedicated MySQL Server Machine: Se marcará esta opción sólo si se quiere utilizar el equipo como un servidor dedicado exclusivamente a MySQL. Con esta opción MySQL Server utilizará el máximo de memoria disponible. Se obtendrá un rendimiento elevado pero el equipo sólo servirá para MySQL.

Esta opción sería la adecuada si se va recibir un elevado número de mensajes de los sensores. Aún así se escogerá "Developer Machine" (consume el mínimo de memoria necesaria para su funcionamiento) puesto que el volumen de mensajes que se recibirán es pequeña:

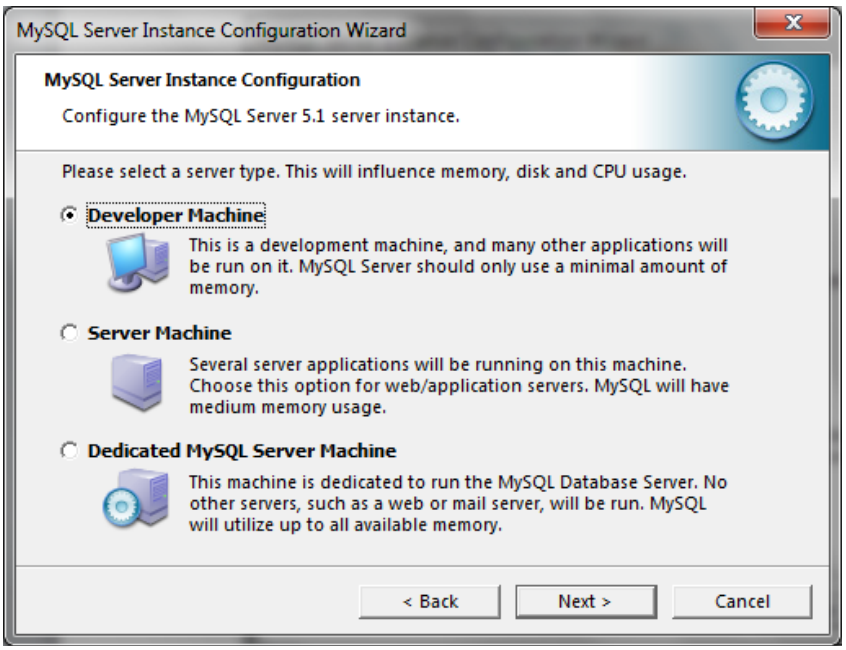


Fig 6.3 *Tipo de servidor*

La siguientes opciones te permite utilizar MySQL como base de datos para transacciones de otra Base de Datos MySQL. Normalmente se marcará "Multifunctional Database" puesto que no será necesario esta opción. La siguiente pestaña permite seleccionar la unidad y la carpeta donde se quiere guardar los ficheros de datos (Tablespace) de la Base de Datos.

A continuación se seleccionará el número aproximado de conexiones concurrentes (varios clientes conectados a la vez) que tendrá el servidor de MySQL. Se seleccionará la primera opción, pues solo tendrá una conexión.

Se dejará marcada la opción "Enable TCP/IP Networking" puesto que se quiere conectar mediante TCP/IP al equipo servidor de MySQL. Por defecto se suele dejar el puerto 3306 (si se tiene instalado algún cortafuegos se deberá abrir dicho puerto):

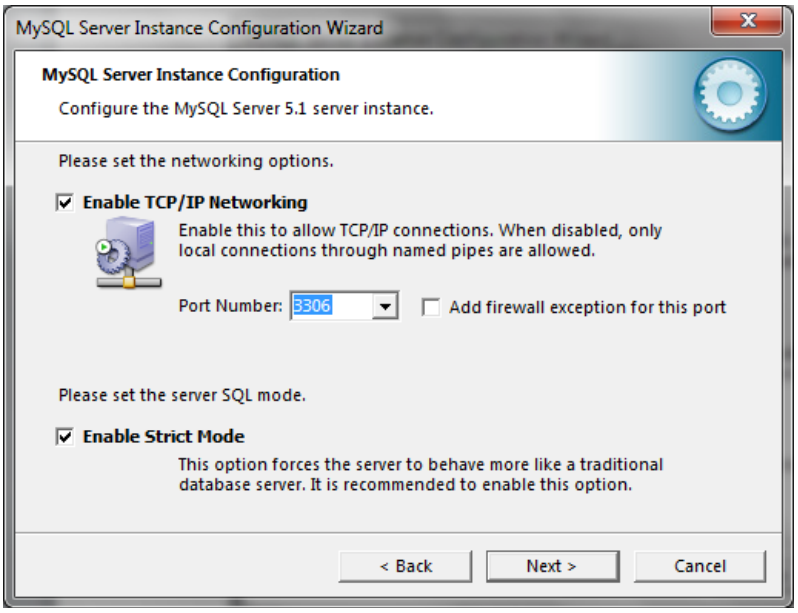


Fig 6.4 *Selección del tipo de red*

Más adelante se selecciona el juego de caracteres que se quiere utilizar, por defecto está marcado "Latin1" válido para Inglaterra y Europa.
El siguiente paso es importante pues se pide que se especifique el tipo de arranque de MySQL Server. Si se selecciona la primera opción ("Install As Windows Service") el programa de instalación creará un Servicio que será el encargado de ejecutar MySQL Server. También permite especificar el nombre del servicio y si se quiere que arranque automáticamente al iniciar el sistema ("Launch the MySQL Server automatically"). La segunda opción "Include Bin Directory in Windows PATH" añadirá las variables de entorno necesarias para la ejecución de los ficheros necesarios para iniciar MySQL . La opción recomendada es "Install As Windows Service":

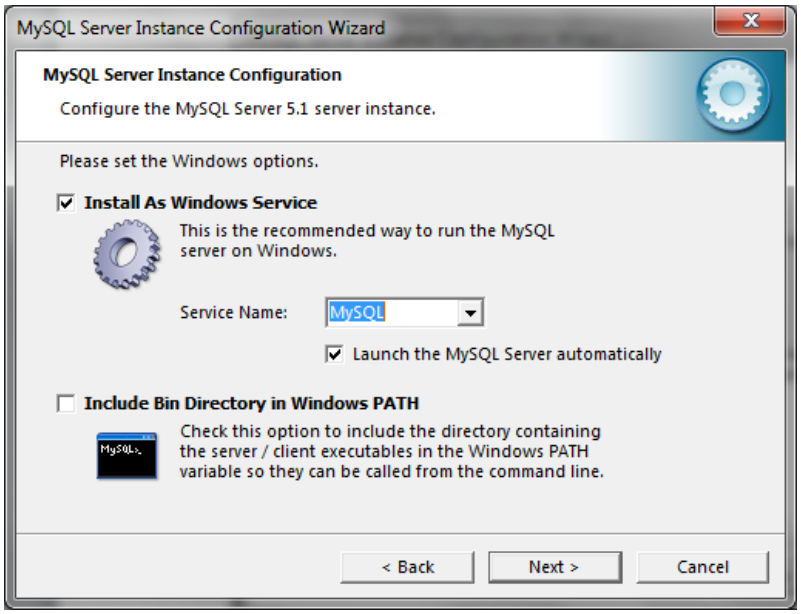


Fig 6.5 *Opciones de Windows*

Luego se introduce la contraseña de usuario administrador y se marcará la opción "Enable root access from remote machines" si se quiere que se pueda acceder como administrador desde otros equipos. Se pondrá como contraseña "root" puesto que en el fichero de configuración inicialmente está especificada así:

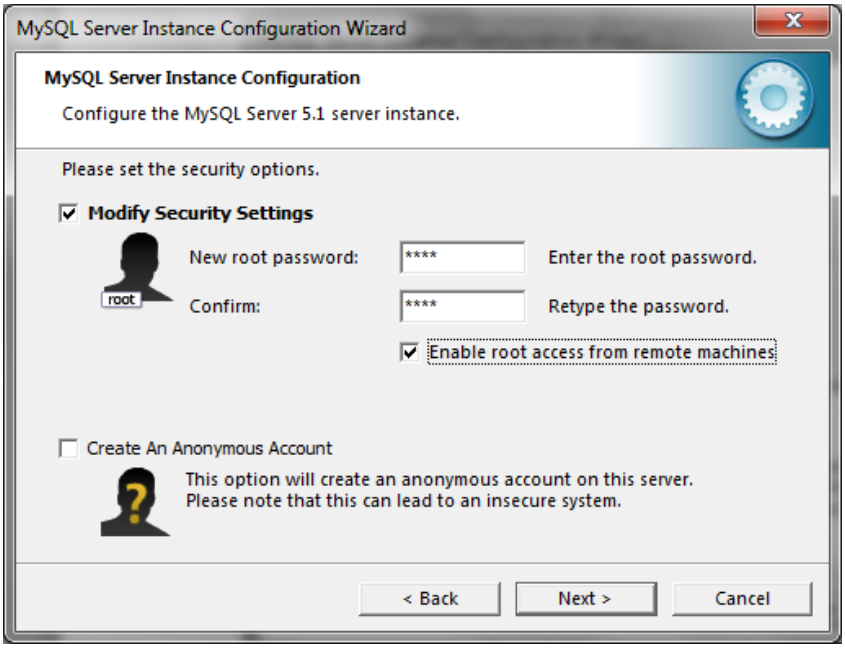


Fig 6.6 *Opciones de seguridad*

Por último se pulsa en "Execute" para finalizar la configuración de MySQL. Si no hay problemas mostrará esta ventana indicando que el proceso de instalación y configuración de MySQL Server ha terminado y se ha instalado e iniciado el Servicio que ejecutará MySQL:

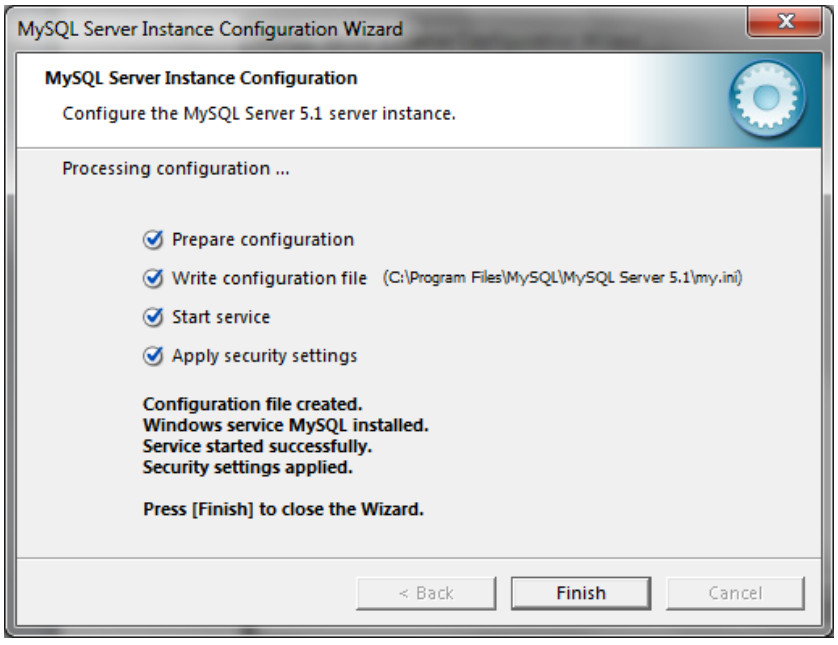


Fig 6.7 *Finalización de la configuración*

Una vez finalizada la instalación de MySQL Server 5.1 se debe proceder a la instalación del paquete de programas MySQL Tools 5.0. Su instalación es muy sencilla y no hay dificultad en el proceso. Al final de su instalación deberá aparecer la siguiente imagen:

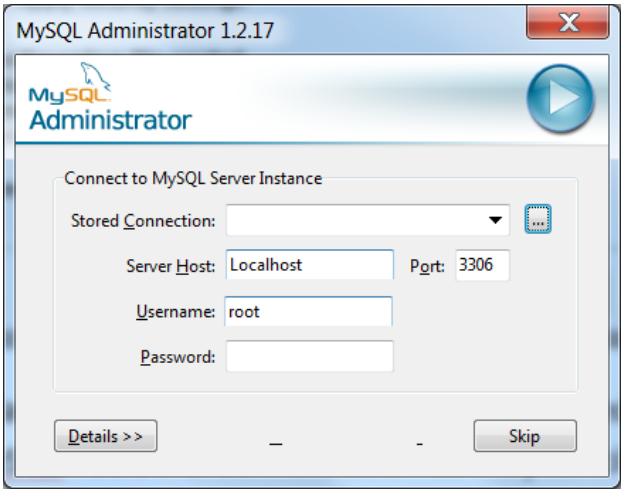


Fig 6.8 *Entrada de acceso a MySQL Administrator*

Se pone la contraseña utilizada en el programa anterior y ya está preparado para ser utilizado. Una vez finalizado la instalación de MySQL es necesario incluir dos drivers, "MySQL connector odbc" y "MySQL connector java". El primero sirve para que desde un sistema Microsoft Windows se pueda acceder a la base de datos más fácilmente (por ejemplo con Excel) y la otra para que exista una conexión con la aplicación Java. Este último ya está incluido en las aplicaciones WSN y WSNGUI4. Para la instalación del driver odbc solo es necesario pinchar en el ejecutable y seguir los pasos.

6.1.2 Creación de la tabla de datos.

Una vez acabada la instalación por completo se debería de proceder a la creación de una tabla de datos que almacenase todos los valores que son enviados desde la aplicación WSN. Pero hay una manera más fácil de hacer esto, y es por medio de un backup desde un archivo que contiene la información necesaria para crear la tabla con sus columnas y filas.
Primero se abre el programa MySQL Administrator y en el recuadro izquierdo se selecciona "Restore".

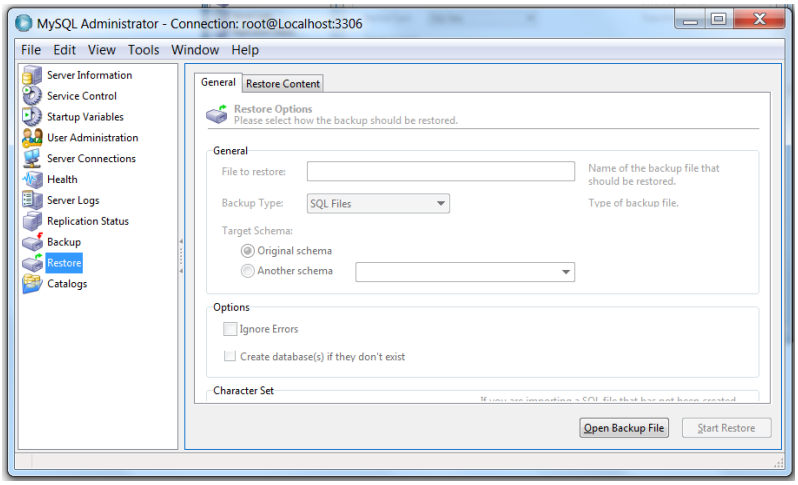


Fig 6.9 *Recuperación de una tabla prediseñada*

Donde pone "Open Back File" se selecciona el archivo mencionado anteriormente y se ejecuta "Start Restore". Este archivo puede encontrarse dentro del CD del proyecto en la carpeta "programas".
La parte que conlleva la instalación de MySQL estaría ya terminada. Como se ha dicho antes, este programa ofrece múltiples posibilidades de utilización. La forma de sacarle provecho se deja en manos del usuario, puesto que no es el objetivo de este proyecto. No obstante, se mostrará una forma útil de darle uso a la base de datos.

6.3 Ejemplo de aplicación.

Una buena utilidad que podría sacarse de la base de datos es la representación en Excel de los valores obtenidos de los sensores. A continuación se mostrará la forma de realizarlo.
Primeramente debe estar correctamente instalado el conector ODBC del que se ha hablado anteriormente.

6.3.1 Instalación del conector ODBC.

Para que pueda existir una conexión de datos entre Excel y MySQL se tiene que crear una nueva conexión desde "Orígenes de datos ODBC". Para ello se busca este programa dentro del Panel de control de Windows en Herramientas Administrativas. Una vez encontrado debería salir una ventana como la que se muestra a continuación:

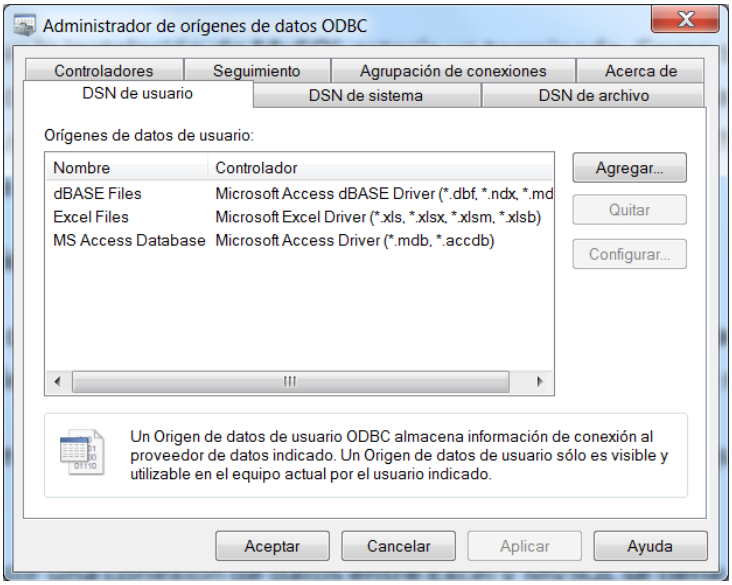


Fig 6.10 *Administrador del ODBC*

Dentro de la pestaña "DNS de usuario" se agrega un nuevo origen.

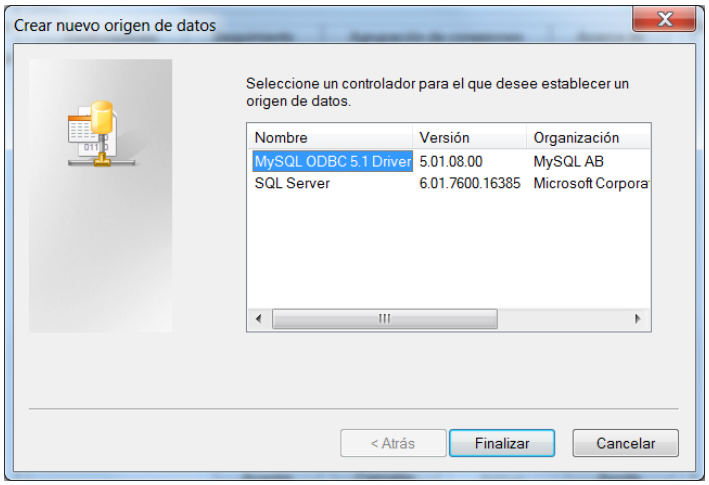


Fig 6.11 *Creación de un nuevo origen de datos*

Se selecciona el driver ODBC y se da a finalizar. Más adelante aparece una tabla que se debe de rellenar como se muestra a continuación:

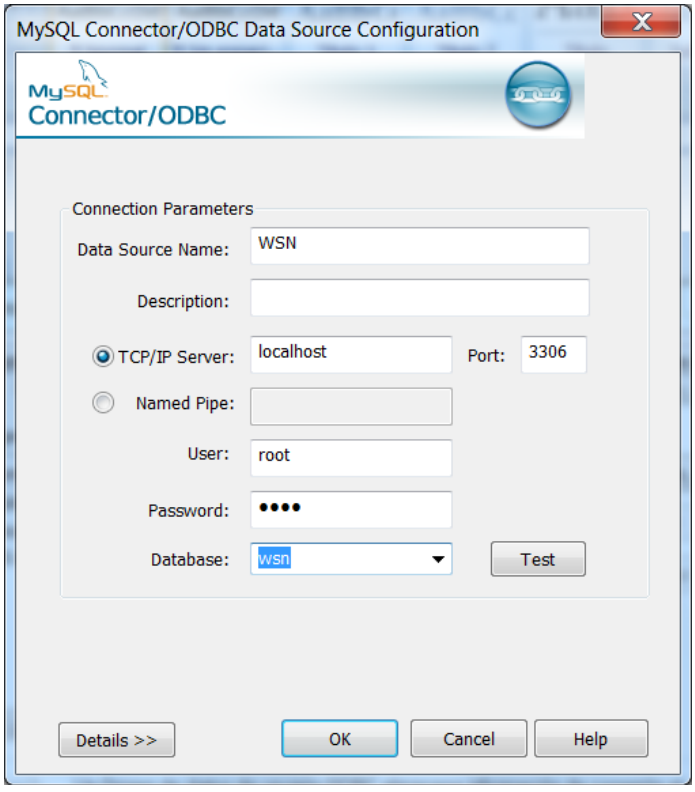


Fig 6.12 *Entrada al sistema ODBC*

El password ha de ser el mismo con el que se ha instalado el paquete de MySQL Tools. Es importante seleccionar la tabla creada en la base de datos. En este caso se llama "WSN". Se comprueba la conexión en "Test" y si ha sido satisfactoria se pincha en "OK". Ya está creado un nuevo origen de datos.

Se continua ahora con Excel. Dentro del programa, se selecciona la pestaña "Datos", se pincha en "de otras fuentes" y ahí en "desde el Asistente para la conexión de datos" se selecciona "DSN..." para introducir el origen de datos creado.

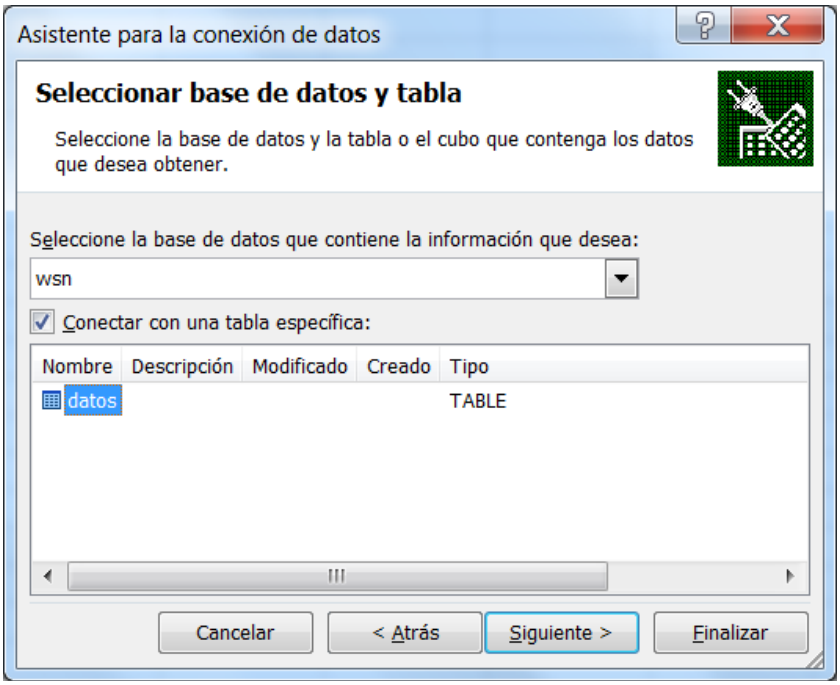


Fig 6.13 *Asistente para la conexión de datos*

Por último se escoge tabla "datos" y se finaliza. La manera de ver los datos ya es cuestión de gustos y necesidades. Se recomienda escoger la forma de visualización en "Tabla".

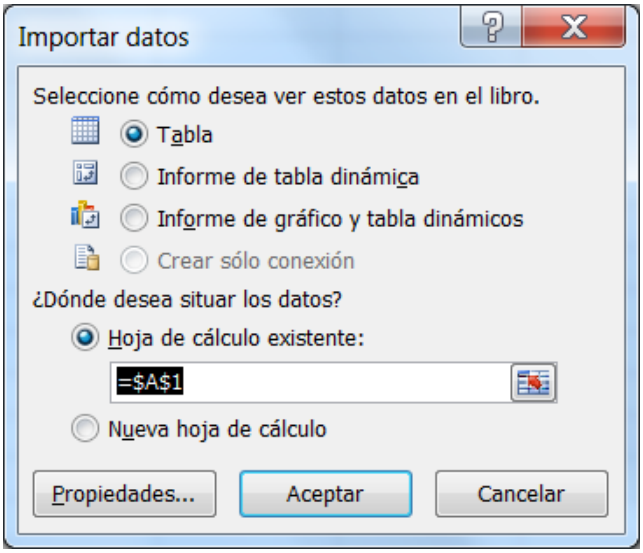


Fig 6.14 *Recuperación*

A continuación se muestra una manera de visualizar los datos en Excel.

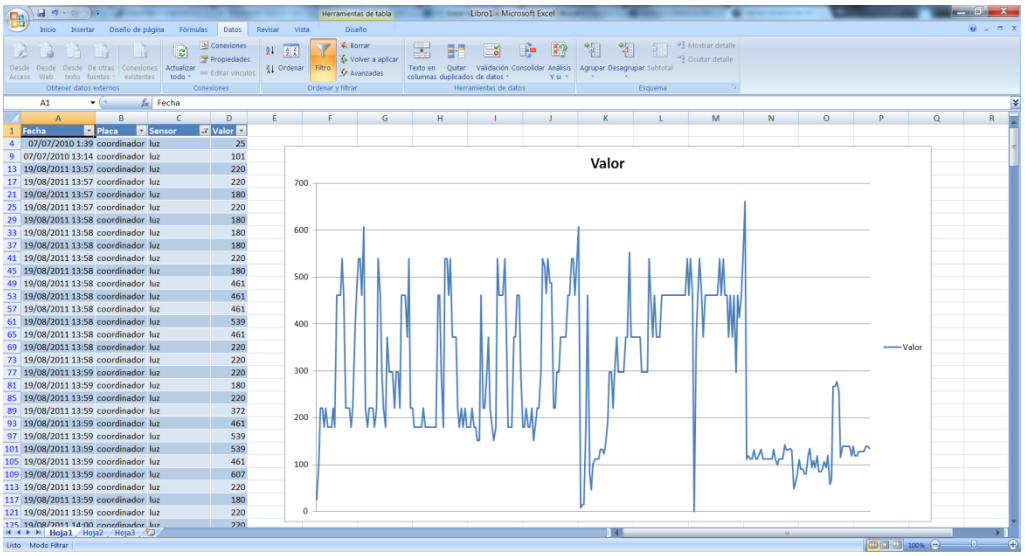


Fig 6.15 *Representación en gráfica Excel*

Esta forma tiene la ventaja de poder seleccionar directamente desde la primera fila la placa y el sensor que se desee visualizar. Además, pinchando en "Actualizar", es posible obtener los últimos datos introducidos en la base de datos.
Como se puede observar, MySQL ofrece muchas posibilidades a la hora de

6.4 Conclusiones.

El hacer uso de una base de datos en este proyecto ha brindado la facilidad de hacer una mejor gestión de los datos recibidos, pues da la posibilidad de leer y escribir en ella con total facilidad mediante un lenguaje propio de base de datos. Ha surgido la necesidad de crear este capítulo por la sencilla razón de despreocuparse con todo lo relacionado a la base de datos, pues no es del todo necesario saber manejar el servidor MySQL aunque sí recomendable, puesto que todo es automático y no necesita de mucha atención mientras estén ejecutandose las aplicaciones.

CAPÍTULO 7

APLICACIÓN WSNGUI4

En este capítulo se expondrá la aplicación principal de este proyecto WSNGui4, desarrollada especialmente para satisfacer la necesidad de elaborar un programa para visualizar en modo de gráficas los datos obtenidos por los sensores de las tarjetas.

Como siempre, se hará una pequeña introducción a la aplicación junto con una breve explicación de su instalación y uso. Finalmente habrá un análisis para cada código realizado en este programa.

7.1 Introducción al software WSNGui4.

WSNGui4 es una aplicación desarrollada en el lenguaje de programación Java. Su función en este proyecto es tener la capacidad de visualizar en forma de gráficas los datos que van introduciéndose en la base de datos MySQL por el coordinador de la red ZIgBee, a través de la aplicación WSN.

El nombre de WSNGui4 proviene de los varios diseños realizados que no han sido satisfactorios. El último y definitivo acabó siendo el cuarto diseño, de ahí que aparezca en su nombre un número 4.

Como se ha comentado, su funcionalidad es obtener en tiempo real y por medio de comandos propios de lenguajes de bases de datos, toda la información generada por el coordinador para luego visualizarla en forma de gráficas. Como se sabe, esta información es guardada a través del puerto serie por la aplicación WSN, que si se recuerda, es la encargada de leer los mensajes que llegan por el puerto al que está conectado dicha tarjeta e introducirlos en la base de datos MySQL en un formato ya estructurado y definido por la aplicación.

7.1.1 Instalación de WSNGui4.

Al igual que ocurre en la aplicación WSN, este programa tampoco requiere ninguna instalación. Solo es necesario ejecutar el archivo "star.cmd" que se encuentra en la carpeta WSNGui4 el cual también se puede localizar dentro del CD ROM de este proyecto. Se recuerda que para que tengan un buen funcionamiento estas aplicaciones es necesario tener instalado los paquetes de Java en el equipo y el driver "MySQL connector Java" para una correcta comunicación entre la aplicación y la base de datos.

7.1.2 Funcionamiento del programa.

Esta aplicación al tener una única función de hacer consultas MySQL es totalmente independiente del otro programa WSN. Sólo representa en gráficas todo lo que está guardado en la base de datos. Por lo tanto, para que sea efectivo su principal función, debe estar instalado y encendido el servidor MySQL como se ha indicado en el capítulo pertinente. Evidentemente si la otra aplicación no introduce nueva información a la base de datos, no habrán datos que representar.

Para el funcionamiento del programa también se efectúa de la misma manera que en WSN. En la carpeta del programa se encuentra el archivo "star.cmd" que inicializa la aplicación. También se trata de una instrucción ".jar" que abre la ventana de comandos y ejecuta la aplicación. Al pulsarlo surge el "cmd" con dicha instrucción y en cuestión de segundo aparece el entorno de la aplicación.

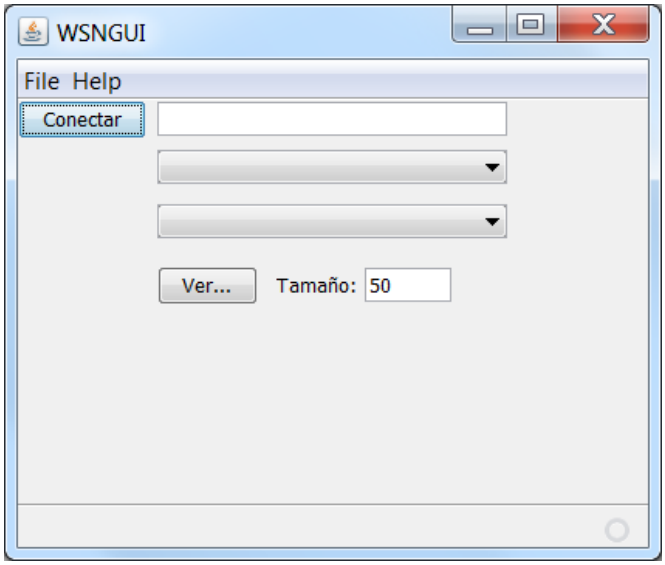


Fig 7.1 *Ventana principal de la aplicación*

En este sencillo programa puede encontrarse los siguientes botones y ventanas de texto:

- Botón conectar: Realiza una conexión con la base de datos.
- Cuadro de texto: Indica si la conexión ha sido satisfactoria o no.
- Menús desplegables: Seleccionan la placa y el tipo de sensor que se quiere representar.
- Cuadro de Tamaño: Cantidad de datos del sensor seleccionado que se quiere mostrar. Por defecto aparecerán 50 datos.
- Botón Ver: Muestra finalmente la gráfica del sensor seleccionado con la cantidad de datos deseada.

Para comenzar con su funcionamiento primeramente debe estar encendido el servidor MySQL. Como ya se ha comentado, existe una herramienta llamada "MySQL Sistem Tray Monitor" que hace funcionar el servidor de una manera más rápida. Si surge un icono como el que se muestra a continuación señala que el servidor está en funcionamiento y ya está preparado para recibir mensajes desde el coordinador y leerlos por este programa.

Fig 7.2 *Icono de actividad de MySQL*

Justo al pulsar el botón "Conectar" debe aparecer "Conectado con la base de datos" e inmediatamente se rellenan los despegables con todas las tarjetas y sus respectivos sensores que han enviado sus datos a MySQL.

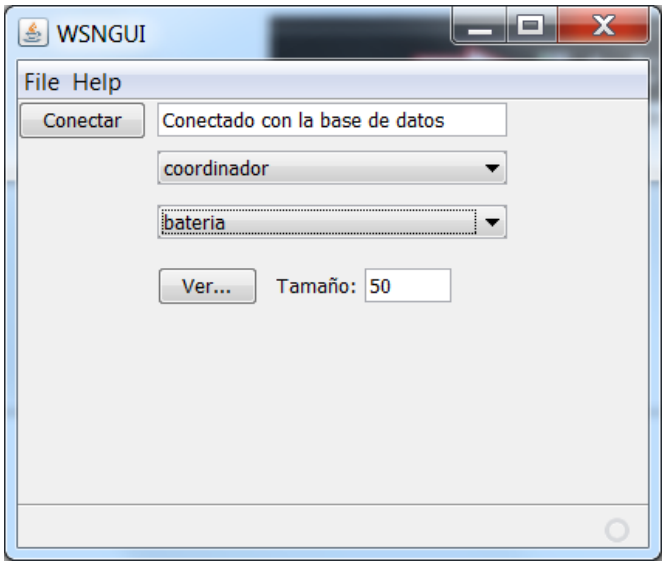


Fig 7.3 *Aplicación en funcionamiento*

Se selecciona la tarjeta y su sensor a visualizar, se indica el tamaño de datos a mostrar y se pincha en "Ver". Al instante surgirá la gráfica deseada.

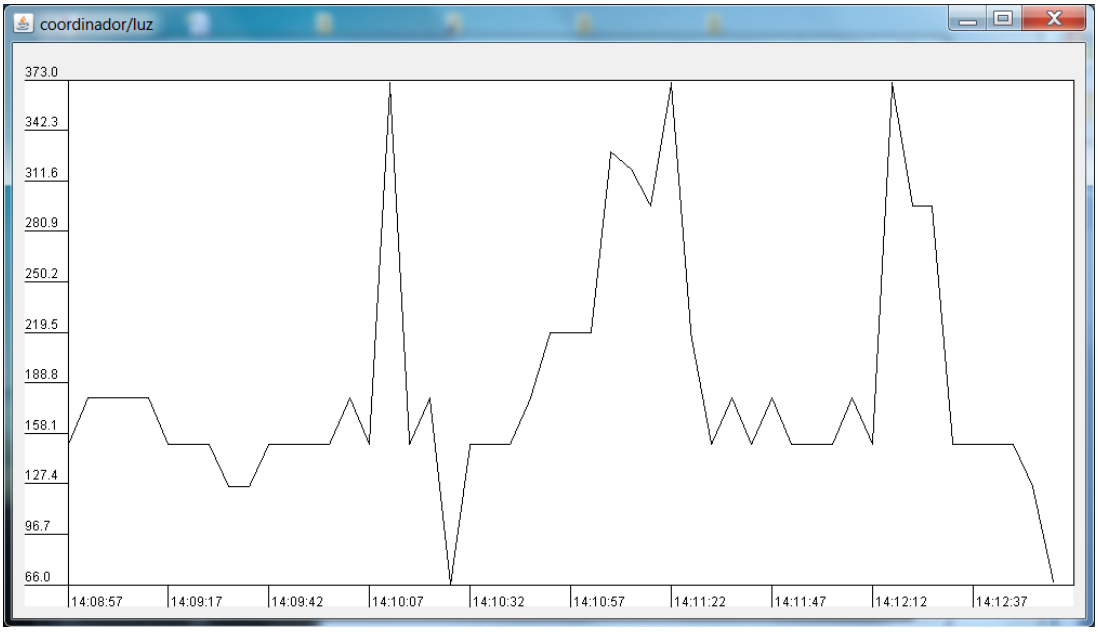


Fig 7.4 *Gráfica del sensor seleccionado*

Como puede apreciarse, en el eje X indica la hora que fue tomada el dato medido por el sensor y en el eje Y el valor del mismo. Cada ventana puede agrandarse todo lo que se desee para una mejor visualización. Además, en cada esquina de las ventanas tiene el detalle de aparecer el nombre de la tarjeta y del sensor seleccionado.

7.1.2.1 Cambiar el nombre de las tarjetas y sensores.

Seguramente a la hora de añadir o modificar una tarjeta o sensor se reconocerá en el programa con una numeración extraña.

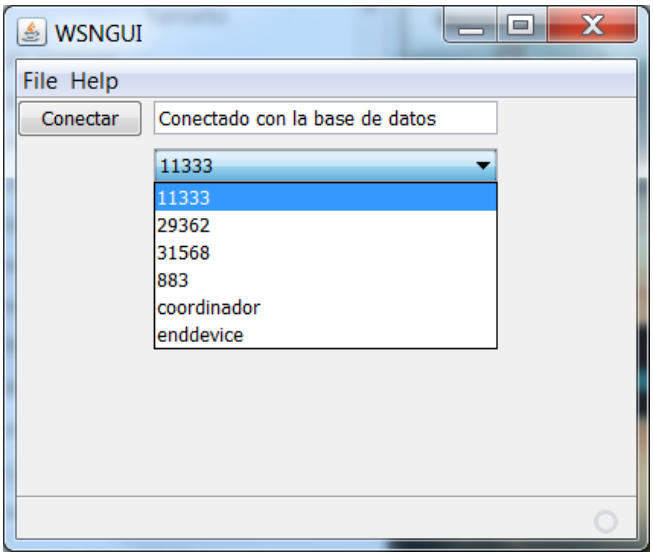


Fig 7.5 *Reconocimiento extraño*

Esta situación no es ni mucho menos problemática. Sucede por ejemplo cuando se inicia un dispositivo nuevo en la red ZigBee. A este dispositivo se le agrega un nuevo "ShortAddr", excepto el Coordinador, el cual siempre será el "0". Es una manera que tiene el código instalado en las tarjetas de reconocer nuevos dispositivos. Si se quisiera modificarle el nombre a la tarjeta adherida no habría nada más que modificar el archivo de texto "Configuración.properties" que se encuentra en la carpetas de los programas WSNGUI4 y WSN. En los dos casos se trata del mismo archivo y como se sabe de capítulos anteriores su finalidad es poder pasar parámetros a los programas.

Pues bien, simplemente sabiendo el "ShortAddr" del dispositivo que al que se le quiere modificar el nombre, solo habría que sustituir o añadir en el archivo de texto mencionado la numeración por el nombre que se desee. Hay dos maneras de saber esa numeración del dispositivo. La primera es las más fácil y evidente. Se trata de reconocer el nuevo número al iniciar el programa WSNGUI4 con el dispositivo adherido a la red y en funcionamiento, es decir, que haya mandado sus primeros mensajes a MySQL. Y la otra forma se hace abriendo el programa "MySQL Query Browser". Se introduce la contraseña por defecto "root" y se inicializa. En la sección de la parte derecha "Schemata" se pincha dos veces en la tabla "datos" y surgirá en el cuadro principal algo parecido como se muestra a continuación:

2011-09-25 21:07:57	coordinador	bateria	660
2011-09-25 21:07:57	coordinador	temperatura	28
2011-09-25 21:07:57	coordinador	luz	117
2011-09-25 21:07:57	coordinador	potenciometro	1.30224609375
2011-09-25 21:07:59	30376	0	425
2011-09-25 21:07:59	30376	1	27
2011-09-25 21:07:59	30376	2	111
2011-09-25 21:07:59	30376	3	280

2880 rows fetched in 0,0326s (0,0033s)

Fig 7.6 *Tabla de datos*

Como puede observarse en la imagen, los últimos mensajes añadidos a la base de datos pertenecerían a la tarjeta adherida a la red inalámbrica, en este caso el shortAddr sería "30376".

Una vez sabido el número de identificación perteneciente a la tarjeta iniciada se intercambia en los archivos "Configuración.properties" de los dos programas, como se muestra a continuación, y listo.

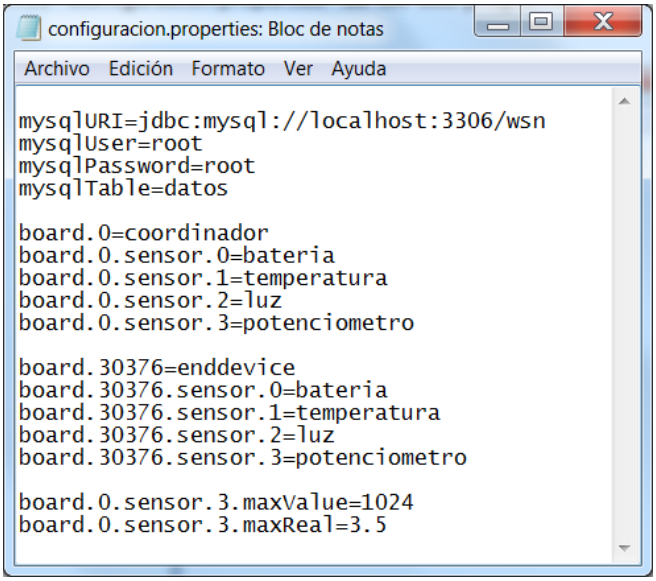


Fig 7.7 *Modificación del archivo "Configuración.properties"*

Si se tratase de una nueva tarjeta adherida, se introduce un nuevo apartado en los archivos con la misma estructura que aparecen los demás. De la misma manera se actúa si se tratase de un nuevo sensor añadido a una tarjeta ya existente. Es evidente que se puede establecer el nombre que se desee para un mejor reconcomiendo en las gráficas.

Pero eso no es todo. Como se sabe, los mensajes que ya están insertados en la base de datos, aparecerán en el monitor de la aplicación, y con ello, también los pertenecientes a las tarjetas con numeración extraña. Se recomienda que, una vez hecho la modificación deseada, se eliminen de la base de datos para evitar que aparezcan en la aplicación.

7.2 Desarrollo de la aplicación.

A continuación se mostrará una explicación del desarrollo de la aplicación. Este programa, al diferencia que con el WSN, se ha diseñado con un entorno de desarrollo integrado libre conocido como "NetBeans", hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso y puede descargarse desde su página oficial de internet.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

Estas explicaciones están fundamentadas en tres archivos ".java" que pueden encontrarse dentro de la carpeta del programa WSNGui4. Solo se hará hincapié en el código fundamental de la aplicación. Los demás archivos son creados automáticamente por el entorno de NetsBeans al desarrollarse la aplicación.

También se recuerda que esta aplicación contiene el mismo archivo "Configuración.properties" que la aplicación WSN, el cual ya se explicó en capítulos anteriores para que servía.

7.2.1 WSNGUIApp.

Se trata de la clase principal de esta aplicación. La manera que tiene el código de esta aplicación de leer desde la base de datos es la siguiente: Al pulsar el botón "Conectar" sencillamente llama a la clase "ObténConexión" y a continuación ejecuta una simple función:

```
ObtenConexion();
      if (this.conn!=null) jTextField1.setText("Conectado con la
      base de datos");
      else jTextField1.setText("Error de conexión");
```

Fig 7.8 *Código para abrir una nueva conexión con la BBDD*

Aquí se dice que si el objeto "This.conn" en la conexión no es nula aparecerá en el recuadro pegado al botón un texto que dirá "Conectado con la base de datos" y si es nulo, es decir, no ha habido conexión pues mostrará "Error de conexión".
Volviendo a la clase "ObtenConexion", una vez comprobada la conexión se ejecuta "DriveManager" que es una clase genérica con métodos estáticos relacionados con Java que junto con "getConnection" obtiene los tres parámetros necesarios para conectarse con la base de datos, que son nombre de usuario (mysqlUser), el password y la cadena de conexión (mysqlURI). Estos tres parámetros se obtienen a partir del fichero de configuración, que si se recuerda, es un fichero que sirve para pasar parámetros al programa y es cargado con la función "inicializaConfiguración".

```
public void ObtenConexion(){
      if (this.conn==null)
      try      {this.conn      =      DriverManager.getConnection
      (prp.getProperty("mysqlURI"),
      prp.getProperty("mysqlUser"),
      prp.getProperty("mysqlPassword"));
      RellenaPlacas();
```

Fig 7.9 *Obtención de las claves de acceso a la BBDD*

MySQLUser y MySQLPassword son los parámetros de identificación y MySQL Uri es la cadena de conexión que apunta a la base de datos MySQL en el puerto 3306. Si hubiese algún error pues saldría alguna excepción. "ObtenConexion" hace otra cosa más. Además de obtener la conexión, llama a "RellenaPlacas".

Esta clase es la encargada de rellenar los dos cuadros de selección que hay, mediante una consulta a la base de datos, cogiendo así las posibles placas que puedan haber conectadas, y para cada placa que sensores existe. Lo primero que hace es comprobar si hay una conexión con la base de datos, si no la hubiera prueba a obtener otra nueva, y si no lo consigue termina la función sin rellenar los cuadros de selección.

```
stm = this.conn.createStatement();
String sql= "select placa from datos group by placa";
rs =stm.executeQuery(sql);
```

Fig 7.10 *Selección de la placa*

La manera que tiene "RellenaPlacas" de funcionar es mediante el objeto principal "This.conn.createStatement" que hace una llamada dentro del objeto conexión y crea una sentencia sql con este String: "select placa from datos group by placa". Por medio de este comando sql consulta en MySQL dentro de la tabla "datos" las distintas placas que existen, y las agrupa. Con lo cual a la conexión se le crea una sentencia "stm.executeQuery(sql)" con la orden dicha. Finalmente la sentencia devolverá un" ResultSet" (rs) con los valores pedidos.

Esta será la manera que se usará con posterioridad para comunicarse con la base de datos y empezar a leer valores e insertarlos dentro de los llamados ComboBox. " ResultSet" es pues un conjunto de resultados, que luego serán introducidos ahí.

```
jComboBox1.removeAllItems();
while(rs.next()){
jComboBox1.addItem(rs.getString("placa"));
                  }
```

Fig 7.11 *Código para rellenar el combobox de las placas*

Viendo el código se observa como se le añade al primer combo el item obtenido del "rs", siempre y cuando "rs.next" no sea nulo, es decir, mientras existan valores en "rs". Lo que ocurre a continuación en el diseño simplemente por interactuar con este objeto y rellenarlo con datos se lanza otra llamada implícitamente al haber modificaciones en el combobox, por lo que se selecciona por defecto el primer elemento, es decir, la primera placa de toda la lista.

```
String placa = (String) jComboBox1.getSelectedItem();
RellenaSensores(placa);
```

Fig 7.12 *Código para rellenar el combobox de los sensores*

Y al tener ya un elemento seleccionado se ejecuta la clase "RellenaSensores" con el objeto seleccionado por medio de un String llamado "placa". Si se seleccionase otro elemento del combo, por el hecho de pincharle en otro objeto, se borraría el actual y se rellenaría el otro combo con los sensores de la placa seleccionada.

La manera que tendría de rellenarse el otro combo con los sensores de la placa seleccionada sería mediante la clase "RellenaSensores", la cual se le ha pasado como parámetro dicha placa. Por lo tanto, por medio de una consulta apropiada a MySQL se hace una lista de la tabla "datos" con aquellos sensores que son de una misma placa y los agrupa por tipos de sensores. Esta clase actuaría de la misma manera que lo ha hecho "RellenaPlacas". Primero se comprueba que haya una conexión, si no la hay, intentará obtenerla. Se crea una sentencia en la cual se ejecuta una consulta sql con la "placa" seleccionada como parámetro. Finalmente le devuelve un" ResultSet" agrupado por sensor con todas las filas que pertenecen a esa placa. Las demás filas se descartan y se queda con los distintos sensores de esa placa. Finalmente se borran todos los items que se encuentran en el segundo combo y lo rellena con los valores obtenidos de "ResultSet".

Llegados a este punto se ha seleccionado una placa y un sensor. Se verá ahora como actúa el botón "ver". Al pulsar este botón se crea el objeto principal "datos" de la clase "DatosSensorView". A este objeto se le pasa como parámetros la placa y el sensor seleccionados de los combos y crea una especie de pantallita en blanco al que se le irá pintando los datos.

```
DatosSensorView datos = new
DatosSensorView(this,(String)jComboBox1.getSelectedItem(),(String)
jComboBox2.getSelectedItem());
```

Fig 7.13 *Obtención de la placa y el sensor seleccionado*

La manera de comunicarse "DatosSensorView" con WSNGUI4 es mediante un puntero al padre que es creado justo en el momento de haberse pulsado el botón. La aplicación padre sería WSNGUI4. A continuación se ejecuta la función "txtTam.getText" encargada de coger de la aplicación del campo de texto (situado al lado de "tamaño") un número para indicar cuantos datos se quieren visualizar en la pantalla, luego es convertido a entero y es pasado mediante el método "createDatos" al objeto "datos".

```
datos.createDatos(Integer.parseInt(txtTam.getText()));
```

Fig 7.14 *Obtención del número "tamaño"*

El algoritmo encargado de rellenar la parte visible de "DatosSensorView" se encuentra en el objeto "DatosSensor". Este algoritmo normaliza los datos, los pinta, crea las escalas, etc.
Por decirlo de otra manera, al pulsar el botón "ver" se ejecuta una vista de la clase "DatosSensorView", que internamente está funcionando el objeto "DatosSensor" encargado de rellenarla mediante un algoritmo que inserta las escalas y va pintando los datos sobre una gráfica. Por lo que saldrá tantas vistas como veces se haya pulsado el botón "ver".
Además, cada ventana abierta crea un hilo de ejecución independiente de la aplicación principal, en bucle infinito que, cada dos segundos, borra un dato de la gráfica mediante la función "clearDatos" y con un puntero al padre "WSNGUI4" obtiene el siguiente dato a insertar en la gráfica con la función "llenaDatos", de tal manera que se va eliminando de la gráfica el último dato metido y a la misma vez introduciendo un nuevo dato. La manera de obtener los datos se consigue por medio de una consulta a la base de datos y siempre por el mismo procedimiento.

```
String sql= "select fecha,valor from datos where
placa='"+data.getPlaca()+"' and sensor='"+data.getSensor()+"'
order by fecha";
rs =stm.executeQuery(sql);
```

Fig 7.15 *Consulta a MySQL*

Primero obtiene una conexión con MySQL, se crea una sentencia en la cual se ejecuta una consulta sql con la placa y sensor seleccionados como parámetros. Los datos obtenidos, son pasados al objeto "data" para luego rellenar a "DatosSensorView". Estos datos son los últimos valores obtenidos del sensor seleccionado, están ordenadas por fecha y van actualizándose cada dos segundos. A continuación el objeto "data" es insertado dentro de la vista y es pintado en la gráfica por medio de "DatosSensor".

7.3 Conclusiones.

La creación de la aplicación WSNGui4 ha cubierto la necesidad de obtener una representación en gráficas de los datos guardados en la base de datos. Se puede considerar que la representación de los datos de los sensores es en tiempo real pues está actualizándose continuamente y obteniendo de inmediato los valores guardados. No obstante es una aplicación básica, pero muy práctica, en la que podría añadirse por ejemplo nuevas formas de visualización y la conexión al servidor MySQL vía internet más fácilmente, pues si se desease hacerlo, habría que modifar el conocido archivo "configuacion.properties" la primera clave encarga de obtener una conexión con el servidor. Aún así, me puedo sentir satisfecho de la aplicación creada.

CAPÍTULO 8

CONCLUSIONES Y TRABAJOS FUTUROS

8.1 Conclusiones.

En este proyecto se ha llevado a cabo un estudio en profundidad de las redes de sensores inalámbricos existentes y se ha estudiado un kit comercial para la implantación de una red de sensores inalámbrica.

Se ha desarrollado un sistema de monitorización mediante redes de sensores inalámbricas, utilizando el kit de desarrollo ZigBee Amp comercializado por la empresa Meshnetics, una interfaz software que permita la monitorización y gestión de los datos capturados por medio del uso del servidor MySQL.

Se ha aprovechado el código que manejarían las tarjetas, conforme al protocolo ZigBee, y modificado para añadir nuevos sensores a la red. Para ello, se ha llevado a cabo un estudio meticuloso del código que permitiese su correcta modificación.

Puesto que el programa WSNDemo que dispone el kit no es posible la visualización de otros sensores que no fuesen los que contienen las tarjetas, se ha diseñado dos aplicaciones que simulasen este programa conectadas por medio de una base de datos. La primera se encarga de gestionar los puertos del ordenador para crear una conexión con la tarjeta coordinadora de la red inalámbrica, la cual recibe vía radio todos los mensajes que envían todas las demás tarjetas; y los almacena en el servidor MySQL. La segunda se encarga de obtener una conexión con MySQL para obtener los datos guardados para luego visualizarlos mediante gráficas.

La experiencia de uso del servidor MySQL ha sido bastante sencilla y satisfactoria. Apenas se ha tenido dificultad para su uso, tanto para añadir nuevos datos como para leerlos posteriormente. Además, su utilización vía internet deja muchas posibilidades al alcance del usuario. Existen varios programas que circulan por la red gratuitamente que brindan muchas ventajas para su gestión. Uno de ellos se habla detalladamente en el capítulo correspondiente.

Finalmente podría llegarse a la conclusión de que a pesar de haber creado unas aplicaciones muy básicas, me puedo sentir satisfecho por su practicidad y fiabilidad.

8.2 Trabajos futuros.

Partiendo de la realización de este proyecto, se proponen varias mejoras que se podrían hacer para trabajos futuros:

- La posibilidad de añadir otros tipos de sensores con diferentes interfaces, tipo como IRQ, I2C, SPI, UART o 1-wire con un mejor estudio del repertorio de APIs para su utilización en los recursos hardware del módulo (EEPROM, app, sleep, y watchdog timers) y controladores de referencia para diseños rápidos e integración sin problemas, como la BSP (Board Support Package) que incluye un completo lote de drivers para controlar

periféricos estándar (sensores, chip UID), colocados en una tarjeta MeshBean de desarrollo.

- La mejora de la aplicación WSNGui4 con la inclusión de otras características de visualización de gráficas, así como un práctico botón que ofrezca la posibilidad de guardar los datos recibidos en un fichero de texto. No obstante, no se ha considerado oportuno realizarlo, pues la base de datos ofrece muchas mejores posibilidades. Además, podría añadirse otro botón que permitiese el control del tiempo de intervalo entre las lecturas de los sensores para un mejor aprovechamiento de la batería.
- La base de datos se podría implementar posteriormente con lenguaje SQL utilizando herramienta gráficas que facilitan la implementación como PhpMyAdmin. E incluso se podría tener un servidor alojado en una web de tipo PHP que permita la consulta en directo a cualquier persona a los datos recibidos.

Podrían añadirse muchas más mejoras, todo dependería de la imaginación de uno y de las necesidades que puedan crearse.

Bibliografía:

[WSN] Wireless Sensor Network: Yang Yu,Victor K Prasanna,Bhaskar Krishnamachari. "Information processing and routing in wireless sensor networks" Ed: World Scientific, (2006)

[IEEE 802.15.4] Definición del estándar IEEE 802.15.4: *"IEEE 802.15.4 - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)"*, revisión 2006.

[CSMA-CA] *Carrier sense multiple access with collision avoidance*, Acceso múltiple por detección de portadora, evitando colisiones. Información online en: Pablo Brenner, "A Technical Tutorial on the IEE 802.11 Protocol", BreezeCOM, 1997 http://www.sss-mag.com/pdf/802_11tut.pdf

[OSI] Open *System Interconnection*, Interconexión de sistemas abiertos, Disponible online en: http://standards.iso.org/ittf/PubliclyAvailableStandards/

[MAC] Media Access Control. Es una subcapa de la capa Enlace de datos del modelo OSI y su función principal es gestionar el acceso al canal de comunicación y el direccionamiento. Más información: A. Tanenbaum, "Computer networks" 4ª Ed. Prentice Hall, Año 2003.Cap 4.

[Zigbee] Zigbee protocol, Disponible online en: http://www.zigbee.org/Products/DownloadZigBeeTechnicalDocuments.aspx

[GTS] GTS , Definición en: "On the IEEE 802.15.4 MAC Layer Attacks: GTS Attack", CAP3.

[6LowPan] IPv6 over *Low power Wireless Personal Area Networks,* IPv6 sobre Redes de área personal inalámbricas de bajo consumo. Más información online en: http://datatracker.ietf.org/wg/6lowpan/charter/

[TinyOS] TinyOS Operating System. Disponible on-line en: http://www.tinyos.net

[Contiki] Contiki Operation System. Disponible on-line en: http://www.sics.se/contiki/

[LiteOs] LiteOs. Sistema operativo de tiempo real y código abierto con base UNIX diseñador para redes de sensores inalámbricos. Disponible on-line en: http://www.liteos.net/

[NutOs] Pequeño sistema operativo de tiempo real diseñado para CPUs de 8 bits. Más información online en: http://www.ethernut.de/en/firmware/nutos.html

[Nano-RK] Nano-RK es un sistema operativo de tiempo real para redes de sensores inalámbricos desarrollado en la Universidad Carnegie Mellon. Más información en: http://www.nanork.org/.

[Mantis] Mantis, Sistema operativo con soporte de múltiples hilos desarrollado por la Universidad de Colorado y escrito en C, para redes de sensores inalámbricos. Más información disponible on-line: http://mantisos.org

[NesC] Lenguaje de programación NesC: Más información online: http://nescc.sourceforge.net/

[VNC] Virtual Network Computing. Protocolo de control del ordenador a distancia. Disponible definición del protocolo online en: http://www.realvnc.com/docs/rfbproto.pdf

[Telnet] *Telecomunication Network*. Protocolo de acceso a máquinas conectadas en red para manejarlas remotamente. Disponible definición del protocolo online en : http://www.faqs.org/rfcs/rfc854.html

[SmartDust] Proyecto financiado por el DARPA y desarrollado por la universidad de Berkeley para construir motes con sensores de de tamaño inferior a 1 mm cuadrado. http://robotics.eecs.berkeley.edu/~pister/SmartDust/

[FTDI] Compañía especializa en la fabricación de chips para la conversión de diversos interfaces a USB. Disponible más información online en: http://www.ftdichip.com/

[Sensiron AG] Empresa líder en la fabricación de sensores de humedad y flujo en líquidos. Listado de producto disponible online en: http://www.sensirion.com/

[Libelium] Empresa desarrolladora de WaspMote. Disponible on-line: http://www.libelium.com/

[Meshnetics] Empresa creadora y distribuidora del kit de desarrollo ZigBit. Disponible on-line: http://www.meshnetics.com/

[Picdem Z] Kit de desarrollo de la casa Microchip para redes 802.11.4. Disponible on-line: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=

[eZ430-RF2480] Kit de desarrollo para redes 802.11.4 de Texas Instruments. Disponible on-line en: www.ti.com/eZ430-RF2480.

[IAR *Embedded Workbench Kickstart]* Entorno de desarrollo para construir y depurar aplicaciones basadas en el microcontrolador MSP430. Descarga libre online en : http://focus.ti.com/docs/toolsw/folders/print/iar-kickstart.html

[Zolertia Discovery kit Z1] Kit de desarrollo lanzado por la empresa Zolertia. Encontramos más información online en: http://www.zolertia.com/products/Z1-starter-kit

[Digi] Empresa distribuidora de Xbee y Xbee Pro tras la adquisición de Maxstream empresa que los desarrolló. Disponible on-line: http://www.digi.com

[Mote] Mote – Mobile transmissions elements. Consiste en una CPU+Radio+Sensores. Se le llama así a los nodos en las redes de sensores.

[Kit ZigBit] Kit de desarrollo facilitado por la empresa comercial NextFor. Disponible on-line: http://www.nextfor.com/nextforweb/default.asp?PAG=meshnetics/meshnetics.html&SUB=opcsistemascontrol.html