Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

Diseño e Implementación de componentes software para dar soporte a la plataforma MEWiN en el sistema operativo Contiki

Titulación: Ingeniería Técnica Industrial
Intensificación: Electrónica Industrial
Alumno: Honorio Navarro Hellín
Director/a/s: D. Juan Antonio López Riquelme
D. Roque Torres Sánchez

Cartagena, 5 de Septiembre de 2011

A mis directores de proyecto, Pencho y todo "el grupo motero"

A mis padres y hermano

A mi abuelo

Índice de capítulos:

Índice de Figuras:

Índice de Tablas:

Capítulo 1

Introducción y objetivos

En el siguiente apartado se pretende introducir al lector en el propósito del presente proyecto que tiene como título: "Diseño e Implementación de componentes software para dar soporte a la plataforma MEWiN en el sistema operativo Contiki". El proyecto fue desarrollado durante el curso académico 2010/2011 y corresponde a la Titulación: Ingeniería Técnica Industrial: Especialidad en Electrónica Industrial.

Las redes de sensores inalámbricas (Wireless Sensor Network, WSN) son un conjunto de dispositivos con capacidad de tratar información procedente de sensores, que establecen una comunicación sin cables interconectados entre sí a través de una red inalámbrica y a su vez conectados a un sistema central en el que se recopilará la información recogida por cada uno de los sensores.

En la actualidad, las redes inalámbricas de sensores están en auge y su uso es cada vez más abundante en multitud de campos, tales como la medicina, agricultura, medio ambiente, domótica, etc.

El grupo de investigación DSIE, perteneciente a la Universidad Politécnica de Cartagena, lleva varios años trabajando en el desarrollo de dispositivos de redes de sensores inalámbricos, que han dado como conclusión la electrónica MEWiN, sistema modulable para la implantación de nodos de redes inalámbricas.

El presente proyecto final de carrera que tiene como título: "Diseño e Implementación de componentes software para dar soporte a la plataforma MEWiN en el sistema operativo Contiki" pretende dar soporte software al dispositivo MEWiN desarrollado.

En la actualidad el sistema operativo Contiki es uno de los más utilizados en el ámbito de las redes inalámbricas de sensores, dada su característica de código libre y su alta escalabilidad. La mayoría de fabricantes de dispositivos electrónicos usados en redes de sensores tienen en proyecto dar soporte en este sistema operativo a sus equipos, con lo que se ha considerado importante compatibilizar un dispositivo hardware desarrollado en el seno de la universidad con un sistema operativo tan popular y con las ventajas de Contiki.

Para llevar a cabo esta labor, el proyecto se ha desarrollado en varias fases que, en si mismas, constituyen objetivos intermedios del proyecto:

1) Búsqueda, estudio y análisis del estado del arte en dispositivos WSN y en sistemas operativos.
2) Análisis de las limitaciones del sistema ZigBee de Texas utilizado en el sistema MEWiN actualmente. Comparativa con Contiki
3) Instalación y configuración del sistema operativo Contiki en Ubuntu bajo LINUX.
4) Adaptación del sistema para poder compilar y cargar código en el dispositivo MEWIN.
5) Creación de las bibliotecas necesarias para poder utilizar de manera intuitiva y sencilla los periféricos de MEWiN en Contiki: modulo de almacenamiento en tarjeta SD, RTC y módulo de radio.

El PFC se ha distribuido en los siguientes capítulos además del presente:

1) Estado del arte, donde se presentan y discuten los dispositivos hardware y software disponibles con sus ventajas e inconvenientes.

2) Arquitectura del sistema, donde se presenta el sistema Contiki estructura y disposición frente a ZigBee. Se presenta también el hardware MEWiN con sus periféricos.

3) Implementación y desarrollo software donde se centran el núcleo del código de Contiki, compilación, APIs, librerías, etc…

4) Conclusiones y futuros trabajos.

Capítulo 2

Estado del arte.

En el siguiente apartado se presentará la actualidad de las redes de sensores inalámbricas (WSN) así como la tecnologías (tanto software como hardware) asociadas a este tipo de redes.

En primer lugar se explicará el funcionamiento, aplicaciones y elementos de una red de sensores. Se continuará con las características y requisitos fundamentales en una WSN. A continuación se presentarán las topologías de red y los protocolos de comunicación más utilizados en este tipo de redes.

En los siguientes apartados se presentarán las tecnologías hardware asociadas: Motes comerciales, transceptores comerciales y microcontroladores comerciales más conocidos.

Para finalizar se analizarán los sistemas operativos y pilas de protocolos más utilizados en redes de sensores así como se realizará una prueba del funcionamiento del sistema operativo Contiki haciendo uso de la pila de protocolos uIPv6.

2.1 Introducción a las Redes de sensores inalámbricas (WSN).

Una red de sensores inalámbrica (WSN) es una red inalámbrica que consta de dispositivos autónomos distribuidos espacialmente utilizando sensores para supervisar conjuntamente elementos físicos o condiciones ambientales. Una definición muy acertada es la siguiente: "*una red inalámbrica de sensores es conjunto de dispositivos inalámbricos o motes que cooperan entre sí para llevar a cabo un objetivo común*" tales como la medición de la temperatura, sonido, vibración, presión, movimiento o detección de elementos contaminantes, en distintos lugares. El desarrollo de las redes inalámbricas de sensores fue originalmente motivado por las aplicaciones militares, como la vigilancia del campo de batalla. Sin embargo, las redes inalámbricas de sensores en la actualidad se utilizan en muchas áreas de aplicación civil, incluidos el control del tráfico, atención sanitaria y domótica, agricultura, etc. El eficiente diseño e implementación de las redes inalámbricas de sensores se ha convertido en un área emergente de investigación en los últimos años, debido al gran potencial de la red, que permite que la aplicación de sensores conecte el mundo físico al mundo virtual.

Por otro lado, el uso de sensores cableados obligaba al despliegue de grandes cantidades de cable e impedía la toma de medidas en lugares de difícil acceso. Con los avances realizados en los campos de la electrónica y las comunicaciones inalámbricas se pudieron solventar estos problemas, abaratando el uso de dispositivos inalámbricos para adquisición de datos, lo que ha permitido el auge de las redes de sensores inalámbricas. Esta tecnología, reconocida en un informe del MIT como una de las diez tecnologías emergentes que cambiarán el mundo [1], constituye una elección muy oportuna para los casos en los que se deben recoger datos de múltiples ubicaciones. Su uso permite el despliegue de un gran número de dispositivos sensores de bajo coste que forman una red inalámbrica robusta, escalable y adaptable a los cambios en el entorno o en su topología.

Además de uno o más sensores, cada nodo en una red de sensores, llamado mote, está equipado normalmente con un transmisor de radio u otro dispositivo de comunicaciones inalámbricas, un pequeño microcontrolador y una fuente de alimentación, que suele ser una batería. Mote es el nombre dado por los científicos de la universidad de Berkeley, en la participación en un proyecto, conocido como Smart Dust [2]. El objetivo del proyecto Smart Dust es reducir el tamaño del dispositivo hasta el tamaño de una mota polvo. El tamaño del sensor de un solo nodo puede variar al igual que el coste de los nodos de sensores, que van desde cientos de dólares a unos pocos centavos, dependiendo del tamaño de la red y de la complejidad necesaria de los distintos nodos de sensores. El tamaño y la limitación del gasto en los nodos de sensores son los causantes de las limitaciones de recursos como la energía, la memoria, la velocidad de cálculo y el ancho de

banda. Las limitaciones de WSN pueden discutirse en términos de potencia, que se considera como el factor decisivo en el despliegue del sensor. Cuando la WSN se coloca en ciertas zonas, como en el bosque o bajo el agua, no es posible mantenerlas bajo supervisión continua, por lo tanto, deben tener la suficiente energía para maximizar la vida de vigilia.

Las limitaciones de las redes inalámbricas de sensores pueden dividirse principalmente en dos categorías, software y hardware. El Software debe incluir algoritmos de baja complejidad y bajo tiempo de ejecución, adaptaciones para cambios en el medio inalámbrico, un eficiente algoritmo de enrutamiento y protocolos de acceso al medio, todos ellos que proporcionen un bajo consumo de energía. El hardware incluye la limitación de la disponibilidad de circuitos integrados o IC, el tamaño del mote, bajo consumo de energía, bajo coste de producción, la capacidad de adaptación al medio ambiente, etc.

Los nodos pueden albergar sensores que midan diferentes variables. Por lo tanto, se pueden tener diferentes datos al mismo tiempo, es decir acústicos, sísmicos, ambientales, etc., transmitidos a través de la red inalámbrica de sensores a la estación base. Estos datos pueden suelen ser almacenados en la estación base, y generalmente, tras un procesamiento de los datos se actúa en consecuencia para conseguir el objetivo deseado. Por ejemplo tras el procesamiento de los datos de una plantación agrícola se puede llegar a la conclusión de que en verano es conveniente regar en intervalos de 2 horas todas las noches a partir de las 2:00 para evitar la evaporación del agua por los rayos solares.

Finalmente podemos decir que los sensores inalámbricos son un paso de gigante hacia la computación proactiva, un paradigma donde las computadoras se anticipan a las necesidades humanas y, en caso necesario, actuar en su nombre.

2.2 Aplicaciones de las redes inalámbricas de sensores.

2.2.1 Aplicaciones agrícolas

La agricultura constituye una de las áreas donde se prevé que pueda implantarse con mayor rapidez este tipo de tecnología. Por ejemplo, las redes de sensores favorecen una reducción en el consumo de agua y pesticidas, contribuyendo a la preservación del entorno. Adicionalmente, pueden alertar sobre la llegada de heladas, así como ayudar en el trabajo de las cosechadoras. Gracias a los desarrollos que se han producido en las redes de sensores inalámbrica en los últimos años, especialmente la miniaturización de los

dispositivos, han surgido nuevas tendencias en el sector agrícola como la llamada agricultura de precisión. Esta disciplina cubre múltiples prácticas relativas a la gestión de cultivos y cosechas, árboles, flores y plantas, etc. Entre las aplicaciones más interesantes se encuentra el control de plagas y enfermedades. Por medio de sensores estratégicamente situados, se pueden monitorizar parámetros tales como el clima, la temperatura o la humedad, con el fin de detectar rápidamente situaciones adversas y desencadenar los tratamientos apropiados. La gran ventaja del uso de esta tecnología es la detección a tiempo y la aplicación óptima de los pesticidas, únicamente en aquellas zonas donde resulta realmente necesario, lo cual unido a la sencillez de despliegue de este tipo de redes, convierte a la agricultura en uno de los campos fundamentales de aplicación de las redes inalámbricas de sensores.

Figura 2-1: Despliegue agrícola de sensores

La Figura 2-1 muestra un despliegue agrícola de sensores[3] . En la primera imagen se puede apreciar un dispositivo final (End- device) que lleva conectado un sensor. La segunda imagen muestra el detalle del hardware del dispositivo final. La tercera ilustración muestra la estación base, se puede apreciar el panel solar necesario para mantener la alimentación de ésta de forma continuada, y la antena en lo alto del mástil. Por último, la cuarta imagen muestra un sensor capaz de detectar la humedad del suelo denominado Watermark ®.

2.2.2 Aplicaciones civiles y militares

IrisNet es una arquitectura software desarrollada por Intel para la gestión de redes mundiales de sensores de diversos tipos, incluyendo vídeo, permitiendo el acceso distribuido a dichos sensores de una forma potente y eficiente. Cuando el número de dispositivos crece de forma significativa, como es el caso de telarañas mundiales de sensores, resulta clave disponer de herramientas eficientes para el acceso a los mismos, en especial por el elevado consumo de ancho de banda. En el proyecto IrisNet se han

demostrado distintas aplicaciones prácticas que hacen uso de dicha herramienta. Entre ellas se incluyen las siguientes:

- Vigilancia de niños y personas mayores mediante videocámaras.
- Seguridad del hogar.
- Avisador de riesgo de epidemias (gripe, fiebres, etc.).
- Monitorización de redes de ordenadores.
- Observatorios terrestres y marítimos (costas).

Además de las propias aplicaciones civiles, las redes de sensores inalámbricos encuentran un importante campo de aplicación en misiones militares. Por ejemplo, en la identificación y seguimiento de tropas o vehículos militares, así como en la detección de armas químicas, biológicas y radiológicas (sustancias NBQ), como el mostrado en la Figura 2-2.

Figura 2-2:Nodo Usado en una red de monitorización de sustancias NBQ[4].

2.2.3 Aplicaciones en el medio ambiente

El mantenimiento y cuidado de espacios y parques naturales resulta complejo en gran medida por las especiales características de los mismos. Se trata de áreas de grandes dimensiones, en algunos casos de difícil acceso, que están repletos de especies vegetales y animales que hay que preservar, por lo que la supervisión de los mismos debe realizarse empleando métodos lo menos intrusivos posibles. Nuevamente las redes inalámbricas de sensores pueden resultar de gran ayuda en este tipo de tareas. Los sensores, de pequeño tamaño, pueden disimularse con en el entorno, procesando los datos de diversos parámetros ecológicos y transmitiendo la información de forma inalámbrica hasta un centro de control, situado normalmente en la caseta de los guardas forestales. De este modo, se evita en la medida de lo posible la circulación de personas y vehículos por el parque. Entre los parámetros a monitorizar podemos enumerar: temperatura, humedad, crecimiento de árboles y arbustos, desplazamientos de especies, conteo de animales, caudales de ríos, etc.

2.2.4 Aplicaciones sociales y sanitarias

El cuidado de personas mayores requiere en la mayor parte de los casos de un seguimiento exhaustivo de sus actividades, lo cual limita su privacidad y al mismo tiempo supone una excesiva carga de trabajo para los cuidadores. Mediante el uso de una red de sensores inalámbricos situados en puntos estratégicos del domicilio del anciano, así como en objetos de uso cotidiano, los cuidadores pueden monitorizar en tiempo real el comportamiento de las personas mayores, evitando la realización de tareas tediosas y centrándose en aspectos más importantes como es la mejora de su calidad de vida. Los sensores pueden integrarse en muebles y aparatos. Se comunican entre ellos y con servidores de la habitación, que a su vez se comunican con otros servidores de la habitación. Todos ellos se integran y organizan con los dispositivos integrados existentes para autoorganizarse, autorregularse y autoadaptarse basándose en modelos de control. De esta forma se puede crear un entorno inteligente.

Adicionalmente, el cuidado médico tanto en hospitales como fuera de los mismos, por ejemplo la rehabilitación de pacientes, también se beneficia del uso de esta tecnología. Un ejemplo de ello es el proyecto CodeBlue [5], desarrollado en la Universidad de Harvard. En este caso se han implementado distintos tipos de sensores para la monitorización de parámetros vitales: tasa de latidos del corazón, concentración de oxígeno en sangre, datos EKG de electrocardiograma, etc. Toda esta información se recoge por los sensores y se distribuye de forma inalámbrica a una PDA u ordenador portátil para su procesamiento. De este modo, cualquier señal de alerta puede detectarse a distancia en tiempo real.

Por otro lado, el hardware electrónico actual, ha llegado a un nivel de miniaturización (ver Figura 2-3), tal que los dispositivos electrónicos de hoy en día son capaces de adaptarse al cuerpo humano. Entre la ciencia ficción y la realidad, la idea es crear redes de sensores inalámbricas que permitan un control del estado de salud de las personas y sirvan para prevenir problemas y enfermedades. Con esta ayuda tecnológica, el hombre podría anticiparse a muchas patologías como los ataques al corazón.

Figura 2-3: Chips de reducido tamaño

2.2.5 Otras aplicaciones comerciales.

- **Control ambiental en edificios de oficinas.**

  Normalmente la calefacción o el aire acondicionado se controlan desde una central, por lo que la temperatura dentro de cada habitación puede variar unos pocos grados debido a que la distribución de aire no está uniformemente distribuida y el control es central. Se podría desplegar una WSN para controlar el flujo de aire y la temperatura en diferentes partes de la habitación.

- **Gestión de inventarios.**

  En un almacén, cada artículo puede llevar adherido un sensor, de modo que se puede conocer en todo momento su localización y número de artículos por categoría. De esta forma, para dar de alta nuevos artículos se les añadiría el sensor y se llevarían al almacén.

2.3 Elementos de una red de sensores inalámbrica

En relación con los elementos que componen una WSN (ver Figura 2-4), caben destacar (1) los nodos sensores, (2) el nodo sumidero, y (3) la red inalámbrica en su conjunto. A su vez, los nodos sensores están compuestos por un dispositivo de cómputo como un microcontrolador o un microprocesador, los sensores y/o actuadores correspondientes, elementos de almacenamiento temporal y permanente, un sistema de alimentación y un transceptor de radio. En cuanto a sus características, son dispositivos que procesan algoritmos sencillos, ya que requieren de una elevada autonomía. Por otro lado, el nodo sumidero hace de puente entre la red inalámbrica y el equipo que hace de servidor. Finalmente, hay que mencionar la red inalámbrica propiamente dicha está basada normalmente en un estándar de comunicaciones como el IEEE 802.15.4 [6] o ZigBee [7].

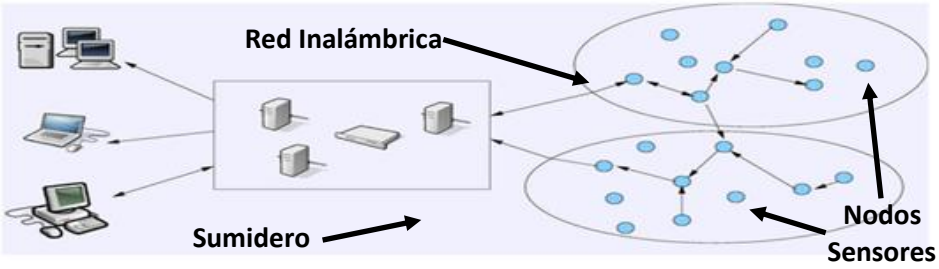


Figura 2-4: Elementos de una red inalámbrica de sensores.

2.4 Características de una red de sensores inalámbrica

Las WSN tienen una serie de características que hay que tener en cuenta en la fase de diseño, a pesar de que la naturaleza de los nodos que forman las redes pueda ser muy distinta y la función que realizarán variará dependiendo del tipo de aplicación. Pero todas ellas tendrán una serie de características comunes:

1. ***Gran escala*** **:** La cantidad de nodos que se despliega en una red puede crecer a lo largo de la vida de ésta, pudiendo llegar a contener miles de nodos. La red estará compuesta de muchos sensores densamente desplegados en el lugar donde se desee llevar a cabo la labor de monitorización.

2. ***Topología variable*** **:** La posición de cada uno de los nodos estará determinada de forma arbitraria o estratégica y normalmente es desconocida por el resto de nodos. La localización no tiene porqué estar diseñada ni preestablecida, lo que va a permitir un despliegue aleatorio en terrenos inaccesibles u operaciones de alivio en desastres.

3. ***Recursos limitados:*** Los sensores, a cambio de su bajo consumo de energía, coste y pequeño tamaño disponen de recursos hardware muy limitados.

4. ***El consumo energético*** **:** En otras redes ad-hoc y móviles, el consumo energético ha sido un factor importante de diseño, pero no el más ponderado, simplemente porque los elementos pueden ser reemplazados por el usuario. Sin embargo, en las WSN el consumo energético es un factor muy importante, directamente relacionado con el tiempo de vida de la red. De modo que como las WSN se caracterizan por una baja tasa de transmisión de datos, los nodos sensores mantienen sus transceptores de radio en modo de bajo consumo la mayor parte del tiempo

5. ***Costes de producción:*** Una WSN consiste en un elevado número de nodos y por ello, es importante que el coste individual de cada nodo no sea muy elevado. Si el coste de la WSN es mayor que el necesario para desplegar soluciones tradicionales, el uso de una WSN pueda no estar justificado.

2.5 Requisitos de una Red inalámbrica de sensores

Para que una red pueda funcionar de acuerdo con las características descritas en el apartado anterior surgen una serie de requisitos no funcionales que una aplicación debe cumplir.

1. ***Eficiencia energética***: Es una de las preocupaciones más importantes en redes de sensores. Cuanto menor sea el consumo de un nodo mayor será el tiempo durante el cual pueda operar y, por tanto, mayor tiempo de vida tendrá la red. La aplicación tiene la capacidad de bajar este consumo de potencia restringiendo el uso de la CPU y la radio FM. Esto se consigue desactivándolos cuando no se utilizan y, sobre todo, disminuyendo el número de mensajes que generan y retransmiten los nodos.

2. ***Autoorganización***: Los nodos desplegados deben formar una topología que permita establecer rutas por las que mandar los datos que han obtenido. Los nodos necesitan conocer su lugar en esta topología, pero resulta inviable asignarlo manualmente a cada uno, debido al gran número de nodos. Por ello, es fundamental que sean capaces de formar la topología deseada sin ayuda del exterior de la red. Este proceso no sólo debe ejecutarse cuando la red comienza su funcionamiento, sino que debe permitir que en cada momento la red se adapte a los cambios que pueda haber en ella.

3. ***Escalabilidad***: Puesto que las aplicaciones van creciendo con el tiempo y el despliegue de la red es progresivo, es necesario que la solución elegida para la red permita su crecimiento sin que las prestaciones de la red caigan drásticamente.

4. ***Tolerancia a fallos***: Los sensores son dispositivos propensos a fallar. Los fallos pueden ser debidos al estado de las baterías, a un error de programación, a condiciones ambientales, al estado de la red, etc. Una de las

razones de esta probabilidad de fallo radica precisamente en el bajo coste de los sensores. En cualquier caso, se deben minimizar las consecuencias de ese fallo evitando que un fallo en un nodo individual provoque el mal funcionamiento del conjunto de la red.

5. ***Tiempo real***: Los datos llegan a su destino con cierto retraso. Pero algunos datos deben entregarse dentro de un intervalo de tiempo conocido. Pasado éste dejan de ser válidos, como puede pasar con datos que impliquen una reacción inmediata del sistema, o se pueden originar problemas serios. En caso de que una aplicación tenga estas restricciones debe tomar las medidas que garanticen la llegada a tiempo de los datos.

6. ***Seguridad***: Las comunicaciones inalámbricas viajan por un medio fácilmente accesible a personas ajenas a la red de sensores. Esto implica un riesgo potencial para los datos recolectados y para el funcionamiento de la red. Se deben establecer mecanismos que permitan tanto proteger los datos de estos intrusos, como protegerse de los datos que estos puedan inyectar en la red.

Según la aplicación que se diseñe algunos de los anteriores requisitos cobran mayor importancia. Es necesario encontrar el peso que cada uno de ellos tiene en el diseño de la red, pues normalmente unos requisitos van en detrimento de otros.

Por ejemplo, dotar a una red de propiedades de tiempo real podría implicar aumentar la frecuencia con la que se mandan mensajes con datos, lo cual repercutiría en un mayor consumo de potencia y un menor tiempo de vida de los nodo. Esto lleva a buscar, para cada aplicación, un compromiso entre los requisitos anteriores que permita lograr un funcionamiento de la red adecuado para la misión que debe llevar a cabo.

2.6 Topologías de red y protocolos de comunicación.

Una WSN debe funcionar de manera autónoma aunque ocurran situaciones como el fallo de unos o varios nodos sensores o la adición de nuevos nodos en la red. Además, en determinadas aplicaciones los motes se pueden encontrar desplegados en lugares de difícil acceso. Por ello, los nodos tienen que funcionar de manera autónoma y tener la máxima autonomía posible, que se traducirá en un tiempo de vida mayor en la WSN. En esta línea es necesario utilizar topologías de red, así como estándares y protocolos de comunicación que garanticen lo anteriormente expuesto.

2.6.1 El estándar IEEE 802.15.4.

El comité de nuevos estándares del IEEE (NesCom, IEEE *New Standards Committee*) estableció un grupo de trabajo en diciembre del año 2000, para comenzar el desarrollo de un estándar con baja tasa de transmisión de datos en redes inalámbricas de área personal. El grupo de trabajo 4 (TG4, *Task Group 4*) del IEEE 802.15 se encargaba de investigar una solución con baja velocidad de datos, con una duración de la batería de varios meses o años y de baja complejidad. A su vez la solución debía operar en una banda de frecuencia internacional y sin licencia. Las aplicaciones potenciales de comunicación del estándar serían los sensores, juguetes interactivos, tarjetas inteligentes, mandos a distancia, y en domótica. El TG4 se puso en estado de hibernación en la reunión de marzo 2004 después de formar un nuevo grupo de trabajo (TG4b). El nuevo TG4b completó su trabajo con la publicación de la revisión 2006 del estándar IEEE 802.15.4, que es compatible con la del 2003. Así, nuevos dispositivos 2006 pueden operar en una red 2003.

El estándar IEEE 802.15.4 define la capa física (PHY, *Physical*) y la especificación de la subcapa del control de acceso al medio (MAC, *Medium Access Control*) para dispositivos fijos, portátiles o móviles sin batería o con requerimientos de consumo para baterías limitadas que operen en redes inalámbricas personales de área local con tasas bajas de envío de datos (LR-WPAN, *Low Rate-Wireless Personal Area Network*). Las WPANs son usadas para intercambiar información sobre distancias relativamente cortas. A diferencia de las redes inalámbricas de área local (WLAN, *Wireless Local Area Network*), las conexiones realizadas a través de una WPAN requieren de infraestructuras muy sencillas e incluso de ninguna infraestructura. Esta característica permite implementar soluciones de pequeño tamaño, eficientes energéticamente y económicas para una amplia gama de dispositivos.

La capa física proporciona una interfaz entre la subcapa MAC y el canal de radio, a través del firmware y el hardware de RF. Conceptualmente la PHY (ver Figura 2-5) incluye una entidad de gestión denominado PLME (P*hysical Layer Management Entity*). Esta entidad proporciona las interfaces de administración de servicios de la capa a través del cual las funciones de gestión de la capa pueden ser invocadas. La PLME también es responsable de mantener una base de datos de los objetos gestionados relacionados con la PHY. Esta base de datos se denomina la base de información PHY PAN (PIB, *PAN Information Base*). La PHY proporciona dos servicios: el servicio de datos PHY y el servicio de interfaz de gestión de la PHY, que está ligado con un servicio de acceso puntual (SAP, *Sevice Access Point*) de la entidad de gestión de la capa física (PLME, *Physical Layer Management Entity*), conocido por la siglas PLME-SAP. El servicio de datos de la PHY permite realizar transmisiones y recepciones de unidades de datos del protocolo PHY (MPDU, MAC Protocol Data Unit) a través del canal físico de radio. El PLME-SAP permite el transporte de comandos de gestión entre MLME (*MAC Sublayer Management Entity*) y la PLME. En definitiva, la PHY define las características físicas y funciones del enlace inalámbrico, entre las que se pueden citar las siguientes: bandas de frecuencia y número de canales de cada banda, la potencia de transmisión, la posibilidad de habilitar y deshabilitar el módulo de

radio y la selección del canal. En relación con las bandas de frecuencia, el estándar define 16 canales en la banda disponible a nivel mundial de 2450MHz, 30 canales en la banda de 915MHz disponible en Norte América y 3 en la banda disponible en Europa de 868MHz.

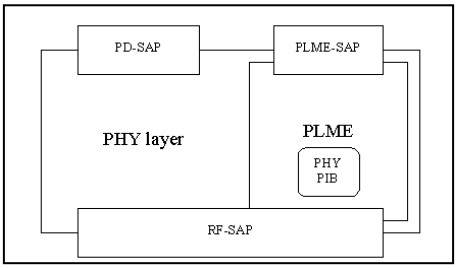


Figura 2-5:Modelo de referencia de la PHY [6].

La subcapa MAC (ver Figura 2-6) proporciona dos servicios: el servicio de datos MAC y el servicio de gestión de la MAC ligado con el servicio de acceso puntual de la entidad de gestión de la subcapa MAC (MLME-SAP, *MAC Sublayer Management Entity-Service Access Point*). El servicio de datos MAC permite la transmisión y recepción de MPDUs a través del servicio de datos PHY. Esta subcapa, tiene entre otras, las siguientes características: gestión de las balizas (beacons), acceso al canal, asociación, diasociación. Además, la subcapa MAC proporciona servicios para la aplicación de mecanismos de seguridad adecuados en las transmisiones de la aplicación. Los *beacons* son tramas especiales de sincronización, que si están habilitadas, el coordinador de la red envía periódicamente a los nodos sensores con el fin de tener la red sincronizada y evitar colisiones en las transmisiones.

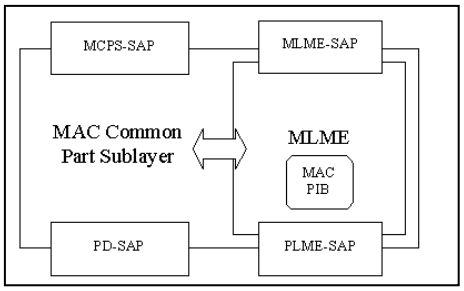


Figura 2-6:Modelo de referencia de la subcapa MAC [6].

En relación con los dispositivos, hay dos tipos diferentes que pueden participar en una red basada el estándar IEEE 802.15.4; los dispositivos de funcionalidad completa (FFD, *Full Function Device*) y los de funcionalidad reducida (RFD, *Reduced Function Device*). El FFD puede operar en tres modos siendo el coordinador PAN (Personal Area Network), un coordinador o un dispositivo. Además, un FFD puede comunicarse con RFDs y otros FFDs, mientras que un RFD sólo puede establecer una comunicación con un dispositivo con el rol de FFD. Un RFD está pensado para aplicaciones que son extremadamente simples, como el interruptor de una luz o un sensor pasivo de infrarrojos; no necesitan envíar una gran cantidad de paquetes y sólo pueden estar asociados con un único FFD en un instante determinado. En consecuencia, el RFD puede ser implementado usando unos mínimos recursos y una memoria de reducida capacidad.

En función de los requerimientos de la aplicación, una IEEE 802.15.4 LR-WPAN puede estar configurada con dos topologías (ver Figura 2-7): la topología en estrella o *star* y entre iguales o *peer-to-peer*. Con la topología en estrella la comunicación se establece entre los dispositivos y un único controlador central, denominado como el coordinador PAN. Típicamente un dispositivo tiene una aplicación asociada y puede ser cualquiera, el punto de inicio o de fin de la red de comunicación. Un coordinador PAN puede tener una aplicación específica, pero tiene la obligación de iniciar, liberar, o intercambiar los paquetes dentro de la red. El coordinador PAN es el coordinador principal de la PAN. Todos los dispositivos que operan en una red con cualquier topología deben tener direcciones únicas de 64-bits. Esta dirección se puede para realizar comunicaciones directas dentro de la PAN. También se pueden usar direcciones de 16 bits, que son asignadas a los dispositivos cuando se asocian con el coordinador y que se denotan con el término '*short address*'. En relación con los requerimientos energéticos, el coordinador PAN tiene que estar siempre activo y generalmente no estará alimentado por baterías, salvo que haya un sistema de energía alternativa o no, que la recargue. Al contrario, el resto de dispositivos de la PAN no tienen elevados consumos energéticos y generalmente estarán alimentados por baterías. Entre las aplicaciones que se pueden beneficiar de una topología en estrella (ver Figura 2-7), caben destacar las siguientes: automatización del hogar y la domótica, periféricos de un computador, juguetes y juegos, y la monitorización de la salud de pacientes.

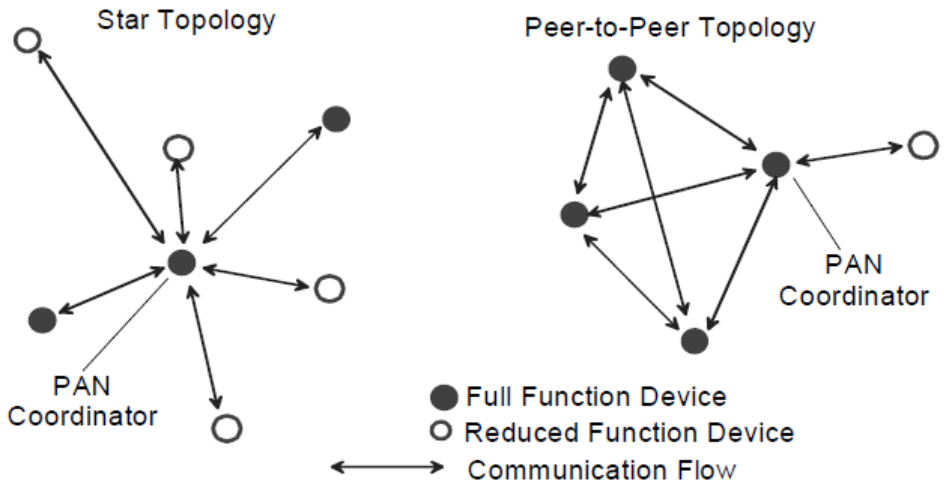


Figura 2-7: Ejemplos de topologías en estrella y peer-to-peer [6].

La topología *peer-to-peer* también tiene un coordinador PAN; sin embargo, se diferencia de la topología en estrella en que cualquier dispositivo puede intercambiar información con otro dispositivo, siempre y cuando esté dentro de su rango de alcance. Además, esta topología permite implementar redes más complejas, como una topología de tipo *mesh*. En este caso las aplicaciones que se pueden beneficiar de esta topología son las siguientes: control industrial y monitorización, WSNs, seguimiento de objetos, agricultura de precisión y la seguridad. Una red *peer-to-peer* puede ser ad hoc, auto-organizada y con capacidad de auto-reparación. También puede permitir intercambiar mensajes entre dos dispositivos a través de múltiples saltos (*hops*) intermedios en otros nodos. Estas funciones

pueden ser añadidas en capas superiores, pero no forman parte del estándar. Estas funciones han sido añadidas en el protocolo de comunicación de alto nivel ZigBee [8].

2.6.2 ZigBee.

La ZigBee Alliance es una asociación de compañías trabajando en conjunto para desarrollar estándares (y productos) para redes inalámbricas de bajo consumo y coste y que sean también seguras. Probablemente, a lo largo de los próximos años la tecnología ZigBee será embebida en una gran cantidad de productos y aplicaciones a nivel mundial, como en el consumo, de tipo comercial, industrial, entre otros [7].

ZigBee define protocolos de comunicación de alto nivel construidos sobre la subcapa MAC del estándar IEEE 802.15.4 para LR-WPANs. ZigBee es simple, de bajo coste, y una tecnología inalámbrica de bajo consumo usada en aplicaciones empotradas.

Los dispositivos ZigBee pueden formar redes malladas conectando entre ellos desde cientos hasta miles de nodos inalámbricos. Los dispositivos no requieren elevados consumos energéticos y pueden funcionar usando baterías durante varios años. En general ZigBee define la especificación de la capa de red para topologías en estrella, árbol y *peer-to-peer*, y proporciona un marco de trabajo para la programación de las aplicaciones.

La versión 2004 del estándar ZigBee fue publicada en junio del año 2005. Esta versión fue revisada con la 2006, que fue publicada en diciembre de 2006. Y durante el último trimestre de 2007 finalizó la publicación de la versión 2007, que se puede encontrar referenciada como ZigBee Pro [8].

ZigBee estandariza las capas superiores (ver Figura 2-8) de la pila del protocolo. La capa de red (NWK, *Network*) está encargada de organizar y proporcionar rutado sobre una red de tipo *multihop*, al contrario que la capa de aplicación (APL, *Application Layer*) proporciona un marco de trabajo para el desarrollo de aplicaciones y comunicaciones distribuidas. La APL engloba el *application framework*, el *ZigBee Device Object* y la subcapa de aplicación (APS, *Application Sub Layer*). El *application framework* puede tener hasta 240 objetos de aplicación, esto es, módulos de aplicación definidos por el usuario que son parte de la aplicación ZigBee. El ZDO proporciona servicios que permiten descubrir unos APOs (*Application Objects*) de los otros, así como en organizarse en aplicaciones distribuidas. La APS ofrece una interfaz de datos y servicios de seguridad entre los APOs y el ZDO.

La capa de red (ver Figura 2-8) es necesaria para proporcionar la funcionalidad que asegure una correcta operación de la subcapa MAC del IEEE 802.15.4 y para proporcionar un conjunto de servicios de interfaz con la capa de aplicación.

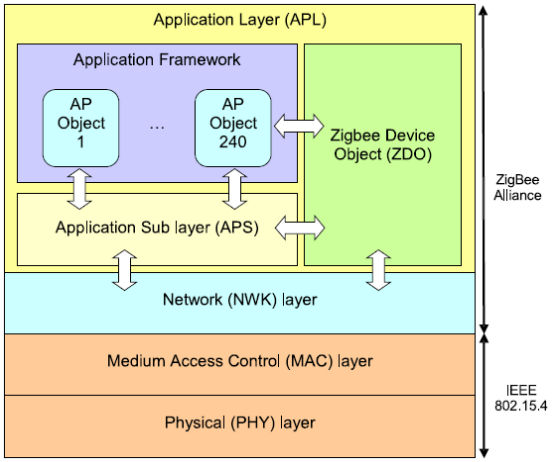


Figura 2-8: Arquitectura de capas funcional y la pila del protocolo ZigBee [7].

Para intercambiar información con la capa de aplicación, conceptualmente la capa de red incluye dos entidades de servicio que proporcionan la funcionalidad necesaria. Estas entidades de servicio son el servicio de datos y el servicio de gestión. La entidad de datos de la capa de red (NLDE, *NWK Layer Data Entity*) proporciona los servicios de transmisión de datos a través de su SAP asociado, el NLDE-SAP, y la entidad de gestión de la capa de red (NLME, *NWK Layer Management Entity*) proporciona el servicio de gestión por medio de su SAP asociado, el NLME-SAP. El NLME utiliza el NLDE para conseguir algunas de sus tareas de gestión y también manteniene una base de datos de los objetos manejados conocida como la *network information base* o por la siglas NIB [9].

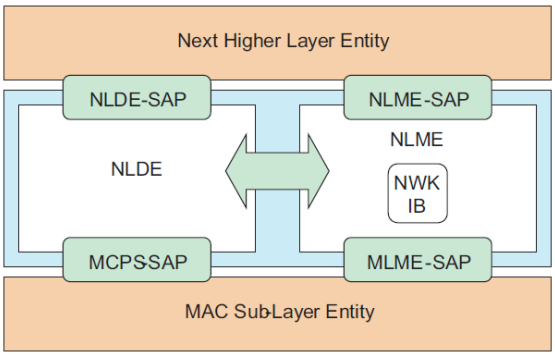


Figura 2-9: Arquitectura de referencia de la capa de red [7].

La capa de aplicación ZigBee está compuesta por la subcapa APS, el ZDO y los objetos aplicación definidos por el desarrollador. Las responsabilidades de la subcapa APS son mantener las tablas de *binding*, que es la habilidad de conectar dos dispositivos entre ellos basándose en sus servicios y sus necesidades, e intercambiar mensajes entre los dispositivos que tienen establecido el *binding*. Otra responsabilidad de la subcapa APS es el descubrimiento (discovery), que es la capacidad de determinar que otros dispositivos están funcionando en el espacio personal de operación del dispositivo. Las obligaciones del ZDO incluyen definir el rol del dispositivo dentro de la red, iniciar y/o responder a peticiones y establecimientos de *binding* y establecer una relación segura entre los dispositivos de la red.

Los objetos aplicación implementados y definidos por el desarrollador implementan la aplicación en relación con las descripciones de la aplicación ZigBee definida.

Entrando más en detalles, la subcapa APS proporciona una interfaz entre la siguiente capa (NHLE, *Next Higher Layer Entity*) y la capa de red. Conceptualmente la subcapa APS incluye una entidad de gestión llamada la entidad de gestión de la subcapa APS (APSME, *APS Sub-layer Management Entity*). Esta entidad proporciona las interfaces de servicio a través de las funciones de gestión que sean invocadas. El APSME es también responsable de mantener una base de datos de los objetos manejados pertenecientes a la subcapa APS. Esta base de datos se denota como la base de información de la subcapa APS (AIB, APS Information Base). La Figura 2-10 muestra las interfaces y los componentes de la subcapa APS.

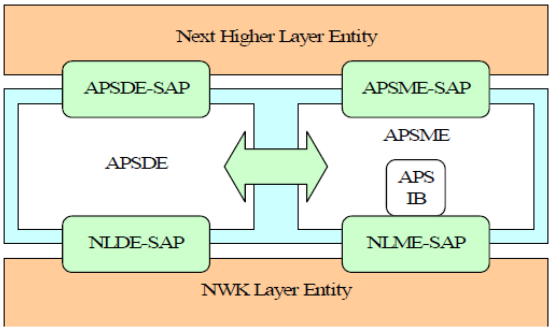


Figura 2-10: Arquitectura de referencia de la capa de la subcapa APS [9].

La subcapa APS proporciona dos servicios, accedidos por medio de dos SAPs. Estos son el servicio de datos APS, accedido a través del SAP de la entidad de datos de la subcapa APS (APSDE-SAP, *APS Sublayer Data Entity SAP*), y el servicio de gestión de la APS, accedido por medio del SAP de la entidad de gestión de la subcapa APS (APSME-SAP, *APS Sub-layer Management Entity SAP*). Estos dos servicios proporcionan la interfaz entre el NHLE y la capa de red, a través del NLDE-SAP y, de forma limitada, con la interfaces NLME-SAP. La interfaz NLME-SAP entre la capa de red y la subcapa APS sólo soporta primitivas NLME-GET y NLME-SET; el resto de las primitivas NLME-SAP están disponibles sólo por medio del ZDO. Además de estas interfaces externas, también hay una interfaz implícita entre el APSME y el APSDE que permite que el primero use el servicio de datos de la APS.

En una red ZigBee hay involucrados tres tipos de dispositivos que define el estándar: el coordinador ZigBee (ZC, *ZigBee Coordinator*), el router ZigBee (ZR, *ZigBee Router*) y el dispositivo final ZigBee (ZED, *ZigBee End Device*). El ZED se corresponde con un RFD o un FFD definido por el estándar IEEE 802.15.4 que actúa como un dispositivo simple. Un ZR es un FFD con capacidad de realizar tareas de enrutado en la red. El ZC es un FFD que maneja la red completa y sólo puede haber uno en cada red.

Algunas de las funcionalidades de la capa de red son: enrutado *multihop*, descubrimiento y mantenimiento de rutas, seguridad y conexión/desconexión de la red, con la consecuente asignación de una *short address* (16-bits) a los nuevos dispositivos

conectados. Además, la capa de red soporta topologías más complejas que las del estándar IEEE 802.15.4. Estas topologías son las siguientes (ver Figura 2-11): estrella (*star*), árbol (*tree*) y malladas (*mesh*). En una topología en estrella, la red está controlada por un único dispositivo, el ZC. El ZC es el responsable de iniciar y mantener los dispositivos en la red. El resto de dispositivos, de tipo ZED, se comunican directamente con el ZC. En las topologías en árbol y malladas, el ZC tiene la responsabilidad de iniciar y configurar ciertos parámetros importantes de la red, pero la red puede extenderse haciendo uso de los ZRs. En redes en árbol, los ZRs intercambian datos y mensajes de control a lo largo de la red usando una estrategia de rutado jerárquica. Además, este tipo de redes pueden emplear comunicaciones con los *beacon* habilitados como describe el estándar IEEE. Las redes malladas permiten una comunicación *peer-to-peer* completa. Para ofrecer este soporte, se utiliza con ligeras modificacaciones el protocolo de rutado AODV (Ad-hoc On-demand Distance Vector) [10]. Finalmente, cabe destacar que la revision actual de ZigBee sólo describe redes intra-PAN, esto es, redes en las que las comunicaciones empiezan y terminar dentro de la misma red.

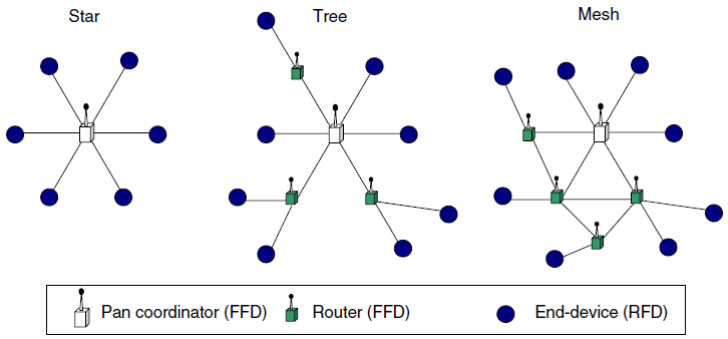


Figura 2-11: Topologías de las redes ZigBee [7].

2.6.3 Otros estándares.

2.6.3.1 WirelessHART.

El estándar WirelessHART [11; 12] proporciona un protocolo de comunicación inalámbrica para aplicaciones de control de procesos e instrumentación. El estándar está basado en el IEEE 802.15.4 con una operación de bajo consumo en la banda de 2,4GHz. El WirelessHART es compatible con todos los productos existentes, herramientas y sistemas. Además, es fiable, seguro y eficiente en términos energéticos. Soporta redes malladas, salto de frecuencia y mensajes sincronizados. La red de comunicación es segura gracias a la encriptación, verificación, autentificación y a la gestión de la clave de la red. Las opciones de gestión energética permiten que los dispositivos inalámbricos sean más eficientes energéticamente. WirelessHART está diseñado para soportar topologías malladas, en estrella, así como combinaciones de las anteriores. Una red WirelessHART está compuesta de los dispositivos inalámbricos situados en campo, gateways, controladores de

la automatización del procreso, las aplicaciones host y el gestor de la red (ver Figura 2-12). Los dispositivos inalámbricos de campo están conectados con el proceso o los equipos de la planta. Los gateways garantizan la comunicación entre los dispositivos inalámbricos de campo y las aplicaciones host. El controlador de la automatización del proceso actúa como un sencillo controlador para el proceso. El gestor de la red configura la red y organiza la comunicación entre los dispositivos. Además, se encargar del rutado y del tráfico de la red. El gestor de la red se puede integrar en un Gateway, en una aplicación host, o en un controlador del proceso. El estándar WirelessHART estuvo disponible para la industria en septiembre del 2007 y pronto estará disponible en productos comerciales.

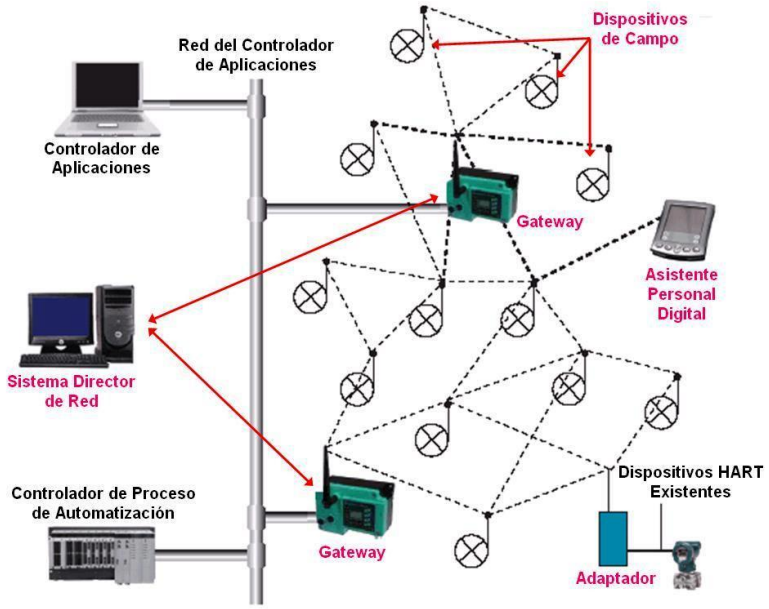


Figura 2-12: Red WirelessHART y sus elementos.

2.6.3.2 ISA100.11a

El estándar ISA100.11a [13] está diseñado para monitorización inalámbrica y control de automatización de procesos con redes inalámbricas con baja velocidad de datos. Define la especificación para la capa OSI, seguridad y el sistema de gestión. El estándar tiene como objetivos un bajo consumo, la escalabilidad de la red, una infraestructura robusta y la interoperabilidad con otros dispositivos inalámbricos. La red sólo usa la banda de frecuencia de 2,4GHz y saltos de frecuencia para incrementar la fiabilidad de la red y minimizar las interferencias. En cuanto a las topologías, ofrece topologías malladas y en estrella. Además, el ISA100.11a proporciona una funcionalidad de seguridad simple, flexible y escalable.

2.6.3.3 6LoWPAN

El estándar 6LoWPAN (IPv6-based Low power Wireless Personal Area Networks) [14;15;16] proporciona el envío de paquetes IPv6 sobre una red basada en el IEEE 802.15.4. Los dispositivos de bajo consume pueden comunicarse directamente con dispositivos IP usando protocolos IP. Usando 6LoWPAN, dispositivos de bajo consumo tienen todos los beneficios de una comunicación y una gestión IP. El estándar 6LoWPAN proporciona una capa de adaptación, un nuevo formato de paquetes, así como gestión del direccionamiento. La capa de adaptación es usada porque el tamaño de los paquetes IPv6 son mucho mayores que el tamaño de trama del IEEE 802.15.4. La capa de adaptación lleva a cabo la funcionalidad para la compresión de la cabecera. Usando la compresión de la cabecera, se crean pequeños paquetes que encajan con el tamaño de trama del IEEE 802.15.4. El mecanismo de gestión de la dirección se encarga de la gestión de las direcciones de los dispositivos para garantizar la comunicación entre ellos. En definitiva, el estándar 6LoWPAN está pensado para aplicaciones en las que usan dispositivos que tienen una baja tasa de envío de datos y que requieren comunicación con Internet.

2.6.3.4 IEEE 802.15.3

El estándar IEEE 802.15.3 [17] define una capa física y una capa de acceso al medio para WPANs con una elevada tasa de datos. Está diseñado para soportar transmisiones de audio y vídeo en tiempo real. El estándar define como la banda de operación la de 2,4GHz y tasas de envíos de datos desde 11 hasta 55Mbps. Para garantizar la calidad del servicio el estándar usa TDMA (*Time Division Multiple Access*). Soporta transferencias de datos síncronas y asíncronas y lleva a cabo las gestiones del consumo de potencia, la escalabilidad de la tasa de datos enviada y la frecuencia. El estándar se usa en dispositivos, como auriculares inalámbricos, equipos de vídeo portátil y conectividad inalámbrica para juegos, teléfonos inalámbricos, impresoras y televisores.

2.6.3.5 Wibree

Wibree [18] es una tecnología de comunicación diseñada para dispostivos de bajo consumo, comunicaciones dentro de pequeños alcances y bajo coste. Wibree permite la comunicación entre dispositivos provistos de pequeñas baterías y dispositivos Bluetooth. Entre los dispositivos provistos de pequeñas baterías caben mencionar los siguientes: relojes, teclados inalámbricos y los sensores de deporte (ver Figura 2-13) que están conectados con ordenadores personales o teléfonos móviles.

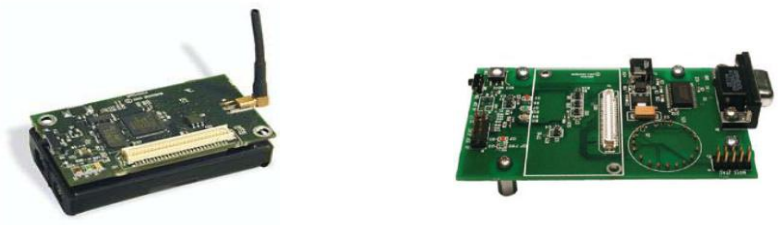
Figura 2-13: Red Wibree de sensores de deporte.

Wibree trabaja en la banda de 2,4GHz, soporta una tasa de envío de datos de 1Mbps y la distancia de los enlaces puede estar comprendida entre 5 y 10m. Wibree está diseñado para trabajar con Bluetooth. La combinación de Bluetooth y Wibree consigue dispositivos más pequeños y más eficientes a nivel energético. Wibree se dio a conocer en octubre del 2006.

2.7 Motes Comerciales

2.7.1 Micaz

La plataforma de sensores MICAz (ver Figura 2-15) es comercializada por Crossbow [19].Estos dispositivos, trabajan en la banda de frecuencias de 2400 MHz a 2483.5 MHz. La familia MICAz usa el Chipcon CC2420 (ver apartado 2.9.1), que cumple con la normativa IEEE 802.15.4 y tiene un transceptor de radio frecuencia Zigbee integrado con un micro controlador Atmega 128L. Dispone de un conector de expansión de 51 pines así como de 512 KBytes memoria flash no volátil (para el almacenamiento de datos) y 128 KBytes de memoria Flash para llevar a cabo la programación del dispositivo.



a) **b)**

Figura 2-14: (a) Micaz Mote. (b) Placa de desarrollo MIB510.

Para la programación de estos motes se emplea la Placa de desarrollo *MIB510* (Figura 15). Esta placa actúa como interfaz entre el PC y los motes a través del puerto serie, permitiendo la programación de los dispositivos acoplados.

2.7.2 Mica2 y Mica2dot

Diseñados por investigadores de la universidad de Berkeley, en California, existen dos modelos llamados Mica2 y Mica2dot. Su funcionalidad es similar pero son muy diferentes en factor de forma, como se puede apreciar en la Figura 2-16. Funcionan ambos a 4MHz con un microprocesador de 8 bits de la marca Atmel. Tiene 128KB de meoria para el programa y 4KB de memoria RAM. Su velocidad de transmisión radio es de 19,2Kb/s sobre un canal CSMA/CA y utiliza el estándar IEEE 802.15.4. También están equipados con una memoria externa de tipo Flash no volátil con 512KB que puede ser usada para guardar otros datos.

(a) **(b)**

Figura 2-15: (a) Mica2. (b) Mica2dot.

A la placa principal del procesador se le pueden adherir placas con sensores/actuadores por medio de unos pines, en el caso del Mica2, o por medio de unos conectores, para el modelo Mica. Como por ejemplo, una placa con sensores de temperatura, luminosidad, un micrófono acelerómetro y un sensor magnético, entre otros.

2.7.3 Intel Mote 2

Intel Mote (ver Figura 2-17) surge de una colaboración[20] entre los laboratorios de Intel en Berkeley, la Universidad de Berkeleyy otros centros de investigación del resto de USA. Los Motes de Intel son pequeños, autónomos, alimentados por medio de baterías y con comunicación vía radio, de forma que son capaces de compartir información con otros y organizarse automáticamente dentro de una red AdHoc.

Uno de los principales objetivos del grupo de trabajo del Intel Mote es colaborar con la comunidad investigadora en la exploración de las potenciales aplicaciones de los Motes y de las redes de sensores. Con este objetivo en mente, se han diseñado los Motes con una total compatibilidad con el sistema operativo TinyOS.

Las investigaciones de mejora se están centrando en tres aspectos:

- El trabajo con consumos de energía ultra bajos.
- Integración del sistema.
- Reconfiguración hardware.

Intel Mote 2 es una plataforma avanzada para la creación de una red de sensores wireless. La plataforma está construida alrededor del procesador de bajo consumo XScale PXA27x con comunicación radio 802.15.4 sobre la placa principal y una antena de 2.4GHz. Contiene 2 interfaces de sensores básicos en uno de los lados de la placa central y otros 2 interfaces avanzados para sensores en el otro lado.

Figura 2-16: Intel Mote

El Intel Mote 2 es una plataforma modular y escalable. La placa principal contiene el procesador y el módulo radio y se le pueden agregar diferentes módulos sensores dependiendo de la aplicación final de la red de sensores.

El procesador que contiene la placa principal puede funcionar a bajo voltaje (0.85V) y una frecuencia también baja (13MHz), cuando se habilita el modo de bajo consumo, mientras que el valor máximo de operación es de 416MHz. También integra una memoria SRAM de 256KB dividida en 4 bancos de 64KB y diferentes opciones de E/S, lo que lo hace extremadamente flexible para soportar diferentes sensores A/D, opciones de radio. Estas E/S incluye I2C, 3 puertos serie síncronos, 3 puertos UART, E/S digitales, un cliente USB. Además, el propio procesador incluye diversos temporizados y un reloj en tiempo real.

Una de las características más importantes y que lo diferencian del resto de motes comerciales, es que existe una placa de expansión (ver Figura 2-19) que incorpora una cámara CMOS, sensor PIR, micrófono y altavoz capaz de reproducir sonidos con una alto nivel de detalle. Lo cual unido a las 64 MB de memoria disponibles y a su compatibilidad con TinyOS2.x.x y TinyOS 1.1.x (Sólo el altavoz, sensor PIR y micrófono), le permite desarrollar aplicaciones que son inpensables en otros dispositivos comerciales.

El principal problema de este dispositivo, es su elevado precio, alrededor de 1000 € el Imote2 con la placa de expansión, se antoja muy elevado en la actualidad.

Figura 2-17: Placa de expansión para Imote2 (cámara, altavoz, micrófono y sensor PIR).

2.7.4 TelosB

El dispositivo TelosB (ver Figura 2-19), también llamado Tmote Sky consta del microcontrolador MSP430 F1611. Este procesador RISC de 16 bits de bajo consumo está diseñado expresamente para el uso en redes de sensores inalámbricas.

Al igual que el Micaz, consta de un transceptor de radio Chipcon CC2420. Otra de las características del Telosb es que dispone de sensores de humedad, temperatura y luminosidad en la propia placa del dispostivo, lo cual permite realizar medir estas dos varibles sin necesidad de incorporar una placa de expansión.

Figura 2-18: TelosB

Una de las principales cualidades del mote TelosB es que totalmente compatible con los dos sistemas operativos más utilizados en redes de sensores inalámbricas: Contiki y TinyOS. Lo cual le ofrece una gran versatilidad a la hora de elegir cuál de ellos se adapta mejor a las características de la aplicación que se desee desarrollar.

2.7.5 Zolertia Z1

El mote Zolertia[21] (ver Figura 2-20) es un mote comercial con escaso tiempo en el mercado que ha logrado hacerse un hueco en el panorama de las redes de sensores gracias a las características de las que hace gala.

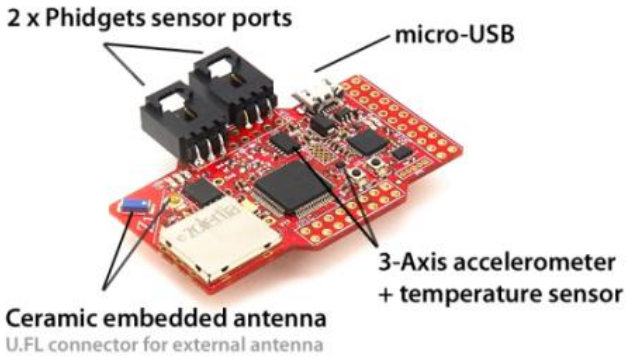


Figura 2-19 : Mote Zolertia Z1

El mote Z1 está equipado con un microcontrolador de bajo consumo MSP430F2617 de segunda generación, el cual dispone de una CPU RISC a 16MHz de frecuencia de reloj, 8 KBytes de memoria RAM y 92 KB de memoria Flash. Además incluye el transceptor CC2420 también de Texas Instruments, compatible con el estándar 802.15.4 y que opera a 2.4 GHz con una tasa efectiva máxima de 250Kbps. Además, el hardware de Z1 garantiza una gran eficiencia energética y robustez.

Este mote dispone a su vez de dos sensores integrados de fábrica. Un sensor de Temperatura y un acelerómetro, ambos digitales y listos para funcionar. Además es compatible con los sensores Phidgets y una gran variedad de sensores analógicos y digitales. Es programable a través del puerto USB. En lo que respecta a la alimentación ofrece un amplio abanico de posibilidades de conexión:

- 2 Pilas AA o AAA.
- Pila de Botón (hasta 3.6V).
- Alimentación directa por USB.
- Conexión directa a una fuente de alimentación a través de dos cables (desde 4 hasta 5 V).

La característica más importante de este mote es que a pesar de ser relativamente reciente, es totalmente compatible tanto con TinyOS como con Contiki. Este hecho es esencial ya que el único mote de los vistos hasta ahora que era compatible con ambos sistemas operativos era el Telosb. Sin embargo, las características técnicas (tanto transceptor de radio como microcontrolador) del Telosb son muy inferiores a las del Z1. Por tanto el mote Z1 es una opción muy interesante tanto por sus características técnicas como por su compatibilidad software.

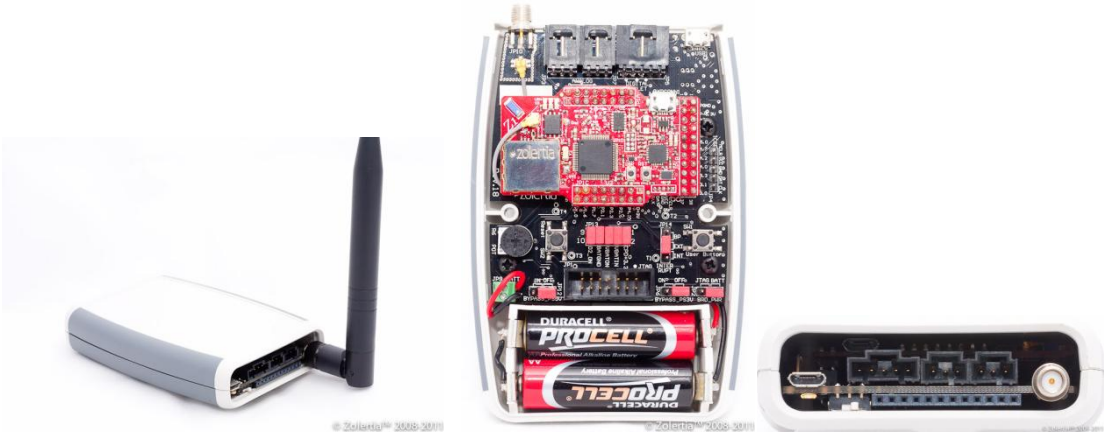


Figura 2-20:Mote Z1 en su caja comercial y con su placa de conexionado (centro)

2.7.6 WASPMote

La empresa Libelium[22] lanzó al mercado en el año 2009 este dispositivo destinado a ser usado en redes de sensores inalámbricas denominado Waspmote.

Figura 2-21: Vista en perspectiva del Waspmote con módulo de radio, GPS y GPRS incorporados

El módulo principal del Waspmote(ver Figura 2-23) de un microcontrolador ATmega1281 que funciona a 8 MHz y dispone de 8 KB de memoria RAM y 128 KB de memoria FLASH. En conjunto con el microcontrolador, la placa principal cuenta con un Reloj en Tiempo Real con su batería de respaldo y de un slot microSD.

La idea de Libelium es vender el resto de componentes del dispositivo en función de las necesidades del cliente. Es por ello que el dispositivo cuenta con diversos conectores de expansión que le permiten añadir diferentes elementos en función de las necesidades finales.

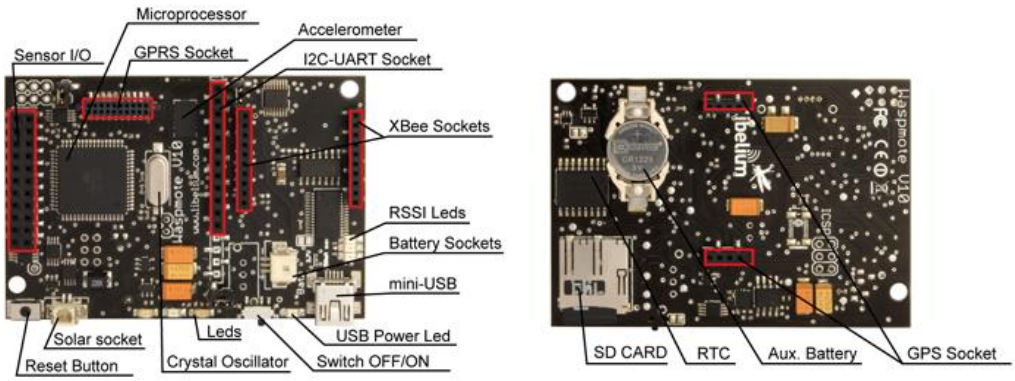


Figura 2-22: Vista de las dos caras de la placa principal del Waspmote de Libelium.

En lo que respecta a la conectividad vía radio, se le pueden añadir diferentes módulos XBee o bien un módulo Bluetooth. Además también permite añadir un módulo GSM/ GPRS para el envío y recepción de datos a través de la red GSM. También es posible añadir un módulo GPS con su correspondiente antena así como diferentes tipos de sensores destinados en su mayoría a la agricultura.

Libelium ofrece además un dispositivo gateway (ver Figura 2-24) que permite comunicarse con los Waspmotes por radio, bluetooth o GSM/GPRS y hacer de pasarela hacia internet. De forma que a pesar de que los Waspmotes no soportan el protocolo IP, el gateway hace de pasarela y nos permite la comunicación a través de él, por ejemplo por medio de una red Wi-Fi.

Figura 2-23 : Gateway comercial de Libelium para la comunicación con los Waspmotes

El principal inconveniente del WaspMote es que no es compatible con Contiki ni con TinyOS. Lo cual limita en gran medida el uso del dispositivo. Ya que se depende totalmente del soporte del fabricante.

2.7.7 Comparativa

En la siguiente Tabla se puede apreciar una comparativa entre los diferentes motes comerciales, mostrando sus características más importantes.

Tabla 2-1:Características principales de diferentes motes

Mote		MICA2 / MICA2Dot	TelosB	Imote2	Zolertia Z1	Waspmote
MCU	*Chip*	ATmega128L	Msp430f1611	PXA271	MSP430F2617	ATmega1281
	Tipo	8 bits	16 bits	32 bits	16 bits	8 bits
	Memoria Programa (KB)	128	48	N/A	96KB	128KB
	RAM (KB)	4	10	32768	8KBytes	8KBytes
Almacenamiento Externo No Volátil	*Chip*	AT45DB014B	M25P80	N/A	N/A	N/A
	Conexión	SPI	SPI	N/A	N/A	SPI
	Tamaño (KB)	512	1024	32768	N/A	SD hasta 2000

Sistema de Alimentación	***Tipo***	2xAA	Coin Cell	2xAA	3xAAA	2xAA/A o Coin Cell	Variable
	Capacidad Típica (mAh)	2850	1000	2850	1100	1100	2000
RF	***Chip***	CC1000		CC2420	CC2420	CC2420	Variable (Xbee)
	Frecuencia	868/916, 433 ó 315 MHz		2,4GHz	2,4GHz	2.4GHz	2.4GHz
	datarate (Kbps)	38,4		250	250	250K	250
	Potencia Transmisión (dBm)	5		0	0	0	Hasta 18dBm
	Alcance Exterior (m)	152,4	152.4-304,8	75-100	~30	150	Hasta3200
Año comercialización		2002	2002	2005	2007	2009	2009

2.8 Microcontroladores Comerciales

Un microcontrolador es un circuito integrado que incluye en su interior las tres unidades funcionales de una computadora: CPU, Memoria y Unidades de E/S, es decir, se trata de un computador completo en un solo circuito integrado. Aunque sus prestaciones son limitadas, además de dicha integración, su característica principal es su alto nivel de especialización. Aunque los hay del tamaño de un sello de correos, lo normal es que sean incluso más pequeños. Es un microprocesador optimizado para ser utilizado para controlar equipos electrónicos..

2.8.1 MSP430

Los μC 's de la familia MSP430 son procesadores con una arquitectura de 16 bits y un conjunto reducido de instrucciones (RISC) cuyo fabricante es Texas Instruments[23]. Tanto los periféricos como la memoria de programa y de datos comparten un mismo espacio de direcciones en una configuración tipo Von-Neumann. El MSP430 integra de manera exitosa una moderna CPU con periféricos análogos y digitales además de implementar distintos modos de operación para lograr uno de los μC de señal mixta más versátiles y de mayor ahorro de energía que se han diseñado últimamente.

En la Figura 2-25 se pueden observar los bloques funcionales de un MSP430 cualquiera.

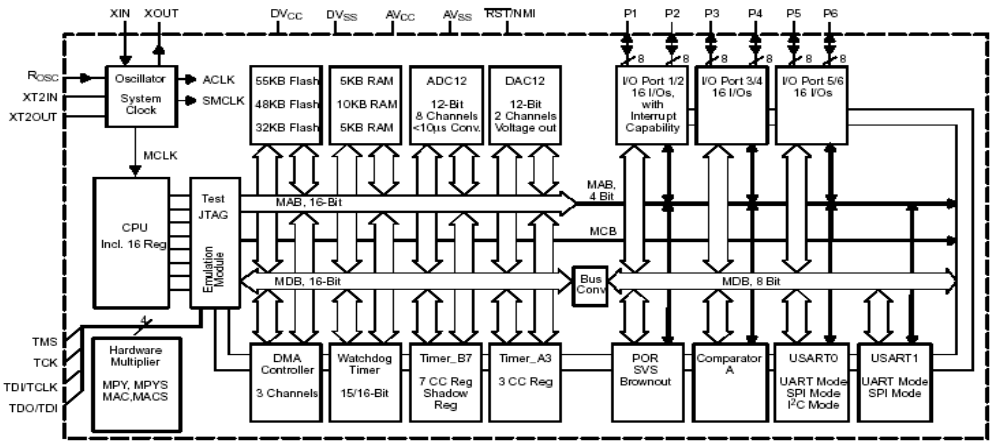


Figura 2-24: Diagrama de bloques para la familia MSP430.

A continuación se presenta una breve descripción de estos:

RISC-CPU: Es la unidad central de procesamiento. Es aquí donde se procesan las instrucciones contenidas en la memoria de programa (Flash).

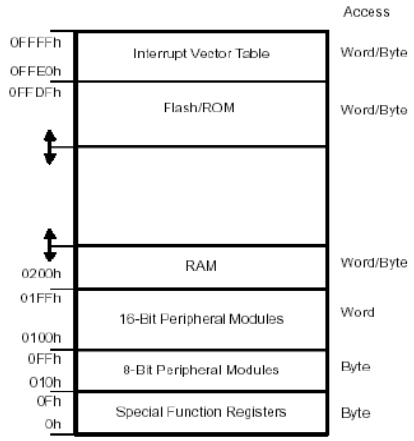


Figura 2-25: Organización del espacio de memoria.

Clock System: El MSP430 cuenta con tres señales de reloj que son generadas a partir de tres fuentes distintas (ver Figura 2-27): un cristal de baja frecuencia, un cristal de alta frecuencia y una señal generada digitalmente a partir de circuitos R-C configurables. A partir de estas tres fuentes se generan las tres señales de reloj:

- Main System clock (MCLK) que es utilizado por la CPU y el sistema.
- Auxiliary Clock (ACLK) que es utilizado por los periféricos.
- Sub-System Clock (SMCLK) que también sirve de fuente para los periféricos.

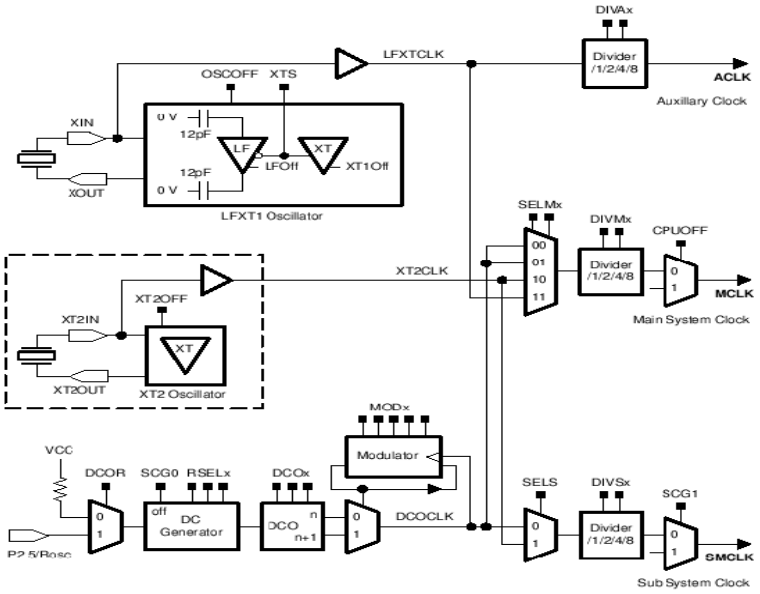


Figura 2-26: Sistema de generación de reloj del MSP430.

Es esta notable flexibilidad a la hora de seleccionar fuentes de reloj lo que le da al MSP430 su gran capacidad de ahorro de energía, al poder utilizar bajas frecuencias en periodos de ocio de la CPU y/o los periféricos. Una baja frecuencia de reloj implica un bajo consumo. En este diseño se optó por usar un cristal de baja frecuencia de 32KHz y uno de alta frecuencia de 8MHz para la CPU. De esta manera, el funcionamiento normal es lo suficientemente rápido y se puede utilizar un modo de bajo consumo.

WatchDog Timer: La función principal del watchdog timer (WDT) es la de realizar un reset controlado del sistema después de que un problema de software ocurra. La detección del problema se logra por medio de sucesivos refrescos al WDT por parte del software. Si el WDT expira, entonces el sistema es reiniciado automáticamente. Adicionalmente, si el WDT no es utilizado para controlar la ejecución del software puede ser utilizado como un timer extra.

Memoria Flash: Esta memoria no volátil se divide en dos zonas(ver Figura 2-26): la zona de código y la zona de información. La zona de código es el lugar donde se almacena el programa a ejecutar por la CPU y la zona de información es una zona de libre propósito para uso del programador.

Memoria RAM: Es una memoria volátil que se utiliza para las variables del programa.

Timer A y B: Este periférico puede funcionar como un timer o un contador de 16 bit, con tres registros de captura o comparación, salidas de PWM (Pulse Width Modulation) y medición de intervalos. El timer posee amplias capacidades de interrupción, las que pueden ser generadas por un overflow del contador o por los registros de captura y comparación.

USART 0 y 1: Los MSP430 generalmente incorporan 2 USART, dependiendo del modelo, estas pueden funcionar hasta en tres modos: modo UART, modo I2C y modo SPI.

- El modo UART es un modo de transmisión asincrónico que se comunica por medio de dos pines, UTXD y URXD. Este modo es compatible con el estándar de transmisión RS-232.
- El modo I2C provee comunicación con cualquier dispositivo que implemente este protocolo. Este modo utiliza 2 cables para transmitir y recibir información.
- El modo SPI conecta el µC con otro dispositivo por medio de tres o cuatro líneas.

2.8.2 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: compiladores de C, ensambladores de macro, programas depuradores/simuladores, Emuladores y kits de evaluación.

El microcontrolador Atmega128 combina un rico 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 CISC convencionales.

El ATmega128 ofrece las siguientes características:

- Avanzada arquitectura RISC
- Interfaz JTAG (estándar IEEE 1149,1 Compliant)
- Dos contadores (Timer / Counters)de 8 bits con Separe Prescalers y modo de comparación.
- Dos contadores ampliados de 16 bits con Separe Prescaler, modo de comparación y modo de captura.
- Dos Canales de PWM de 8 bits.
- 6 PWM canales programables con la Resolución del 2 a 16 Bits .
- 8 canales de 10 bits ADC : 8 Canales simples, 7 Canales diferenciales, 2 canales diferenciales programables.
- La doble programable de serie UARTs.
- SPI interfaz serie Maestro / esclavo

2.9 Transceptores de Radio comerciales

El término **transceptor** se aplica a un dispositivo que realiza, dentro de una misma caja o chasis, funciones tanto de transmisión como de recepción, utilizando componentes de circuito comunes para ambas funciones. Dado que determinados elementos se utilizan tanto para la transmisión como para la recepción, la comunicación que provee un transceptor solo puede ser semiduplex, lo que significa que pueden enviarse señales entre dos terminales en ambos sentidos, pero no simultáneamente.

2.9.1 Chipcon CC2420

El chip CC2420 compatible con IEEE 802.15.4 es un transmisor de radio a 2,4 GHz de Texas Instruments[23]. Fue diseñado para aplicaciones inalámbricas de baja potencia y baja tensión. CC2420 incluye un espectro digital extendido de secuencia directa y una efectiva transmisión de 250 kbps.

El CC2420 es un dispositivo de bajo coste, está altamente integrado y ofrece una robusta solución para la comunicación inalámbrica en la banda ISM de 2,4 GHz. El CC2420 proporciona un amplio hardware de apoyo a la manipulación de paquetes, los Búfer de datos, ráfagas de las transmisiones, la encriptación y autenticación de los datos y de paquetes de información.

Sus características principales son:

- Chip de 2,4 GHz compatible con IEEE 802.15.4 con base módem y Apoyo MAC
- DSSS base módem con 2 MChips y 250 kbps de Tasa efectivo de datos
- Descarga Bajo consumo (RX: 18,8 MA; TX: 17,4 mA)
- Baja tensión de alimentación (2,1 - 3,6 V) con regulador de voltaje Integrado
- Baja tensión de alimentación (1,6 - 2,0 V) con Regulador de tensión externa
- Potencia de salida programable

2.9.2 Xbee

Los módulos XBee y XBee - PRO OEM de RF [24] fueron diseñados para cumplir con el estándar IEEE 802.15.4 y cumplir con la necesidad de bajo costo y baja potencia de las redes de sensores inalámbricas. Los módulos requieren un mínimo de energía para la

entrega de los datos entre los dispositivos y funcionan en la banda ISM de 2,4 GHz de frecuencia.

Tabla 2-2: Principales características de los módulos XBee y XBee Pro

Especificaciones		Xbee	Xbee-Pro
Rendimiento	Alcance en ambientes interiores / zonas urbanas	Hasta 30 m	Hasta 100 m
	Alcance de RF en línea de Visión para ambientes exteriores	Hasta 100 m	Hasta 1200 m
	Potencia de salida de Transmisión	1 mW (0 dB)	60mW (18 dB), 100mW EIRP
	Régimen RF de datos	250.000 bps	250.000 bps
	Sensibilidad del receptor	-92 dBm(1% PER)	-100 dBm (1% PER)
Rendimientos de potencia	Suministro de voltaje	2.8-3.4 V	2.8-3.4 V
	Corriente de transmisión	45 mA @3.3 V	270 mA @3.3V
	Corriente de recepción	50 mA @3.3 V	55 mA @ 3.3 V
	Corriente Power-Down	<10 uA	<10 uA
Información general	Frecuencia	ISM 2.4 GHz	ISM 2.4 GHz

2.9.3 Chipcon CC2520

El módulo CC2520 es la segunda generación ZigBee / IEEE 802.15.4 transceiver para la banda de 2.4GHz en la banda ISM. Este chip habilita a diferentes tipos de aplicaciones ofreciendo una excelente calidad del enlace de transmisión radio, puede operar hasta a 125°C y operando a bajo voltaje.

Además, el módulo de radio CC2520 proporciona soporte para "frame handling", "data buffering", "burst transmissions", "data encryption", "data authentication", "clear channel assessment", indicación de calidad de enlace e información de los tiempos de la

trama. Estas características reducen la carga en el controlador host. En un sistema típico, el CC2520 debe ser usado en conjunto con un microcontrolador y una serie de componentes pasivos (resistencias, bobinas, condensadores etc).

El dispositivo proporciona una excelente calidad de enlace (103 dB) y un rango de 400 m en linea de visión directa. El rango de temperaturas de operación se encuentra entre –40°C to +125°C. El rango de voltajes de operación se encuentra entre 1.8 y 3.8 V.

Otra de las características importantes es la capacidad para utilizar encriptación AES-128 y el modo de compatibilidad con el módulo CC2420 de TI.

La principal diferencia respecto al Módulo de radio CC2420 es su menor consumo tanto en recepción como en transmisión:

- (–50 dBm) 18.5 mA en recepción.
- 33.6 mA @ +5 dBm en transmisión.

2.10 Sistemas operativos para dispositivos embebidos

Las necesidades que tiene un nodo de una WSN son totalmente distintas a las que pueda tener cualquier otro dispositivo como puede ser un PC, por lo tanto estos nodos tiene sus propios sistemas operativos.

Los sistemas operativos para WSN son típicamente menos complejos que los de propósito general, tanto debido a los requisitos especiales de las aplicaciones en las que se usan, como a las restricciones de recursos encontradas en las plataformas hardware utilizadas. Por ejemplo, las aplicaciones de WSN no son tan interactivas como son las aplicaciones para PC y debido a esto, estos sistemas no necesitan incluir el soporte de interfaces de usuario. Además, las restricciones de los recursos en términos de memoria hacen imposible de implementar los mecanismos de memoria virtual.

Los nodos sensores incluyen un microcontrolador capaz de ejecutar tareas que requieren el acceso a elementos de hardware (sensores, memoria, radio, etc.). Las funciones principales de un sistema operativo son :

- Gestionar eficientemente los recursos de hardware
- Facilitar la programación de aplicaciones de alto nivel

En la Figura 2-28 se puede apreciar el nivel que ocupa el sistema operativo dentro de las diferentes capas y niveles de abstracción.

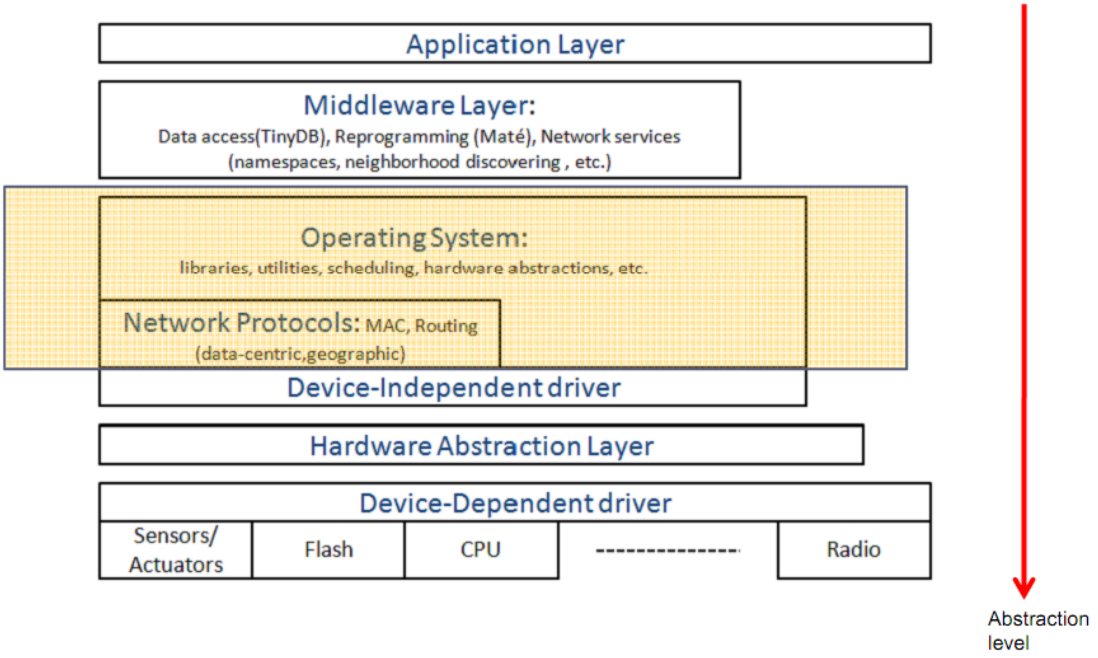


Figura 2-27: Niveles de abstracción y capas que los componen.

Estos sistemas operativos están diseñados específicamente teniendo en cuenta las restricciones de hardware de los nodos sensores. Ya que los sistemas operativos diseñados para otro tipo de sistemas empotrados no se adaptan a las fuertes restricciones de los nodos sensores, sobretodo relacionadas con el tamaño de memoria, consumo y requisitos de las aplicaciones.

El sistema operativo es responsable de:

- Gestionar el microcontrolador (típicamente monotarea), tiempo (temporizadores) y concurrencia.
- Gestionar el resto de dispositivos del hardware.
- Ofrecer interfaces de acceso (APIs) a los elementos de hardware (sensores, memoria, radio, etc.).
- Programación de aplicaciones.
- Ofrecer una interfaz para la instalación del código ejecutable en el microcontrolador.
- Ahorro de energía.
- Multiplataforma: Soportan un subconjunto de las plataformas disponibles.
- Arquitectura monolítica: Cuando se compila la aplicación se crea una imagen indivisible compuesta por el S.O. y la propia aplicación.
- Footprint (huella) de pequeño tamaño: Limitado por la capacidad de memoria (e.g. 4KB RAM y 128 KB ROM).
- Ofrecen interfaces de alto nivel (al nivel de aplicación) y usan el interfaz de bajo nivel para acceder a los dispositivos de hardware.
- Proveen implementaciones para la gestión de red (protocolos de enrutamiento, MAC, localización), memoria flash (sistemas de ficheros), temporización (relojes de software), etc.

2.11 Sistemas operativos existentes para los dispositivos utilizados en las WSN

En el marco de los sistemas operativos existentes para el trabajo con dispositivos destinados a ser usados en Redes Inalámbricas de Sensores, TinyOS fue el primer sistema operativo diseñado específicamente para ellas. A diferencia de la mayoría sistemas operativos, TinyOS se basa en un modelo de programación controlado por eventos, en lugar de multiprocesos. Los programas de TinyOS están compuestos por eventos y tareas guiadas. Cuando un evento externo ocurre, como puede ser la entrada de un paquete de datos o la lectura de un sensor, TinyOS llama al evento apropiado y lo ejecuta. El lanzador de eventos puede posponer tareas durante cierto tiempo. Tanto TinyOS como los

programas escritos para él son escritos en un lenguaje de programación especial llamado NesC, que es una extensión del lenguaje de programación C. NesC está diseñado para determinar las prioridades entre tareas y eventos.

Hay también sistemas operativos que permiten programar en C. Por ejemplo Contiki, MANTIS, BTnut, SOS y Nano-RK. Contiki está diseñado para soportar carga de módulos a través de la red y para soportar cargas de ficheros ELF. El kernel de Contiki está basado en lanzamiento de eventos, como TinyOS, pero puede llegar a soportar multitareas básicas. A diferencia del kernel de Contiki basado en eventos, los kernel MANTIS y Nano-RK están basados en multitareas preventivas. El kernel divide el tiempo entre los procesos activos y decide qué proceso debe ser ejecutado para hacer la programación de la aplicación más fácil. Nano-RK tiene un kernel basado en tiempo real que permite controlar la manera de acceder de las tareas, a la CPU, a la red y a las placas sensoras. Como TinyOS y Contiki, SOS es un sistema operativo basado en eventos. La principal característica de SOS es su soporte para módulos cargables. Un sistema complejo se construye a partir de módulos más pequeños, probablemente en tiempo de ejecución. Para soportar el dinamismo inherente en su módulo de interfaz, SOS soporta una gestión de la memoria dinámica. BTnut se basa en multitareas cooperativas y se propaga en código C plano, además viene empaquetado con un kit de desarrollo y manuales.

2.11.1 TinyOS

TinyOS [25] es un sistema operativo orientado a trabajar con redes de sensores, desarrollado en la Universidad de Berkeley. 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 nos permiten 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.

El lenguaje en el que se encuentra programado TinyOS es un meta-lenguaje que deriva de C, cuyo nombre es NesC. Además existen varias herramientas que ayudan al estudio y desarrollo de aplicaciones para las redes de sensores, que van desde aplicaciones para la obtención y manejo de datos, hasta sistemas complejos de simulación.

El diseño de TinyOS está basado en responder a las características y necesidades de las redes de sensores, tales como reducido tamaño de memoria, bajo consumo de energía, operaciones de concurrencia intensiva, diversidad en diseños y usos, y finalmente operaciones robustas para facilitar el desarrollo confiable de aplicaciones. Además se encuentra optimizado en términos de uso de memoria y eficiencia de energía.

El diseño del Kernel de TinyOS está basado en una estructura de dos niveles de planificación.

- Eventos: pensados para realizar un proceso pequeño (por ejemplo cuando el contador del timer se interrumpe, o atender las interrupciones de un conversor analógico-digital). Además pueden interrumpir las tareas que se están ejecutando.

- Tareas: las tareas están pensadas para hacer una cantidad mayor de procesamiento y no son críticas en tiempo (por ejemplo calcular el promedio en un array). Las tareas se ejecutan en su totalidad, pero la solicitud de iniciar una tarea, y el término de ella son funciones separadas.

Con este diseño permitimos que los eventos (que son rápidamente ejecutables), puedan ser realizados inmediatamente, pudiendo interrumpir a las tareas que tienen mayor complejidad en comparación a los eventos.

El enfoque basado en eventos es la solución ideal para alcanzar un alto rendimiento en aplicaciones de concurrencia intensiva. Adicionalmente, este enfoque usa las capacidades de la CPU de manera eficiente y de esta forma gasta el mínimo de energía.

TinyOS se encuentra programado en NesC, un lenguaje diseñado para reflejar las ideas propias del enfoque de componentes, incorporando además un modelo de programación que soporta concurrencia, manejo de comunicaciones y fácil interacción con el medio (manejo de hardware).

2.11.2 Microsoft .Net Micro Framework

La .NET Micro Framework[26] fue creada desde el inicio como una solución .NET para dispositivos integrados pequeños de sensores industriales e instrumentación para sistemas empotrados.

Además de estar totalmente integrada con Visual Studio, el kit de desarrollo de software .NET Micro Framework (SDK) viene equipado con un emulador extensible para simular capacidades de hardware. La estructura permite a los desarrolladores de dispositivos conectar diversas soluciones de hardware para prácticamente cualquier dispositivo periférico mediante conexiones de comunicación estándares de la industria y unidades gestionadas personalizadamente.

La .NET Micro Framework dispone de ofertas integradas de Microsoft en un nuevo mercado de dispositivos que se basan en procesadores de 32 bits de bajo coste y están constreñidos en términos de memoria, potencia de la batería y otros recursos. Ofreciendo paradigmas de programación potentes y modernos a este terreno, .NET Micro Framework pretende acelerar la innovación de dispositivos pequeños y conectados.

- Requiere sólo unos pocos cientos de kilobytes de RAM, y tan poco como 512 K de memoria flash.

- Soporta procesadores con y sin MMU.

- Dispone de una interfaz de control de energía que permite que la aplicación maximice la vida de la batería.

Hasta ahora Microsoft ha publicado el SDK de Microsoft .NET Micro Framework 2.5 el cual permite desarrollar aplicaciones para pequeños dispositivos utilizando para tal motivo, Visual Studio, C#. Esta parte de Visual Studio es gratis y puede descargarse de la página Web de Microsoft, su descarga es de apenas 9Mb y en inglés por ahora para Visual Studio 2005 con SP1. Microsoft .NET Micro Framework es un pequeño subconjunto reducido de clases, en donde podemos desarrollar soluciones en C# para pequeños dispositivos, para trabajar con este pequeño Framework se necesita disponer de Microsoft Visual Studio 2005 Standard o superior, y un sistema operativo Windows XP, Vista o Server 2003. Dispone de un conjunto de librería .NET completamente integradas enfocadas al uso en sistemas empotrados.

Varias empresas están apostando por esta tecnología, Digi International Inc dio a conocer sus planes para un lanzamiento previo del kit de desarrollo Digi Connect ME para Microsoft .NET Micro Framework. El Digi Connect ME incluye soporte para redes Ethernet, un puerto serie y señales de propósito generales entrada/salida (GPIO). Es la primera solución disponible para .NET Micro Framework para dar apoyo a las redes Ethernet.

EmbeddedFusion, que entrega soluciones centrales de hardware y software integrado para desarrolladores de sistemas integrados, anunció el Meridian CPU, que es un módulo central CPU que incorpora procesadores Freescale i.MXS, RAM, Flash y .NET Micro Framework. Para asistir a los desarrolladores en el aprendizaje de cómo se aplica .NET Micro Framework en varios escenarios integrados, EmbeddedFusion también creó la plataforma de desarrollo Tahoe, que permite la experimentación y exploración de .NET Micro Framework.

Freescale también introdujo un kit de desarrollo para .NET Micro Framework que permite a los clientes entregar soluciones diferenciadas en el mercado con rendimiento ARM9 a muy baja potencia.

Además, Rhode Consulting, un especialista en tecnologías Microsoft Windows Embedded, anunció la disponibilidad del kit de evaluación de FlexiDis con .NET Micro Framework instalado. La plataforma FlexiDis utiliza los procesadores centrales Atmel ARM7 y ARM9 con velocidad de hasta 180 Mhz. La combinación de esas velocidades, hasta 16 Mb de memoria flash y SDRAM, y una pantalla QVGA de 2,2 hace de FlexiDis un componente de elección para varios tipos de aplicaciones industriales en las que se requieren HMI integrado o soluciones de visualización.

2.11.3 MANTIS

MANTIS[27](*MultimodAl NeTworks In-situ Sensors*). El sistema operativo MANTIS suministra un nuevo sistema operativo empotrado de plataforma múltiple para redes de sensores inalámbricos. Ante el incremento de complejidad de las tareas realizadas por las redes de sensores como compresión, agregación y procesado de señales, los multiprocesos MANTIS sensor OS (MOS) permiten interpaginar complejas tareas con tareas susceptibles al tiempo para así mitigar los problemas en los saltos de buffers. Para conseguir una eficiencia en el uso de la memoria, MOS es implementado para que utilice una pequeña cantidad de RAM. Usando menos de 500 bytes de memoria, incluyendo el kernel, los controles de tiempo y la pila de comunicación. Para conseguir la eficiencia energética, el controlador de eficiencia energética de MOS hace que el microcontrolador duerma después de ejecutar todas las tareas activas, reduciendo el consumo de energía a un rango de μA.

Una de las características principales de MOS es la flexibilidad en el soporte de múltiples plataformas como PCs, PDAs y diferentes plataformas de microsensores. Otra de las características destacada del diseño de MOS es el soporte de control remoto, permitiendo una reprogramación dinámica y un acceso remoto.

2.11.4 eCos

eCos[28](embedded Configurable operating system) es un sistema operativo de código abierto, gratuito y de operación en tiempo real desarrollado para sistemas empotrados y para aplicaciones que necesiten un procesador con múltiples sesiones. Puede ser personalizado para cumplir los requisitos que la aplicación precise, con cientos de opciones, pudiendo llegar a la mejor combinación entre el rendimiento en tiempo real y el hardware necesario.Este sistema es programable bajo lenguaje C y tiene capas y APIs compatibles para POSIX y μITRON.

Ecos fue diseñado para aparatos con un tamaño de memoria sobre cientos de kilobytes o con requerimientos en tiempo real. Puede ser usado en hardware con muy poco RAM soportando Linux empotrado a partir de un mínimo de 2 MB de RAM, sin incluir las necesidades de la aplicación y del servicio.

eCos funciona correctamente en una amplia variedad de plataformas hardware como pueden ser ARM, CalmRISC, FR-V, Hitachi H8, IA-32, Motorola 68000, Matsushita AM3x, MIPS, NEC V8xx, Nios II, PowerPC, SPARC y SuperH.

Incluido con la distribución de eCos disponemos de RedBoot, una aplicación de código abierto que usa la capa de abstracción de hardware de eCos que provee soporte de arranque para sistemas empotrados.

eCos fue inicialmente desarrollado por Cygnus Solutions, aunque más tarde fue comprado por Red Hat. A principios de 2002 cesó el desarrollo de eCos y se ubicó al personal que estaba trabajando en el proyecto en otra compañía recién formada, eCosCentric, para continuar con su desarrollo y dar soporte comercial para eCos. En Enero de 2004, a partir de una solicitud de los desarrolladores de eCos, Red Hat aceptó transferir los derechos de eCos a la Fundación de Software Libre. Esta transferencia fue finalmente ejecutada en octubre de 2005.

ECosPro está formado por una distribución de eCos y RedBoot creada por eCosCentric y está orientada a desarrolladores que quieran integrar eCos y RedBoot dentro de productos comerciales. Está definido como estable, completamente testeado, certificado y con soporte, sin embargo, algunas de sus características no han sido liberadas como software libre.

2.11.5 µC/OS

MicroC/OS-III [29], es un sistema operativo multitarea, en tiempo real, basado en prioridad preventiva, de bajo coste donde el kernel está escrito principalmente en el lenguaje de programación C. Es principalmente entendido para uso en sistemas empotrados.

La designación II es debido a que es la segunda generación de un kernel que originalmente fue publicado en 1992 en la segunda parte de un artículo en la revista Embedded Systems Programming bajo el título µC/OS The Real-Time Kernel y escrito por Jean J. Labrosse. El autor intentó describir como funciona un sistema operativo portable por dentro y las razones por las que µC/OS-II es soportado por Micrium Inc y se obtiene bajo licencia del producto, aunque el uso de este sistema operativo es gratis para uso educacional o no comercial. Adicionalmente Micrium distribuye otros productos software como µC/OS-View, µC/CAN, µC/TCP-IP, µC/FS, µC/GUI, µC/MOD-BUS, µC/LCD, µC/USB y un largo grupo de aplicaciones para µC/TCP-IP como software cliente para DHCP, POP3, SNTP, FTP, TFTP, DNS, SMTP, Y TTCP. El software en su modalidad de servicio incluye http, FTP Y TFTP. PPT es también disponible.

Está disponible para la mayor cantidad de procesadores y placas que existen en el mercado y es adecuado para el uso en sistemas empotrados donde la seguridad es crítica como en aviación, sistemas médicos o instalaciones nucleares.

2.11.6 Contiki

Contiki[30] es un pequeño sistema operativo de código abierto, altamente portable y multitarea, desarrollado para uso para uso en pequeños sistemas, desde ordenadores de 8-bits a sistemas empotrados sobre microcontroladores, incluyendo nodos de redes de sensores. El nombre Contiki viene de la famosa balsa Kon-Tiki de Thor Heyerdahl.

A pesar de incluir multitarea y una pila TCP/IP, Contiki sólo requiere varios kilobytes de código y unos cientos de bytes de RAM. Un sistema totalmente completo con una GUI requiere aproximadamente 30 kilobytes de RAM.

El núcleo básico y la mayor parte de las funciones principales fueron desarrollados por Adam Dunkels en el grupo de sistemas de redes empotradas en el instituto sueco de ciencias computacionales.

Contiki fue diseñado para sistemas empotrados con poca cantidad de memoria. Una configuración típica de Contiki es 2 kilobytes de RAM y 40 kilobytes de ROM. Contiki consiste en un núcleo orientado a eventos, el cual hace uso de protothreads, sobre el cual los programas son cargados y descargados dinámicamente. También soporta multihilado apropiativo opcional por proceso, comunicación entre procesos mediante paso de mensajes a través de eventos, al igual que un subsistema GUI opcional, el cual puede usar un soporte directo de gráficos para terminales locales, terminales virtuales en red mediante VNC o sobre Telnet.

Contiki funciona en una variedad de plataformas, desde microcontroladores empotrados, como el MSP430 y el AVR, a viejas computadores domésticas. El tamaño del código está en el orden de los kilobytes y el uso de la memoria puede configurarse para que sea de sólo unas decenas de bytes.

2.11.7 SOS

SOS[31] es un sistema operativo desarrollado en la Universidad de California (UCLA), específicamente en el NESL (Networked & Embedded Systems Laboratory).

SOS es un sistema operativo para redes de sensores que procura remediar algunos de las limitaciones propias de la naturaleza estática de muchos de los sistemas precursores a éste (por ejemplo TinyOS).

SOS implementa un sistema de mensajería que permite múltiples hebras 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. Ésta se define como la habilidad para modificar el software de nodos individuales en una red de sensores, una vez que éstos han sido desplegados físicamente e iniciado su funcionamiento. En el caso de encontrar un problema, en caso de no contar con esta solución, habría sido necesario recolectar todos los nodos para poder modificar su software.

La capacidad de dinámicamente agregar o remover módulos, permite la construcción de software mucho más tolerante a fallos. Esto presenta dos grandes ventajas: una es el hecho de poder realizar updates fácilmente, y la otra es la capacidad de anular el funcionamiento de algún módulo defectuoso, de algún nodo que pertenece a la red.

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

- Módulos cargados dinámicamente.
- Programación flexible de prioridades.
- Simple subsistema de memoria dinámica.

2.11.8 Nano-RK

Nano-RK[32] es un sistema operativo completamente preventivo basado en reserva bajo tiempo real (RTOS) desarrollado en la universidad de Carnegie Mellon con soporte para redes multisalto adecuado para el uso en redes de sensores inalámbricas. Nano-RK funciona adecuadamente con plataformas como redes de sensores FireFly y sobre los motes MicaZ.

Incluye un kernel con recursos empotrados de bajo peso con bastantes funcionalidades y soporte de tiempo usando menos de 2 KB de memoria RAM y 18 KB de ROM. Nano-RK soporta multitareas preventivas con prioridad para asegurar que los plazos de las tareas son conocidos, además de soporte de CPU, red y sensores y actuadores.

Las tareas pueden especificar las demandas de recursos y el sistema operativo provee el acceso controlado y garantizado para los ciclos de CPU y los paquetes de red. Todos estos recursos forman la reserva de energía virtual que permite al sistema operativo controlar el nivel de energía del sistema y de las tareas.

2.11.9Otras implementaciones

Todos los sistema operativos vistos anteriormente son en su mayoria de código abierto, sin embargo, los propios fabricantes de dispositivos electrónicos (Texas Instruments, Jennic, National Instruments, etc) suelen proporcionar sistemas operativos (con sus correspondientes pilas de comunicaciones) gratuitos para ser usados por sus clientes en sus dispositivos.

La principal diferencia entre los sistemas operativos proporcionados por los fabricantes y los sistemas operativos de código libre analizados anteriormente es que los sistemas operativos de código libre son compatibles con diversos microcontroladores y chips de radio, independientemente del fabricante de los mismos, sin embargo los sistemas operativos proporcionados por los fabricantes sólo podrán ser utilizados en los dispositivos del propio fabricante.

Por ejemplo:

- Jennic[33] proporciona diversas alternativas cada una de ellas con diversas características (Número máximo de nodos soportados, protocolo de comunicación utilizado, facilidad de desarrollo, etc) dejando a la elección del usuario final de los dispositivos la pila que desee utilizar:

Tabla 2-3: Pilas de comunicaciones proporcionadas por el fabricante Jennic.

Criteria	IEEE802.15.4	JenNet	JenNet-IP	ZigBee PRO
Recommended Topologies	Star Point-to-point	Tree Star Linear	Tree Star Linear	Mesh
Maximum Network Size	50 nodes	500 nodes	500 nodes	50 nodes
Network Recovery	None	Self-repairing	Self-repairing	Self-repairing
Development Complexity	Medium	Low	Low	High
Standards Compliance	IEEE802.15.4 standards	Proprietary networking layer built on standard IEEE802.15.4 layers	IETF IP, UDP and 6LoWPAN on standard IEEE802.15.4 layers	ZigBee standard networking built on standard IEEE802.15.4 layers
Third-party Interoperability	No provision	No provision	Interoperability possible through public defined MIBs accessed via SNAP	Interoperability through ZigBee public profiles and compliance / certification
Licensing Costs	Free	Free	License-free Compliance-free	ZigBee Alliance membership and product certification fees
Solutions	Cable Replacement Remote Control Logistics Audio	Cable Replacement Active RFID Intelligent Lighting	A/V RF Remote Control Smart Lighting Healthcare Security Smart Energy Asset Management Building Control Environment Monitoring	Smart Energy Home Automation

Se puede apreciar en la tabla adjunta, que el fabricante Jennic proporciona 4 implementaciones de sistemas operativos para ser usadas en sus dispositivos:

1. IEEE802.15.4
   Implementación del protocolo IEEE802.15.4 punto a punto.

2. JenNet
   Similar a la anterior pero proporciona soporte para topología en árbol, es decir implememenata internamente el autoencaminamiento.

3. JenNet-IP
   Proporciona soporte para IPv6 (6LoWPAN), implementación del protocolo IPv6 desarrollado especificamente para ser usado en dispositivos embebidos.

4. Zigbee PRO
   Proporciona una implementación de la pila de protocolos ZigBee implementada bajo las capas del estandar IEEE802.15.4.

2.12 ZStack.

Otros fabricantes como Texas Instruments[23] también proporcionan una serie de bibliotecas escritas en C que soportan la pila de protocolos ZigBee, denominada **Zstack.** Sin embargo no proporciona ninguna implementación de 6LoWPAN.

La pila de comunicaciones Zstack, es una implementación ZigBee del estandard IEEE 802.15.4. por la Alianza ZigBee. Ofrece soporte para el amplificador CC2591, el cual permite hasta 25 dBm de potencia de transmisión. Así como permite hacer uso del sistema OAD ("over air download"), el cual permite programar los dispositivos a través de la radio de los mismos.

La pila consta de los siguientes componentes:

- ***HAL (Hardware Abtraction Layer***): Capa de Abstracción de Hardware. Éste es el componente a más bajo nivel, se encarga de abstraer al programador de las capas inferiores de hardware.

  Por ejemplo:

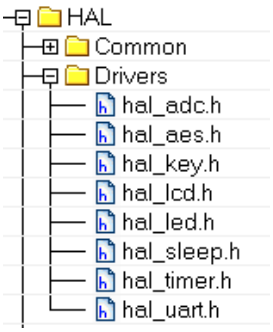


Figura 2-28: Capa HAL de la pila Zstack

En la figura anterior se puede apreciar algunas de las bibliotecas dentro de la capa HAL. En este caso , por ejemplo el archivo "hal_adc.h" proporciona una serie de funciones que para hacer más sencillo el uso del conversor adc.

```
/**************************************************************************
 *                                          FUNCTIONS - API
 *************************************************************************/

/*
 * Initialize ADC Service
 */
extern void HalAdcInit ( void );

/*
 * Read value from a specified ADC Channel at the given resolution
 */
extern uint16 HalAdcRead ( uint8 channel, uint8 resolution );
```

Figura 2-29: Funciones que proporciona la biblioteca HAL_adc.

Como se puede apreciar en la Figura 2-30, proporciona dos funciones: Una para configurar e inicializar el conversor ADC y otra para leer un dato del conversor.

- ***OSAL (Operating system abstraction layer):*** Capa de abstracción del sistema operativo. Proporciona una serie de funciones a alto nivel de diversa índole.

  Por ejemplo, la función osal_memcmp(..):

```
uint8 osal_memcmp( const void GENERIC *src1, const void GENERIC *src2, unsigned int len )
{
  const uint8 GENERIC *pSrc1;
  const uint8 GENERIC *pSrc2;

  pSrc1 = src1;
  pSrc2 = src2;

  while ( len-- )
  {
    if( *pSrc1++ != *pSrc2++ )
      return FALSE;
  }
  return TRUE;
}
```

Figura 2-30: Función de la capa OSAL.

En la figura anterior se puede apreciar que la función osal_memcp(..) compara dos cadenas de caracteres y devuelve true si son las mismas y false en caso contrario.

- ***ZigBee Stack + IEEE 802.15.4 MAC:*** Esta capa se encarga de implementar el protocolo de comunicación ZigBee, ofrece una gran cantidad de funciones tanto a nivel de capa MAC "ZMAC.h" como a nivel de aplicacion "ZDO.h".

- ***Aplicación de Usuario:*** (ver Figura 2-32)Componente software que permite al usuario escribir el código de su aplicación haciendo uso de las funciones ofrecidas por las capas inferiores.

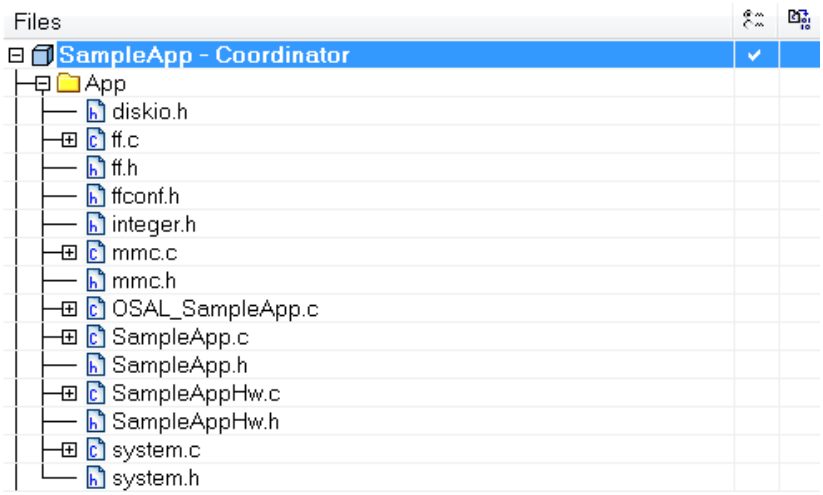


Figura 2-31: Aplicación de usuario. ("SampleApp.c" y "SampleApp.h")

2.13 Protocolos de comunicación más usados en los dispositivos de las WSN´s.

2.13.1 Protocolo SPI

El protocolo SPI (Serial Pheriferal Interface) como ya indica su nombre, es un sistema de comunicación serie entre periféricos, es un sistema de bajo coste y baja velocidad para comunicaciones de corta distancia, como por ejemplo, entre pequeños procesadores y sus periféricos.

Es un sistema full duplex, muy fácil de implementar entre dos hosts. Si este sistema se utiliza para más de dos hosts, empieza a perder sus ventajas pero el sistema de bus, basado en el protocolo I2C, funciona mejor. Sin embargo, SPI es capaz de ofrecer ratios mucho mayores, pudiendo llegar a las decenas de megahercios.

El protocolo SPI suele constar de un componente maestro y uno o más componentes esclavos. El maestro, se define normalmente como un microcomputador provisto de un reloj SPI (SPI clock), y los esclavos, como cualquier circuito integrado, reciben el reloj SPI del maestro. El ASIC (Application-Specific Integrated Circuit) en los productos de la tecnología VTI siempre opera, como componentes esclavos en modo de operación maestro-esclavo.

SPI, es una interfaz serie síncrona de 4 hilos, la comunicación de datos se activa mediante una señal baja, aplicada en la entrada Slave Select (SS) o en el Chip Select (CSB). Los datos, se transmiten mediante 3 conexiones: conexión para la entrada de datos serie (MOSI), la de salida de datos (MISO) y la señal de reloj (SCK)(ver Figuras 2-33,234, y tabla 2-4).

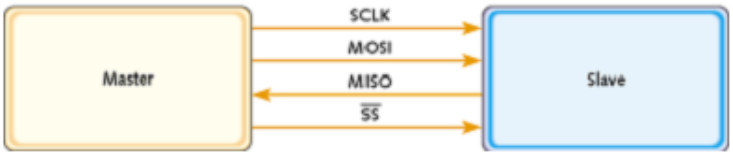


Figura 2-32: Configuración Master y Esclavo.

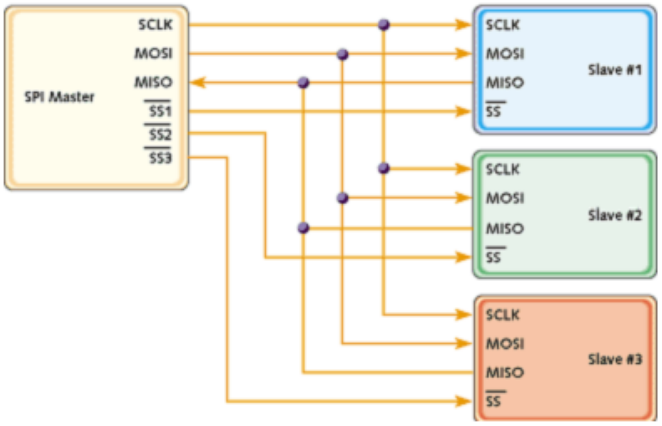


Figura 2-33: Configuración Master y varios Esclavos.

Tabla 2-4:Descripción de las señales del Bus SPI.

Signal Name	Description
SCLK	Clock
MOSI	Master out - Slave in
MISO	Master in - Slave out
SS	Slave select

El master, genera una señal de reloj y la envía a los esclavos. La línea de selección, slave line, se utiliza para indicar con que esclavo se está intentando comunicar el Master.

Como se puede ver, todas las señales excepto la de selección de esclavo son comunes a todos los esclavos, por eso, el master debe indicar cuál de los esclavos esta activo durante la comunicación.

Se puede imaginar, que con más de uno o dos esclavos, el número de salidas del master y el número de pistas en la placa, serian demasiadas para justificar el uso de este protocolo.

SPI usa un par de parámetros llamados clock polarity (CPOL) y clock phase (CPHA), para determinar, cuando los datos son validos dependiendo del señal de reloj. Estos deben establecerse, tanto en el dispositivo maestro como en los esclavos, para que la comunicación funcione correctamente. Mediante estos dos parámetros, se determina cual es el momento idóneo de muestreo.

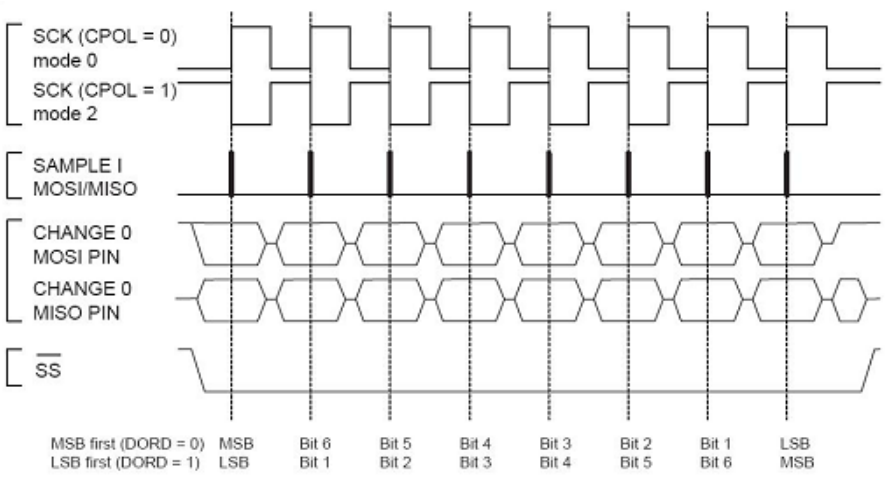


Figura 2-34: Transferencia de datos bus SPI.

En este ejemplo (Figura 2-35), se puede observar que tanto el MSB como el LSB se pueden transmitir en primer lugar, dependiendo de cómo está configurado el hardware SPI.

En nuestro caso, nos interesará sobretodo la comunicación entre Master y Esclavo ya que es el modo en que funcionará la comunicación entre la SD y el uC, el funcionamiento es el siguiente:

1. El Master pone en nivel bajo la línea slave select del esclavo con el que quiere interactuar. Esto indica al esclavo, que debe prepararse para iniciar la comunicación.
2. El master, genera la señal de clock de acuerdo con el modo SPI. Tanto el master como el esclavo transmiten un bit por ciclo de reloj.
3. Después de un byte, el Master pone la línea slave select en nivel alto.

Se han descrito las partes más importantes del protocolo SPI. No hay más conocimiento o estándar de comunicaciones avanzado, que se pueda implementar por encima de esta capa. Lo que ha sido descrito aquí, es todo lo que el protocolo SPI proporciona. No hay ninguna conformidad de conexión, o reconocimiento o cualquier norma de comunicación avanzada asociada con ello. Así aquellos tendrían que ser puestos en práctica sobre esta capa de ser necesario. SPI es un fabuloso y pequeño protocolo, para aquellas situaciones que requieran una simple comunicación entre dos hosts.

2.13.2 Protocolo I2C.

La interfaz de comunicación I2C es una comunicación serie, por tanto sólo usa dos lineas, "Serial Data Line" (SDA) y "Serial Clock" (SCL). Los voltajes usados son +5 V o +3.3V. En nuestro caso usaremos 3.3V puesto que es la salida que proporciona el microcontrolador.

El diseño de referencia de I2C usa 7 bits para el espacio de direcciones, con 16 direcciones reservadas, lo que hace un total de 112 nodos conectados al mismo bus. Las velocidades comunes del bus de datos son de 100 kbit/s en modo standard y de 10kbit/s en modo de baja velocidad. Hay revisiones más recientes del bus I2C que pueden conseguir velocidades de hasta 3.4 Mbit/s en modo de alta velocidad. Sin embargo puesto que en nuestro caso solo se enviaran 42 Bytes para la lectura o escritura de los registros del RTC, la velocidad no es apenas relevante. En nuestro caso configuraremos el bus del puerto I2C de nuestro microcontrolador a 100 kHz.

Existirán dos tipos de nodos dentro del bus I2C:

- Nodo maestro: Nuestro microcontrolador.
- Nodo esclavo: El Reloj en tiempo real.

Podrán haber varios nodos maestros en el bus y además, los roles de nodos maestros y esclavos pueden cambiar durante la comunicación entre dispositivos. Hay cuatro posibles modos de operación, a pesar de que la mayoría de dispositivos usan un solo rol y sus dos modos de operación:

- El Nodo maestro trasmite : El nodo maestro envía datos al esclavo.
- El Nodo maestro recibe — El nodo maestro recibe datos del esclavo.
- El Nodo esclavo transmite — El nodo esclavo trasmite datos al maestro.
- El Nodo esclavo recibe— El nodo esclavo recibe datos del maestro.

El maestro entra inicialmente en modo de transmisión maestra enviando un bit de start seguido de 7 bits que indican la dirección del esclavo con el que se desea comunicar, seguido a continuación de un bit que indica si el maestro desea leer o escribir en el esclavo.

Si el nodo esclavo se encuentra conectado al maestro, responderá con un bit de ACK(activo a bajo) para esa dirección. El maestro a continuación continua en modo de transmisión o recepción (en función del bit enviado tras la dirección) y el esclavo continua en el modo contrario al maestro.

La dirección y los bits de datos se envían con el bit más significativo primero. El bit de start se indica con una transición de alta a baja de la línea SDA, permaneciendo la línea SCL en alta. El bit de stop es indicado con una transición de baja a alta de la línea SDA, permaneciendo la línea SCL en alta.

Si el nodo maestro desea escribir en el esclavo, entonces manda repetidamente la dirección del registro a leer al cual el esclavo responderá con sendos ACK´s. (en este caso , el nodo maestro estaría en modo de transmisión maestra y el esclavo en modo recepción esclava). Si el nodo maestro desea leer del esclavo, entonces el maestro recibirá repetidamente un byte del esclavo, al cual el maestro responderá con sendos ACK´s. (en este caso , el nodo maestro estaría en modo de recepción maestra y el esclavo en modo transmisión esclava).

El nodo maestro finalizará la transmisión con un bit de stop, o enviará de nuevo un bit de start si desea obtener el control del bus para otra transferencia.

Bibliografía:

[1] G. Huang. "Casting the Wireless Sensor Net". July/August 2003. MIT Technology Review, pp. 51-56.

[2] Smart Dust, University of California, Berkeley: Disponible on-line en: "<http://robotics.eecs.berkeley.edu/~pister/SmartDust/>"

[3] Vellidis, G., Tucker, M., Perry, C., Kvien, C., Bednarz, C., 2008. "A real-time wireless smart sensor array for scheduling irrigation". Comput. Electron. Agric. 61, 44–50.

[4] Ryan Dickey,Timothy Franklin,Jake Harmon2004."Nuclear, Biological and Chemical (NBC) Communications network". Systems and Information Engineering Design Symposium

[5] CodeBlue, University of Harvard. Harvard Sensor Network Lab. http://fiji.eecs.harvard.edu/CodeBlue

[6] IEEE, Institute of Electrical and Electronics Engineers, 2006. "IEEE Standard for Information technology-Telecomunications and information exchange between systems – Local and metropolitan area networks – Specific requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-RateWireless Personal Area Networks (WPANs)". IEEE Standard 802.15.4-2006, Institute of Electrical and Electronics Engineers, New York.

[7] Baronti, P., Pillai, P., Chook, V.W., Chessa, S., Gotta, A., Fu, Y.F., 2007. "Wireless Sensor Networks: a survey on the state of the art and the 802.15.4 and ZigBee standards". Comput. Commun. 30, 1655–1695.

[8] Drew Gislason. "ZigBee Wireless Networking". Newnes. 2008. ISBN-10: 9780750685979.

[9] ZigBee Alliance, especificación de la revisión ZigBee 2007. Disponible on-line en: http://www.zigbee.org.

[10] King, B.A., Wall, R.W., Wall, L.R., 2000. "Supervisory Control and Data Acquisition System for Closed-Loop Center Pivot Irrigation". ASAE Technical Paper No. 00-2020. The American Society of Agriculture Engineers, St. Joseph, Michigan, USA.

[11]HART.TheLogicalWirelessSolution:<http://www.hartcomm2.org/hart_protocol/wireless_hart/hart_the_logical_solution.html>.

[12]Draft standard: What's in the April'07 WirelessHART specification: <http://www.controleng.com/article/CA6427951.html>.

[13]ISA100.11a:<http://www.isa.org/MSTemplate.cfm?MicrositeID=1134&CommitteeID=689>.

[14] 6LoWPAN, <http://6lowpan.net/>.

[15] G. Mulligan, L.W. Group. "The 6LoWPAN architecture". Proceedings of the EmNets, Cork, Ireland, 2007.

[16] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler. "Transmission of IPv6 packets over IEEE 802.15.4 networks". RFC 4944 (2007).

[17] IEEE Standard 802.15.3. "Wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless person area networks (WPANs)". September 2003.

[18] Wibree, <http://www.wibree.com/>.

[19]Crossbow :< www.crossbow.com>

[20]The intel mote, disponible on-line en: http://www.ieee.or.com/Archive/intel_mote.htm

[21] Zolertia :<http://www.zolertia.com/ti>

[22] Libelium Company, disponible on-line en: <http://www.libelium.com/>

[23]Texas Instruments , disponible on-line en: <www.ti.com>

[24] Digi Wireless Solutions, disponible on-line en: <http://www.digi.com/>

[25] Tinyos , disponible online en: <http://www.tinyos.net/>

[26] Microsoft .NET Micro Framewrok, disponible en:< http://www.microsoft.com/en-us/netmf/default.aspx>

[27] MantisOS, disponible on-line en: <http://mantisos.org/index/tiki-index.php.html>

[28] eCos , disponible on-line en: <http://ecos.sourceware.org/>

[29] Mircrium, disponible on-line en: <http://micrium.com/page/home>

[30] Contiki Operative System, disponible on-line en: <*www.sics.se/contiki*/>

[31] SimpleOS, disponible on-line en*:* <http://sos.enix.org/fr/PagePrincipale>

[32] NanoRK, disponible on-line en: <www.nanork.org>

[33] Jennic, disponible on-line en:<*http://www.jennic.com*/>

Capítulo 3

Arquitectura del sistema.

En el siguiente apartado se mostrará la arquitectura del sistema realizado.

En primer lugar se realizará un análisis de los posibles sistemas operativos a utilizar en nuestro sistema, haciendo hincapié en sus ventajas e inconvenientes. A continuación, se elegirá uno de ellos, en el cual se pretenderá dar soporte a nuestra plataforma, intentando conseguir su total funcionalidad.

Por último, se explicará la arquitectura hardware del nodo MEWIN desarrollado en el seno del DSIE, en la UPCT.

3.1 Descripción software

3.1.1 Elección de un sistema operativo en el que dar soporte a la plataforma

En el capítulo anterior, se han expuesto los sistemas operativos más comunes en el panorama actual de las redes de sensores inalámbricas. Tras analizar las características de cada uno de ellos, se ha decido optar por un sistema operativo de libre distribución en contraposición a otras alternativas como una implementación de la pila ZigBee de Texas Instruments, debido a que al ser una arquitectura totalmente abierta, permite una mayor customización del sistema, y porque ofrece diversas alternativas de comunicación entre los motes, que como se verá en el capítulo siguiente, ofrecen una mayor versatilidad. Además, puesto que se pretende que un usuario sin conocimientos de la arquitectura utilizada sea capaz de programar y trabajar con el dispositivo, un sistema operativo como Contiki o TinyOS es mucho más intuitivo de usar para el usuario final que cualquiera de las implementaciones de la pila ZigBee proporcionadas por los diferentes fabricantes de circuitos integrados, ya que, en la mayoría de los casos, para adaptar el dispositivo a los sensores sobre los que trabajará, será necesario trabajar a muy bajo nivel.

Como ya se ha comentado en el apartado anterior, existen diversos sistemas operativos de libre distribución para dispositivos embebidos tales como : TinyOS, Contiki, PalOS y SOS.

El más conocido y más utilizado en el ámbito de las redes de sensores es probablemente TinyOS, sin embargo, en los últimos años, el sistema operativo Contiki se ha alzado como una alternativa a tener en cuenta. El principal inconveniente de Contiki es que debido a que no lleva demasiado tiempo en el mercado, no existe por tanto demasiada documentación al respecto y los primeros pasos con este sistema operativo pueden ser complicados.

En la difícil elección de seleccionar un sistema operativo sobre el que dar soporte a la plataforma MEWiN, se deben considerar las ventajas e inconvenientes de cada uno de ellos. Es complicado elegir uno de ellos basándose simplemente en sus características técnicas ya que como se vió en los primeros capítulos, las características de ambos son muy similares a pesar de que están implementados de formas diferentes, sin embargo, en este sentido, se puede decir que Contiki presenta la ventaja sobre TinyOS-2.x, de permitir programación multihilo de forma realmente sencilla gracias a los PROTOTHREADS, así como la carga dinámica de código, permitiendo que la implementación en Contiki de programas más complejos , por contra, la documentación existente para TinyOS en la red es cuantiosa, todo lo contrario que ocurre con Contiki.

A pesar de ello, éstas características citadas no son suficientes como para decantarse por uno de ellos directamente. Por tanto, se ha decido analizar las implementaciones de comunicación inalámbrica de la que disponen ContikiOS y TinyOS, más concretamente en la implementación de la pila 6LOWPAN . En función de las ventajas que proporcionan cada uno de ellos se realizará la elección del SO.

3.1.2 Implementaciones de 6LoWPAN

En la actualidad existen tres implementaciones de 6LoWPAN muy reconocidas. Dos de ellas desarrolladas para el sistema operativo TinyOS-2.x y la restante desarrollada para el Sistema Operativo Contiki.

La primera implementación es conocida como 6lowpancli, fue lanzada en 2007 y actualmente está integrada como una aplicación nativa en TinyOS-2.x. 6lowpancli soporta compresión de cabeceras, fragmentación y direccionado. Respecto a los protocolos de alto nivel, 6lowpancli soporta UDP, llevándolo a cabo con el formato HC_UDP y mensajes ICMPv6. Por lo tanto es posible realizar un ping a los motes y usar la comunicación UDP desde cualquier punto de internet hasta un pequeño nodo sensor. Para realizar de puente entre la red de sensores y la red convencional, TinyOS-2.x provee de una pequeña aplicación para emular un túnel IPv6 a través del puerto USB por el que se puede conectar el nodo sensor. El principal inconveniente de 6lowpancli, es que es totalmente estático y necesita ser configurado manualmente. No soporta ningún mecanismo de descubrimiento de nodos vecinos (Neighbor Discovery Protocol) o movilidad. Además la configuración de red en malla (mesh networks) no está soportada y cuando un paquete con dirección de destino diferente a la del nodo es recibido, éste es omitido.

La segunda implementación de 6LoWPAN desarrollada para TinyOS-2.x se denomina BLIP (Berkeley Low-power IP stack). En este caso el código no está incluido nativamente en TinyOS-2.x pero puede encontrarse en las contribuciones de la Universidad de Berkeley. BLIP implementa las características básicas definidas en el RFC4944, es decir: compresión de cabeceras, fragmentación y direccionamiento. Soporte de paquetes ICMPv6 y UDP. Además de eso, la última versión añade el primer prototipo de la pila TCP, que se encuentra todavía en fase experimental. Adicionalmente, BLIP incorpora una versión "ligera" del protocolo de descubrimiento de nodos vecinos (Neighbor Discovery Protocol) , siendo capaz de configurar un dirección de enlace local o global en los nodos vecinos, en función de si se ha recibido o no una trama de aviso procedente del router. Al igual que 6lowpancli, BLIP incluye una aplicación para crear un túnel IPv6 para conectar una red convencional a la red inalámbrica de sensores. Además BLIP, al contrario que 6lowpancli soporta redes en malla (mesh networks) tal y como se define en el RFC4944 también llamadas "mesh under". "Mesh under" indica que desde el punto de vista de la capa IP, todos los nodos están a un salto de distancia, ya que los saltos múltiples son llevados a cabo por la capa MAC.

SicslowPAN es la primera implementación de 6LoWPAN desarrollada para el Sistema Operativo Contiki. SICSlowPAN está basada en el RFC4944, e implementa mecanismos de fragmentación y direccionamiento. Además añade un nuevo mecanismo de compresión de cabeceras. SICSlowPAN no implementa las redes "Mesh under" pero las soporta utilizando otras técnicas de ruteado.

Por otra parte SICSlowPAN se encuentra entra la capa IPv6 y la capa MAC. Por tanto, cuando la capa MAC recibe un paquete de IPv6, llama a la capa SICSlowPAN para adaptar el paquete desde la capa MAC a la capa IP. Además, cuando la capa uIPv6 necesita enviar un paquete, también utiliza la capa SICSlowPAN para conformar el paquete.

El Sistema Operativo Contiki implementa por defecto un protocolo no-IP en la capa MAC, llamado RIME. Las pilas uIPv4 y uIPv6 han sido añadida recientemente. La pila uIPv6 es independiente de cualquiera de las capas inferiores, siendo posible enviar datos a través del estandard 802.15.4, 802.11 o 6lowPAN. La pila uIPv6 soporta paquetes ICMPv6, implementa el protocolo de descubrimiento de vecinos (Neighbour Discovery) y soporta tanto UDP como TCP.

La tabla siguiente muestra una comparativa de las tres implementaciones de 6LOWPAN descritas anteriormente.

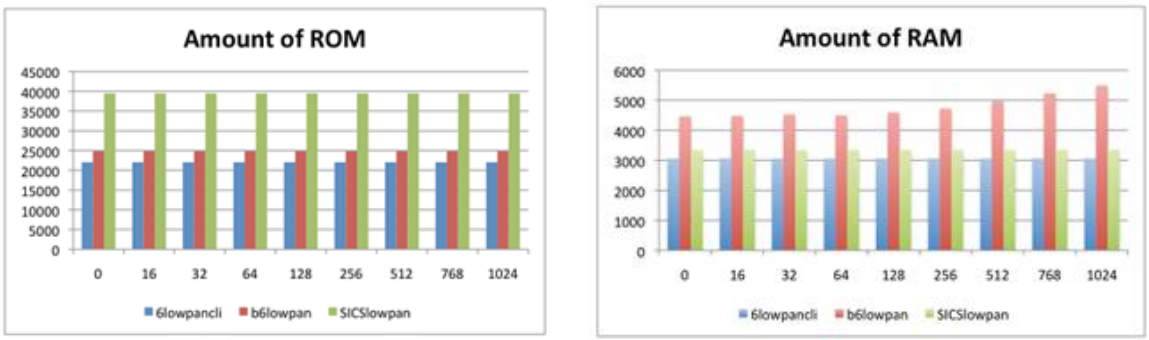
Tabla 3-1: Tabla resumen de características de las implementaciones de 6LoWPAN.

Característica	**6lowpancli**	**BLIP**	**Sicslowpan**
OS	*Tinyos-2.x*	*TinyOS-2.x*	*ContikiOS*
ICMPv6 Echo	*SI*	*SI*	*SI*
UDP	*SI*	*SI*	*SI*
TCP	*NO*	*PROTOTIPO*	*SI*
Neighbor Discovery	*NO*	*SI*	*SI*
Mesh Header	*NO*	*SI*	*SI*
Route Over	*NO*	*SI*	*SI*

3.1.3 Evaluación de las implementaciones de 6LoWPAN

Existen diversos artículos de investigación que analizan las prestaciones de cada una de las implementaciones. En este caso, en [3] se realizan una serie de pruebas que se analizarán a continuación.

La primera prueba consiste en programar el mismo dispositivo (mote TelosB) con las tres diferentes implementaciones de 6LoWPAN, y medir las cantidades de memoria (RAM y ROM) que ocupan cada una de las implementaciones en función del tamaño del paquete que se envía. La figuras 3-1 muestra los resultados de esta comparativa.



a) **b)**

Figura 3-1: (a)Cantidades de ROM y (b)RAM consumida por el dispositivo de las implementaciones de 6LoWPAN.

Una vez programados los dispositivos se realizó la prueba de medir el retardo desde que el emisor empieza a enviar el paquete IP hasta que es recibido en su totalidad por el receptor.

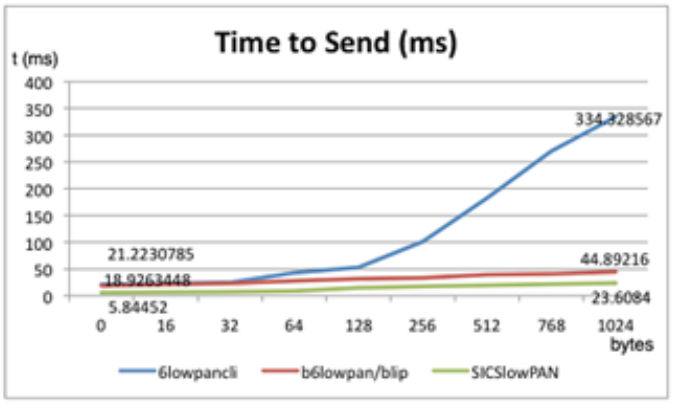


Figura 3-2: Tiempo de envío de un paquete de datos para cada una de las implementaciones.

Observando los resultados obtenidos en las Figuras 3-1 y 3-2, se puede concluir que SICSlowpan requiere una mayor cantidad de ROM y que BLIP requiere una mayor cantidad de RAM. Respecto al otro test realizado, se puede concluir como los tiempos de envío en SICSlowpan y BLIP son similares, siendo significativamente menor en SICSlowpan para todos los tamaños de paquetes de datos IP enviados. Sin embargo, tanto la medida de cantidad de memoria necesitada como el tiempo de envío no son tan concluyentes como para decantarse por uno de ellos. Por tanto se deberá evaluar todas los características comentadas anteriormente para elegir una de las implementaciones.

3.1.4 Elección de una implementación de 6LoWPAN

A la hora de elegir una implementación de 6LoWPAN se deben tener en cuenta, además de las características que proporciona cada una de las implementaciones de 6LoWPAN, las características del sistema operativo donde se encuentran alojadas cada una de las pilas de protocolos. En este sentido, se ha considerado que ContikiOS es más ventajoso que TinyOS-2.x, ya que ContikiOS permite la programación multihilo, así como la carga dinámica de código. Éstas dos características permiten la implementación en ContikiOS de programas más complejos y permite cargar código sin necesidad de recargar todo el sistema operativo.

Con respecto a las características de las implementaciones de 6LoWPAN, la pila SICSLoWPAN cuenta con la desventaja con respecto a BLIP (B6lowpan) de no implementar las funciones relacionadas con las redes "Mesh under". Sin embargo, el

ruteado se realiza a través de otro tipo de técnicas, por lo que este problema queda solventado. Por otro lado, se debe tener en cuenta que SICSLoWPAN es la única pila de las tres citadas que cuenta con el Logo IPv6 Ready, que certifica su robustez y se espera que continúe mejorando.

Por tanto, se ha concluido que la mejor elección es trabajar con ContikiOS y su implementación SICSlowPAN, por todas las ventajas citadas anteriormente.

3.1.5 Ejemplos de funcionamiento de la pilas uIPv4 e uIPv6 de Contiki en un dispositivo comercial (TelosB)

En este apartado se pretende demostrar la potencia del uso del protocolo IP en Redes de Sensores Inalámbricas. Para ello, se utilizará el sistema operativo Contiki, con el cual se realizarán dos ejemplos prácticos.

3.1.5.1 Ejemplo 1: Servidor WEB compatible con AJAX en un mote comercial.

Este ejemplo muestra como un mote comercial usado habitualmente en redes de sensores(TelosB), es capaz de actuar de servidor Web compatible con AJAX (carga dinámica de código) , de manera que es posible acceder al dispositivo desde un navegador Web. Para ello sólo necesitaremos: El mote programado con Contiki y un ordenador acceso a Internet.

De esta forma, simplemente introduciendo la dirección IP del mote en el navegador Web, se podrán comprobar los datos tomados por los sensores en cualquier momento.

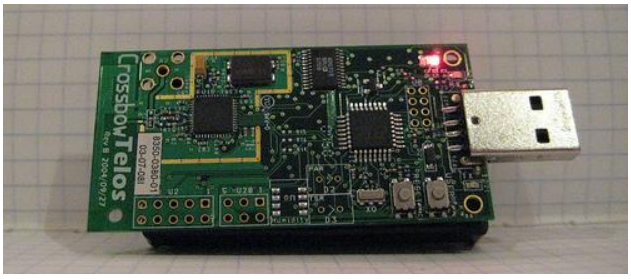


Figura 3-3: Mote comercial: TelosB

En primer lugar se programará un TelosB (ver Figura 3-3) con el código de ejemplo disponible en: "examples/sky-ip/sky-webserver", dentro del directorio principal de Contiki.

Tras programar el dispositivo, se lanza la aplicación linslip, la cual asigna al puerto USB en el que está conectado el mote, una dirección IP. Para comprobar que el mote es accesible, se realiza un ping al dispositivo.

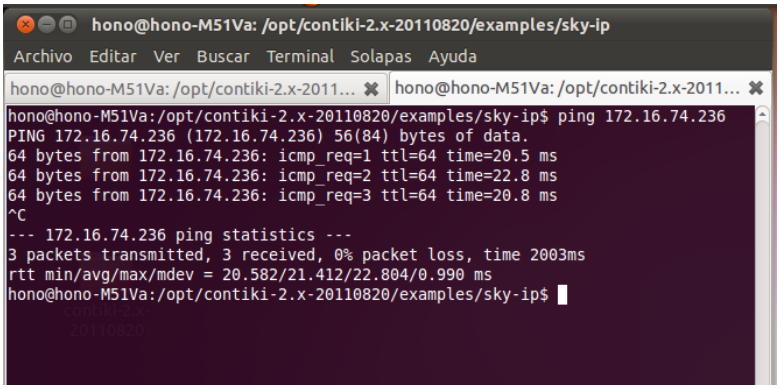


Figura 3-4:Resultado del comando ping a la IP del Mote

Como se puede observar en la Figura 3-4, el comando ping se ha realizado con éxito. A continuación se introducirá la IP en un navegador Web y observaremos como el dispositivo muestra una página Web en la cual se pueden acceder los datos e incluso los carga de forma dinámica como si de un servidor Web se tratara(ver Figura 3-5).

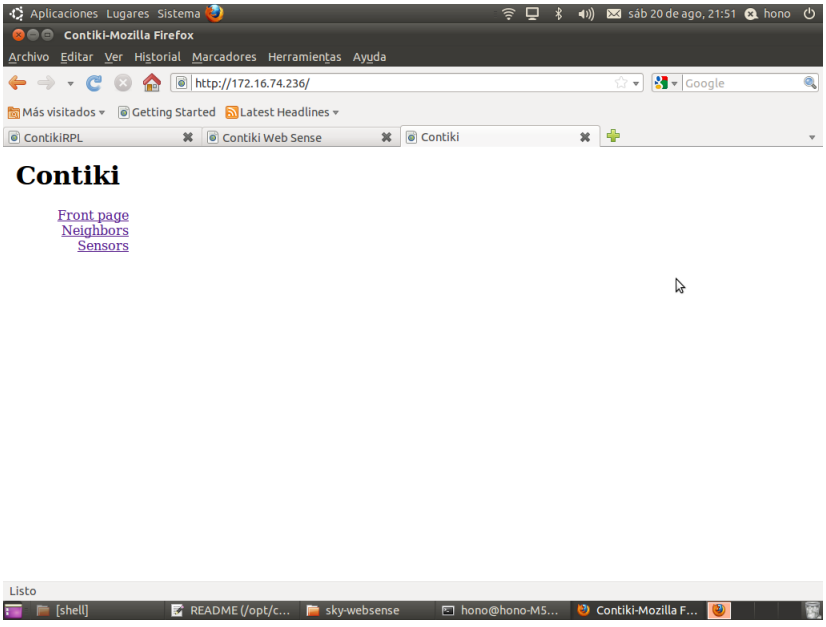


Figura 3-5:Página de inicio mostrada por el dispositvo

Si se hace "click" sobre Sensors, se obtienen los datos de los sensores y consumo del dispositivo. (ver Figura 3-6).

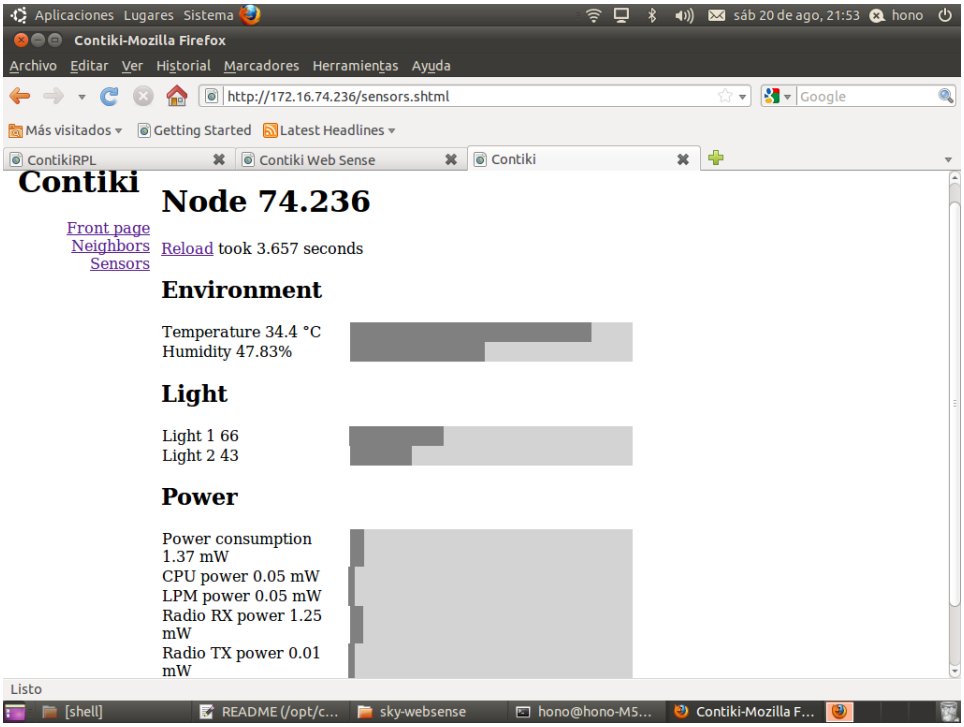


Figura 3-6: Datos de los sensores y consumo del dispositivo

3.1.5.2 Ejemplo 2: Servidor WEB usando IPv6 en un mote comercial

Este ejemplo muestra como un mote comercial usado habitualmente en redes de sensores(TelosB), es capaz de actuar de servidor Web usando el protocolo IPv6 de manera que es posible acceder al dispositivo desde un navegador Web. Para ello sólo necesitaremos: El mote programado con Contiki y un ordenador acceso a Internet.

Además, se mostrará como usando un único mote conectador a Internet, es posible acceder al resto de motes de nuestra WSN y leer los datos obtenidos por sus sensores. Este hecho es muy común en las redes de sensores como ya se comentó en Capítulo 2, se suele utilizar un nodo Gateway que provea de acceso a Internet al resto de motes de la Red Inalámbrica de Sensores. Este mote actuará como sumidero de información.

En primer lugar se programará un TelosB (ver Figura 3-3) con el código de ejemplo disponible en: "examples/ipv6/sky-websense", dentro del directorio principal de Contiki.

A continuación, se programara el mote que actuará como router con la aplicación "rpl-border-router". A continuación se establecerá la conexión con el router, obteniendo la ip del mismo. Ya sólo resta introducir la IP en el navegador y observar (ver Figura 3-7) como se muestran las direcciones de los nodos de nuestra red.

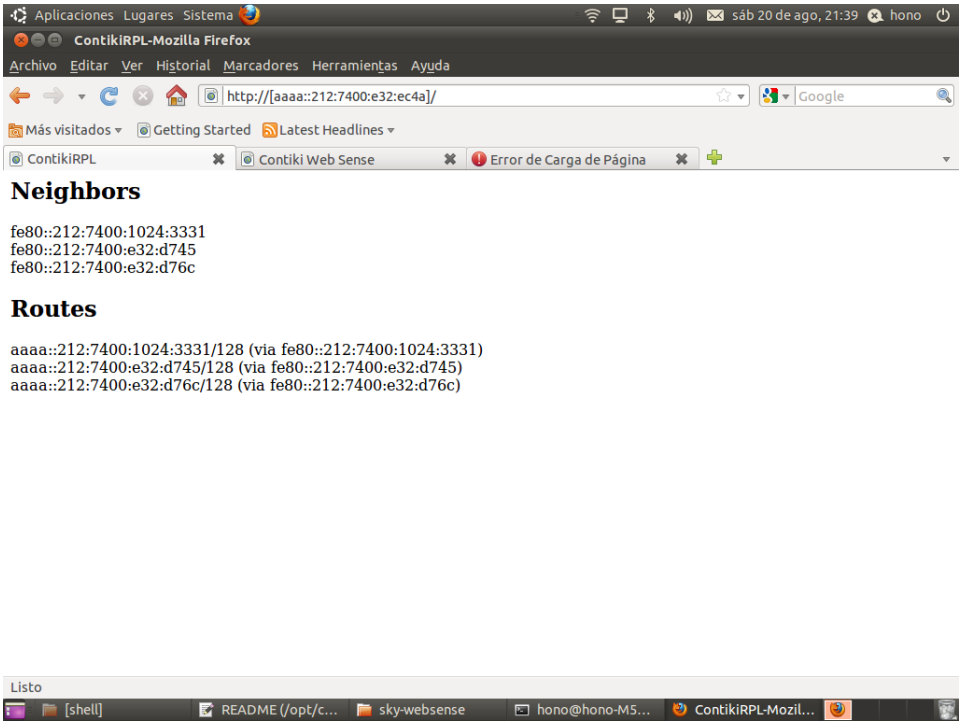


Figura 3-7:Detalle de los 3 sensores de nuestra red, mostrados al acceder a la IP del router en un navegador Web

A continuación, sólo restará introducir en el navegador cualquiera de las direcciones de los nodos, para obtener el valor de los sensores de temperatura y luminosidad (ver Figura 3-8).

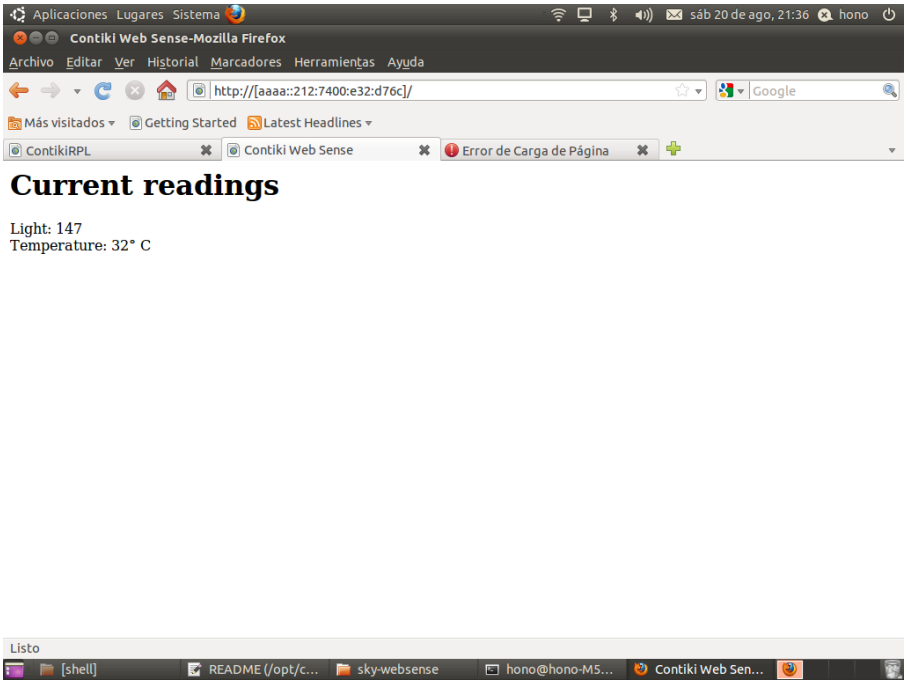


Figura 3-8: Acceso a los datos de los sensores usando el protocolo IPv6.

3.1.6 Descripción del sistema operativo Contiki

Contiki es un pequeño sistema operativo de código abierto, altamente portable y multitarea, desarrollado para uso en pequeños sistemas, desde ordenadores de 8-bits a sistemas empotrados sobre microcontroladores, incluyendo nodos de redes de sensores. El nombre Contiki viene de la famosa balsa Kon-Tiki de Thor Heyerdahl.

A pesar de incluir multitarea y una pila TCP/IP, Contiki sólo requiere varios kilobytes de código y unos cientos de bytes de RAM. Un sistema totalmente completo con una GUI requiere aproximadamente 30 kilobytes de RAM.

El núcleo básico y la mayor parte de las funciones principales fueron desarrollados por Adam Dunkels en el grupo de sistemas de redes empotradas en el instituto sueco de ciencias computacionales.

Contiki fue diseñado para sistemas empotrados con poca cantidad de memoria. Una configuración típica de Contiki es 2 kilobytes de RAM y 40 kilobytes de ROM. Contiki consiste en un núcleo orientado a eventos, el cual hace uso de protothreads, sobre el cual los programas son cargados y descargados dinámicamente. También soporta multihilado apropiativo opcional por proceso, comunicación entre procesos mediante paso de mensajes a través de eventos, al igual que un subsistema GUI opcional, el cual puede usar un soporte directo de gráficos para terminales locales, terminales virtuales en red mediante VNC o sobre Telnet.

Contiki funciona en una variedad de plataformas, desde microcontroladores empotrados, como MSP430 de Texas Instruments y el AVR, a viejas computadores domésticas. El tamaño del código está en el orden de los kilobytes y el uso de la memoria puede configurarse para que sea de sólo unas decenas de bytes.

Contiki proporciona una pila de comunicaciones IP, tanto para IPv4 como para IPv6. Además la pila uIPv6 de Contiki cuenta con el logo "IPv6 Ready", lo que garantiza su robustez.

Muchas de los mecanismos e ideas usados en Contiki han sido ampliamente adoptados en la industria. La pila uIP , lanzada originalmente en 2001, es usada a día de hoy por cientos de compañias en sus satélites y equipamientos industriales. Contiki y uIP son reconocidos por la popular aplicación de escaneo de redes de datos NMAP. Los protothreads de Contiki, lanzados inicialmente en 2005, han sido utilizados en un gran número de dispositivos embebidos, desde decodificadores de televisión digital hasta sensores de vibración inalámbricos.

Contiki introdució la idea de usar la comunicación IP en redes de sensores inalámbricas. Esto llevó más tarde a un estandar IETF y la IPSO Aliance, una Alianza de la industria internacional. La revista TIME listó "el Internet de las cosas" y la Alianza IPSO como la trigésima innovación más importante de 2008.

Contiki dispone de dos pilas de comunicaciones: uIP y Rime. Como ya se ha comentado, uIP es una pequeña pila que implementa el protocolo TCP/IP y hace posible al sistema operativo comunicarse a través de Internet. Rime es una pequeña pila de comunicación diseñada para radiotransmisores de baja potencia. Rime proporciona una amplia gama de primitivas de comunicación, tales como "best-effort local area broadcast" o "reliable multi-hop bulk data flooding".

3.1.6.1 Protothreads en Contiki

Los protothreads en Contiki son hilos de proceso ligeros y que no necesitan pila asociada, están diseñados especificamente para dispositivos con grandes restricciones de memoria tales como los sistemas usados en redes de sensores inalámbricas.

Los protothreads proporcionan una ejecución lineal del código para sistemas dirigidos por eventos implementados en C. Los protothreads pueden ser usados sin necesidad de un sistema operativo en tiempo real (RTOS,"Real Time Operative System"). El propósito de los protothreads es implementar un control de flujo secuencial sin necesidad de usar complejas máquinas de estados o multihilos complejos. Los protothreads proporcionan bloqueo condicional dentro de las funciones en C.

La ventaja de los protothreads sobre una programación basada unicamente en eventos, es que los protothreads proporcionan una estructura de código secuencial que permiten el bloqueo de funciones. En la programación basada unicamente en eventos, el bloqueo de las funciones se debe implementar dividiendo la función en dos partes de forma manual, una parte antes de la llamada bloqueante y la otra tras la llamada bloqueante. Esto complica el uso de las llamadas de control, tales como sentencias if() condicionales y bucles while(). La ventaja de los protothreads sobre los threads es que los protothreads no requieren un pila separada. En dispositivos embebidos, el sobrecostes de albergar múltiples pilas puede consumir grandes cantidades de memoria. En contraste, cada protothreads sólo necesita entre 2 y 12 bytes de estado dependiendo de la arquitectura del sistema.

Principales ventajas de los protothreads:

- No hay código específico para cada máquina, los protothreads están escritos en C.
- Bajo consumo de memoria por cada protothread (entre 2 y 12 bytes).
- Pueden ser usados con o sin sistema operativo.
- Proporcionan capacidades de bloqueo sin necesidad de máquinas de estados.

Ejemplos de utilización:

- Dispositivos con grandes restricciones de memoria
- Pilas de protocolos dirigidas por eventos.
- Dispositivos embebidos.
- Redes de sensores.

3.1.6.2 Organización del sistema operativo Contiki

El sistema operativo Contiki consta de la siguiente estructura:

Figura 3-9: Estructura del sistema operativo Contiki.

A continuación se detallarán las carpetas más importantes:

- ***Core***: En esta carpeta se encuentra el grueso del sistema operativo: Implementación de los módulos de radio, Timers, watchdog, funciones de lectura/escritura, etc.
- ***Cpu:*** En este directorio se encuentran las diferentes implementaciones para cada uno de los microcontroladores soportados por el sistema operativo.
- ***Doc:*** Aquí encontramos la documentación del sistema operativo.
- ***Examples:*** Aqui se encuentran diversos ejemplos divididos en plataformas, es decir, dentro de la carpeta encontraremos sucesivas subcarpetas con el nombre de

las plataformas dentro de las cuales se encuentran diferentes ejemplos funcionales, creados para comprender el funcionamiento

- ***Platform:*** En este directorio encontramos las bibliotecas específicas para cada plataforma, por ejemplo en nuestro caso, puesto que la placa MEWiN consta de una tarjeta SD y un reloj RTC, en esta carpeta deberemos crear las bibliotecas que permitan al usuario final utilizarlos a través de funciones sencillas que le ayuden a abstraerse del hardware .

3.2 Descripción Hardware (Dispositivo MEWiN)

La placa MEWiN es un dispositivo multifuncional desarrollado en el departamento de tecnología electrónica de la UPCT. Su función es proporcionar capacidad de procesamiento, comunicación inalámbrica y almacenamiento a los dispositivos sensores que se conectarán a él a través de los diferentes puertos de expansión.

En este tipo de dispositivos destinados a redes inalámbricas de sensores, es muy importante que el consumo de todo el conjunto sea el mínimo posible. Para ello la placa dispone de un microcontrolador de la gama MSP430 de Texas Instruments. Este microcontrolador está diseñado especificamente por TI para llevar a cabo labores que precisan de un consumo muy bajo.

Por otra parte, es necesario que el dispositivo conste de una interfaz inalámbrica que le permita comunicarse con el resto de dispositivos que conforman la red (Motes). Esta comunicación se lleva a cabo a través del modulo de radio CC2520 conectado al microcontrolador a través del bus SPI . Este módulo de radio trabaja en la banda de 2.4GHz siguiendo el estandard IEEE 802.15.4.

Para conseguir alcances mayores, el módulo de radio está conectado a su vez a un dispositivo amplificador cuya misión es aumentar la potencia de transmisión y mejorar la sensibilidad del sistema para aumentar el rango del dispositivo.

El dispositivo MEWiN también cuenta con un reloj en tiempo real. Este elemento es muy importante, ya que permite conocer el momento exacto en que se han muestreado cada uno de los datos de los sensores. Este reloj en tiempo real, dispone de una batería de backup que le permite seguir funcionando cuando la placa MEWiN no esté conectada a ninguna alimentación

Además, el dispositivo dispone de una tarjeta SD que le permite almacenar los datos tomados de los diferentes sensores conectados a él.

Los 4 leds de estado, permiten al programador o usuario saber el estado del sistema(despierto, leyendo datos de los sensores, escribiendo en la SD, etc.) en cada momento.

El dispositivo dispone de dos interfaces RS232, una de ellas se externaliza a través de un puerto de expansión(ver Figura 3-4) y permite al dispositivo establecer una comunicación bidireccional con un PC a través del puerto serie. La otra interfaz se direcciona a través de los puertos de expansión y está destinada a ser usada por los sensores conectados al dispositivo que usen este protocolo de comunicación.

En caso de que sea necesario reiniciar el dispositivo, la placa cuenta con un pulsador para resetearla instantáneamente.

Por último y no menos importante, MEWiN cuenta con un dispositivo capaz de cargar la batería cuando este se conecta a la alimentación, de manera, que permite cargar la batería sin necesidad de extraerla del dispositivo, con las ventajas que ello conlleva.

A continuación se explicarán los diferentes componentes que conforman la placa MEWiN:

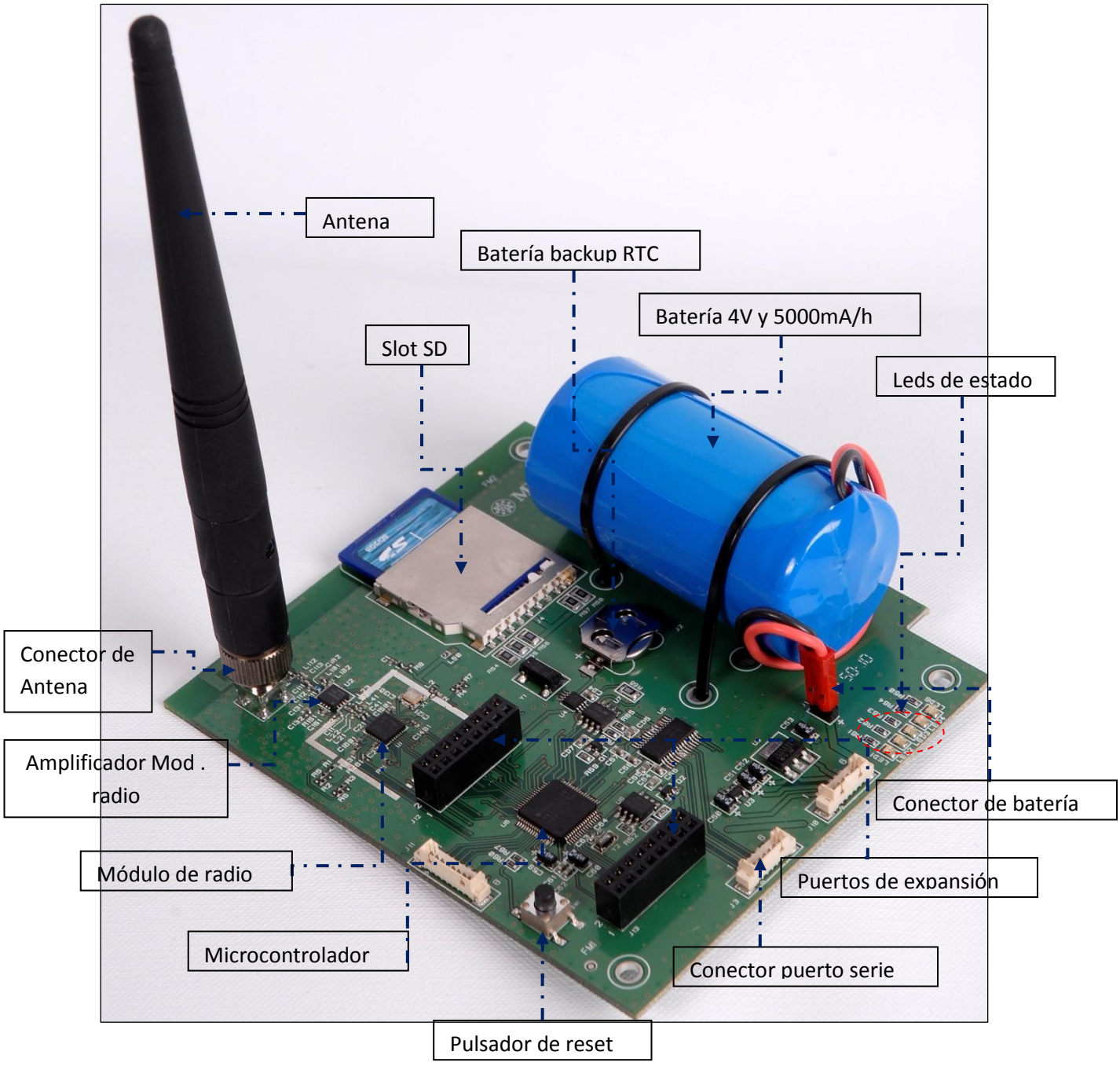


Figura 3-10: Vista en perspectiva del dispositivo MEWiN

3.2.1 Microcontrolador MSP430-F2618.

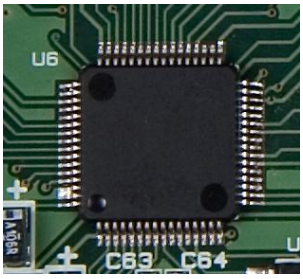


Figura 3-11: Microcontrolador MSP430F2618 montado en placa.

La familia MSP430(ver Figura 3-11) de Texas Instruments, consiste en una familia de microcontroladores de muy bajo consumo, que disponen de una gran cantidad de periféricos pensados para una gran cantidad de aplicaciones. La arquitectura, combinada con cinco tipos de modos de bajo consumo está optimizada para lograr un alto tiempo de vida de la bateria en aplicaciones portátiles donde es necesario medir diferentes parámetros. El dispositivo cuenta con una potente CPU RISC de 16 bits y registros de 16 bits.

El oscilador controlado digitalmente ("DCO", Digital Controlled Oscillator) permite al dispostivo despertarse desde los modos de bajo consumo hasta el modo activo en menos de 1 microsegundo.

La familia MSP430F261x consta de dos timers de 16 bits, un conversor A/D rápido de 12 bits, un comparador, un conversor dual de 12 bits D/A, cuatro interfaces universales de comunicación serie ("USCI", Universal Serial Communication), DMA y hasta 64 pines de entrada/salida.

El dispositivo consta de 116KB + 256B de Memoria Flash y 8 KB de Memoria RAM. El rango de voltajes necesario para su correcto funcionamiento se encuentra entre 1.8V y 3.6V. Los consumos según los modos de funcionamiento son:

Modo activo: 365 μA a 1 MHz , 2.2 V.

Modo Standby (VLO): 0.5 μA.

Modo Off (Retención en RAM): 0.1 μA

La programación del dispositivo se realiza a través de la interfaz JTAG, a través del dispositivo MSP-FET430UIF.

3.2.2 Módulo de radio CC2520

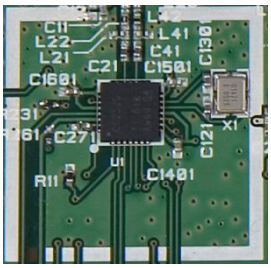


Figura 3-12: Módulo de radio cc2520 montado en placa.

El módulo CC2520(ver Figura 3-12) es la segunda generación ZigBee / IEEE 802.15.4 transceiver para la banda de 2.4GHz en la banda ISM. Este chip habilita a diferentes tipos de aplicaciones ofreciendo una excelente calidad del enlace de transmisión radio, puede operar hasta a 125°C y operando a bajo voltaje.

Además, el módulo de radio CC2520 proporciona soporte para "frame handling", "data buffering", "burst transmissions", "data encryption", "data authentication", "clear channel assessment", indicación de calidad de enlace e información de los tiempos de la trama. Estas características reducen la carga en el controlador host. En un sistema típico, el CC2520 debe ser usado en conjunto con un microcontrolador y una serie de componentes pasivos (resistencias, bobinas, condensadores etc).

El dispositivo proporciona una excelente calidad de enlace (103 dB) y un rango de 400 m en linea de visión directa. El rando de temperaturas de operación se encuentra entre –40°C to +125°C. El rango de voltajes de operación se encuentra entre 1.8 y 3.8 V.

Otra de las características importantes es la capacidad para utilizar encriptación AES-128 y el modo de compatibilidad con el módulo CC2420 de TI.

El dispositivo dispone de una antena integrada en la placa, sin embargo, para conseguir distancias mayores, dispone también de un conector SMA (ver Figura 3-13), que podrá ser utilizado con antenas de diferentes características en función de las necesidades del sistema. Por ejemplo, los motes que actúe de estación base (o Coordinador en la red ZigBee)o de router necesitarán una antena con una mayor ganancia que los motes que actúen de recolectores de datos (End Devices en la red ZigBee):

Figura 3-13: Conector de antena montado en placa.

Consumo del dispositivo:

En recepción (–50 dBm) 18.5 mA .

En transmisión 33.6 mA @ +5 dBm.

3.2.3 Módulo amplificador CC2591

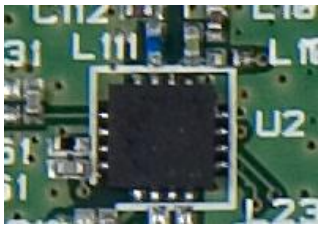


Figura 3-14: Módulo amplificador CC2591 montado en placa.

El amplificador CC2591 (ver Figura 3-7) es un componente diseñado para el uso de aplicaciones inalámbricas de bajo voltaje. Este módulo mejora la calidad del enlace a través de una amplificación de la potencia de transmisión y de un amplificador de bajo ruido ("LNA", Low Noise Amplifier) para la mejora de la sensibilidad en la recepción.

El dispositivo ofrece una mejora de la calidad en la transmisión y recepción del CC2520.Consiguiendo:

- Hasta 22 dBm de potencia de transmisión.
- 6 dB de mejora en la sensibilidad en recepción.

El consumo 112 mA de corriente en transmisión a 3V y +20 dBm de potencia de transmisión.

3.2.4 Reloj en tiempo real (RTC, "Real Time Clock")

Figura 3-15: Reloj en tiempo real DS1339 montado en placa.

El reloj en tiempo real ("RTC", Real Time Clock) DS1339 (ver Figura 3-15) de Maxim es un dispositivo de bajo consumo que permite conocer la hora y la fecha exacta, dispone además de dos alarmas programables. La dirección y los datos son transferidos en serie a través del bus I2C. El RTC proporciona información de los segundos, minutos, horas, día, fecha, mes y año. La fecha al final de los meses se ajusta automáticamente para meses con menos de 31 días, incluyendo las correcciones para años bisiestos. El reloj puede operar en formato 24 o 12 horas con indicador de AM y PM. El DS1339 dispone de un sensor integrado de alimentación que es capaz de detectar problemas de alimentación y pasar a usar la bateria de backup, manteniendo la fecha, hora y alarmas operativas.

3.2.5 Slot para tarjetas SD

Figura 3-16: Slot tarjetas SD.

La placa MEWiN dispone de un slot(ver Figura 3-16) donde se puede insertar una tarjeta SD. Este slot está conectado con una de las interfaces de entrada/salida del microcontrolador, de manera que la tarjeta y el microcontrolador se comunicarán a través del bus SPI.

La existencia de este slot es muy importante, ya que esta memoria puede ser utilizada por el programador no sólo para el logging de los datos, sino que además se puede usar como una memoria flash (de acceso más lento que la que dispone el propio microcontrolador) para guardar algún valor o variable de gran tamaño que necesite ser almacenada en la una memoria no volatil, para que siga disponible aunque se desconecte la alimentación, ya que la propia flash del microcontrolador es muy pequeña y está muy limitada.

3.2.6 Batería del sistema

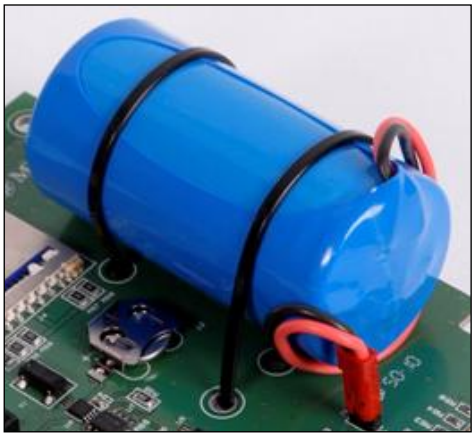


Figura 3-17: Batería y conector de batería.

El sistema está alimentado por una batería recargable de polímero de Litio de 3,7 V de tensión nominal y 5000 mAh de capacidad. Además, la placa lleva incorporado un cargador de baterías MAX1555 de manera que alimentando el dispositivo a través del conector "J10" y conectando la batería a su conector, conseguimos cargarla, apagándose automáticamente cuando esté cargada en su totalidad. Para indicar que la batería esta cargándose, se enciende el LED5 y se apagará cuando finalice la carga.

3.2.7 Leds de estado

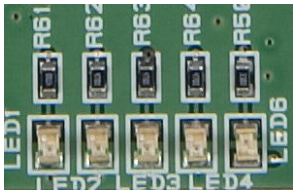


Figura 3-18: Leds de estado montados en placa.

La placa consta de cuatro diodos leds de estado accesibles por el microcontrolador y un último utilizado por el cargador de baterías, para indicar el estado de carga de la misma.

3.2.8 Pulsador de reset

Figura 3-19: Pulsador de reset montado en placa.

El dispositivo consta de un pulsador de reset. Es muy util cuando queremos reiniciar el sistema, de esta forma no hace falta desconectar la batería o el conector de alimentación.

3.2.9 Conversor de niveles RS232

Figura 3-20: Conversor de niveles RS232 montado en placa.

La placa MEWIN es capaz de comunicarse (recibir y enviar datos) a través de la interfaz RS-232. Para ello la placa dispone de una convertidor que transforma la tensión de la salida de la UART del micro a los niveles estándar usados para la comunicación con un ordenador a través del puerto serie.

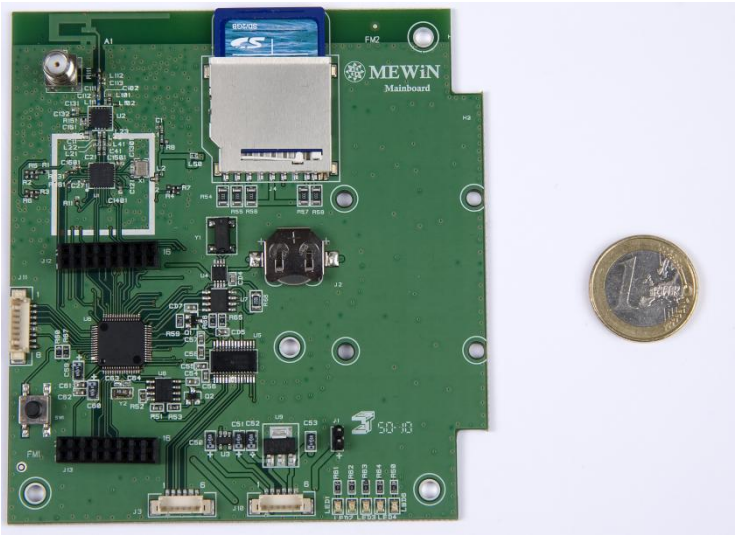


Figura 3-21: Detalle del tamaño de la placa MEWiN.

Capítulo 4

Implementación y Desarrollo Software.

En este apartado se pretende describir todos los pasos dados para conseguir el objetivo marcado previamente.

En primer lugar se hablará de Contiki, su configuración e instalación en un PC. A continuación se presentarán todos los cambios que han sido necesarios hacer para conseguir disponer de una compatibilidad parcial de la plataforma MEWiN en el sistema operativo. De manera que podamos compilar y cargar código compilado con Contiki en el módulo MEWiN.

Tras ello, se describen los pasos dados para obtener soporte en Contiki del módulo SD, usando para ello unas bibliotecas del sistema FAT denominado FATfs. Así como las funciones para la inicialización, lectura y escritura en el RTC de la placa. Por último se describen los pasos dados para intentar compatibilizar el módulo de radio CC2520 con el sistema operativo. Cabe destacar que la compatibilización del módulo de radio no ha podido realizarse en su totalidad. Ya que debido a la extensión de esta tarea, solamente se ha conseguido inicializar el módulo de radio en Contiki.

Cabe destacar que todos los detalles de los pasos llevados a cabo para realizar las modificaciones comentadas anteriormente, se encuentra exhaustivamente detallados en el Anexo 1. En él se encontrarán también los códigos fuentes de cada una de las aplicaciones y bibliotecas desarrolladas en Contiki.

En este capítulo se describirá la implementación sobre Contiki de cada una de las funcionalidades de las que dispone la placa MEWiN. Es decir, se pretenderá conseguir que todos los dispositivos controlables por el microcontrolador sean accesibles para el programador desde el sistema operativo Contiki y de una forma sencilla e intuitiva.

4.1 Instalación de Contiki.

El primer paso es instalar el sistema operativo Contiki en el ordenador. Existen diversas alternativas para conseguirlo. La más sencilla sería utilizar InstantContiki.

InstantContiki es una imagen para el software de máquinas virtuales VMware que se ejecutará sobre Windows, esta alternativa es la más rápida pero no es del todo elegante, de forma que se ha considerado instalar Contiki de forma nativa utilizando alguna de las distribuciones de Linux disponibles en el mercado. En esta caso se utilizará Ubuntu 10.04 sobre la cual se instalará el compilador de Ubuntu y todos sus directorios.

Una vez instalado Ubuntu, se debe de instalar Contiki, el problema principal surge debido a que Contiki no soporta oficialmente el microcontrolador MSP430F2618, por ello se debe de instalar en primer lugar el compilador para el dispositivo. Los pasos a seguir para conseguir compilar nuestra aplicación en la placa MEWiN están descritos en el Anexo1-1. Tras la instalación de Contiki se instalará una pequeña aplicación para poder llevar a cabo la compilación de las aplicaciones en el dispositivo MEWiN (ver Anexo A1-1).

4.2 Habilitación de la programación de MEWiN a través del puerto USB.

En este momento, Contiki se encuentra instalado y funcionando en nuestro sistema operativo. Sin embargo, a pesar de que ya es posible compilar las aplicaciones creando el binario necesario, no es posible cargarlo en MEWiN ya que Contiki no soporta por defecto el dispositivo MSP430 USB-Debug -interface, que es el que se utiliza para cargar y hacer debug en nuestro dispositivo:

Figura 4-1: Dispositivo para programar el microcontrolador a través del USB, por medio de la interfaz JTAG.

Por tanto, para cargar la aplicación en el microcontrolador a través del MSP430-Debug Interface se usará la aplicación MSPDebug, la cual está diseñada para programar y hacer Debug en microcontroladores de la familia del MSP430. Soporta los programadores eZ430-F2013, eZ430-RF2500, FET430UIF (el que vamos a utilizar) y el Olimex MSP-JTAG-TINY.

Todos los pasos necesarios para instalar este dispositivo se encuentran descritos en el Anexo 1-2.

4.3 Habilitación de la comunicación de la placa MEWiN a través del puerto serie.

Contiki dispone de una serie de bibliotecas para manejar la recepción y envío de datos a través de la interfaz serie. Además, la plataforma Zolertia Z1 tiene los mismos pines que externalizan el conector serie que nuestra placa MEWiN, por tanto sólo hará falta realizar unos cambios menores en los archivos de configuración de Contiki para adaptarnos a las características(ver Anexo 1-3).

4.4 Creación de la Plataforma MEWiN en Contiki.

Se creará una nueva plataforma, tomando como base la plataforma Z1, para ello se copiará la carpeta Z1 y se renombra a MEWiN. A continuación se deben de cambiar una serie de archivos para conseguir compilar la aplicación sobre la nueva plataforma creada (ver Anexo 1-3).

4.5 Habilitación de los LEDs de la placa MEWiN en Contiki.

Una vez realizado los pasos citados en el anexo, ya es posible compilar nuestra aplicación usando la nueva plataforma MEWIN. Simplemente se deberán seguir los pasos citados en el Anexo 1-4. Sin embargo si se intenta usar alguna de las interfaces globales proporcionadas por Contiki:

- Por ejemplo leds_on(LEDS_RED);// Enciende el Led ROJO

No funcionará correctamente porque las conexiones de nuestra placa MEWiN no son las mismas que las de la placa que hemos utilizado como base (Z1). Se deben realizar una serie de modificaciones en las definiciones para conseguir que funcione correctamente sobre nuestra plataforma.

Como los cambios a realizar para que funcionen todas las directivas de Contiki, así como los periféricos de la placa MEWIN (RTC, SD) son muy diversos, se empezará con la parte más básica.

Para ello en primer lugar se pretende conseguir que funcione correctamente las diversas directivas para encender/ apagar los LEDS de la placa:

- leds_on(LED_A_ENCENDER)
- leds_off(LED_A_APAGAR)

Para llevar a cabo lo anterior, en primer lugar se deben saber cuáles son los puertos donde están conectados los leds de la placa Z1 al microcontrolador(ver Anexo 1-6).

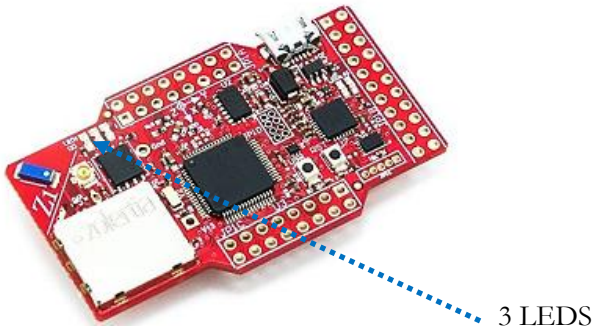


Figura 4-2: Detalle mote Zolertia Z1.

Como se puede apreciar, la placa Z1 dispone de tres LEDS de estado.

- LED1 en el puerto P5.4
- LED2 en el puerto P5.6
- LED3 en el puerto P5.5

Por tanto se deberán cambiar los puertos dentro del archivo de configuración de Contiki, además se deberá añadir las funciones y variables necesarias para el correcto funcionamiento del cuarto LED de la placa MEWiN (ver Anexo 1-7)

A continuación, tras realizar los cambios citados, se realiza un pequeño programa para comprobar el funcionamiento correcto de las directivas de los LEDS (ver Anexo A1-8).

4.6 Módulo SD de la placa MEWiN

La placa MEWiN dispone de una ranura para insertar una tarjeta SD. La idea de la existencia de una tarjeta SD (ver Anexo 1-9) es dotar al sistema diferentes funcionalidades, a continuación se citarán las más importantes:

1. Mejorar la capacidad de almacenamiento del dispositivo.

   Este hecho es muy importante, sobretodo en Redes inalámbricas de sensores, ya que como se ha comentado anteriormente, los dispositivos que forman las WSN tienen muchas restricciones hardware debido a que

necesitan tener un bajo consumo energético. En nuestro caso, el MSP430F2618 dispone de 116KB + 256B de memoria Flash y 8KB de memoria RAM por tanto, dotar al dispositivo de una cantidad de memoria (2GB)que sea accesible al programador es un hecho notable.

Además, disponer de esta ingente cantidad de almacenamiento, lo hace idóneo para funcionar como memoria de backup, es decir el dispositivo cada vez que tome una muestra, además de transmitirla por la radio, guardará en la tarjeta de memoria un log de los datos que constará de la fecha y hora en la que se tomó la muestra seguido de el valor de ésta, por tanto el RTC es prácticamente imprescindible si queremos usar las tarjetas SD como una memoria de backup en nuestro dispositivo.

2. Mejora en la comunicación con el usuario.

En toda red de sensores existe un nodo sumidero o coordinador que recibe los datos del resto de nodos de la red, este sumidero está conectado normalmente a un PC y se comunica con él a través de un puerto serie. Si nuestro dispositivo no dispusiera de una tarjeta SD, la única forma de comunicación desde el dispositivo hasta el usuario es mediante puerto serie. El hecho de disponer de una tarjeta SD tanto en el sumidero como en los nodos sensores, añade además una forma de comunicación extra con el usuario, de manera que podrá llevarse consigo los datos recibidos por los sensores sin necesidad de usar un PC.

3. Facilidad de uso .

Otra de las características muy interesantes de las tarjetas SD es su facilidad de uso, en cualquier momento, el usuario será capaz de cambiar la tarjeta SD por otra sin necesidad de tener conocimientos de soldadura. Además, la interfaz eléctrica de las tarjetas SD es muy simple, requiriendo como máximo seis cables para comunicarse y permitiendo velocidades de transferencia en rango de Mbps. En comparación con la interfaz USB y CF/CF+, la interfaz física de las tarjetas SD es muy simple, disminuyendo la complejidad en el diseño.

4. Bajo Coste.

Una de las principales razones del éxito de las tarjetas SD es su bajo precio. Una tarjeta SD de 2 GB ronda los 8 € y supone un desembolso muy pequeño en comparación con el coste global de hardware de una red extensa de este tipo de dispositivos.

5. Bajo Consumo.

   El bajo consumo es una de las características imprescindibles en una red inalámbrica de sensores. En éste caso, el dispositivo MEWiN está diseñado de forma que mientras no haya comunicación entre el microcontrolador y la tarjeta SD, ésta no suponga ningún consumo extra de energía, es decir , si no se está leyendo o escribiendo ningún dato en la tarjeta, el consumo del dispositivo es el mismo que sin ella. Además cabe destacar que las tarjetas SD tienen un consumo bajo en comparación con otras tarjetas como las Compact Flash,XD,SMC, etc.

4.6.1 Modos de funcionamiento de las tarjetas SD.

Existen 3 modos de transferencia soportados por SD:

- Modo SPI: entrada separada serial y salida serial.
- Modo un-bit SD: separa comandos, canales de datos y un formato propietario de transferencia.
- Modo cuatro-bit SD: utiliza terminales extra más algunos terminales reasignados para soportar transferencias paralelas de cuatro bits.

Las tarjetas de baja velocidad soportan tasas de transferencia de 0 a 400 Kbps y modo de trasferencia un-bit SD, mientras que las tarjetas de alta velocidad soportan tasas de transferencia de 0 a 100 Mbps en el modo de cuatro-bit, y de 0 a 25 Mbps en el modo un-bit SD.

En este caso, el modo a utilizar será el Modo SPI, ya que puesto que el microcontrolador MSP430F2618 soporta la comunicación con este protocolo, será la forma más fácil de llevar a cabo la comunicación con el microcontrolador.

4.6.2 Conexionado de las tarjetas SD.

En este apartado se describirá el conexionado de la tarjeta SD al microcontrolador. El diagrama de una tarjeta SD es el siguiente:

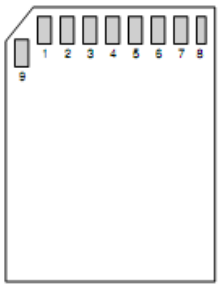


Figura 4-3: Pines tarjeta SD.

La tabla siguiente muestra las asignaciones de los pines de las tarjetas SD:

Tabla 4-1:Asignación de pines de la tarjeta SD.

Pin	Name	Function (SD Mode)	Function (SPI Mode)
1	DAT3/CS	Data Line 3	Chip Select/Slave Select ($\overline{SS}$)
2	CMD/DI	Command Line	Master Out Slave In (MOSI)
3	VSS1	Ground	Ground
4	VDD	Supply Voltage	Supply Voltage
5	CLK	Clock	Clock (SCK)
6	VSS2	Ground	Ground
7	DAT0/DO	Data Line 0	Master In Slave Out (MISO)
8	DAT1/IRQ	Data Line 1	Unused or IRQ
9	DAT2/NC	Data Line 2	Unused

Como ya se ha indicado, las tarjetas SD soportan varios tipos de protocolos. Puesto que el uC MSP430F2618 es totalmente compatible con el protocolo SPI, este protocolo es el más adecuado y el que se utilizará para llevar a cabo la implementación sobre el microcontrolador. El modo de comunicación SPI soporta solamente una parte de todas las características del protocolo completo de la SD. Sin embargo, los comandos no

soportados no son necesarios en el modo SPI. Por tanto, es posible realizar una implementación completamente funcional usando el modo SPI.

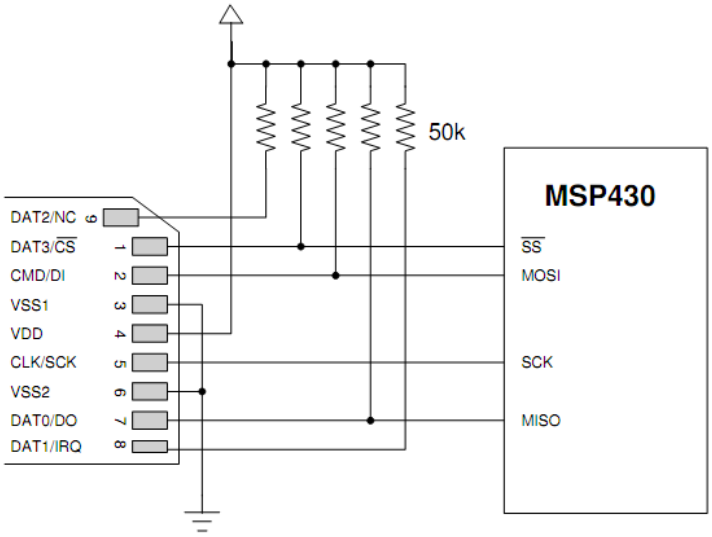


Figura 4-4: Esquemático de la tarjeta SD.

La Figura 4-4 muestra el conexionado eléctrico para el funcionamiento en modo SPI. Las resistencias de pullup externas son necesarias para el correcto funcionamiento del protocolo.

Cabe destacar que el tamaño máximo de la tarjeta SD con la que podemos trabajar es de 2 GBytes, esto es debido a que las tarjetas SD con capacidad mayor de 2GBytes son denominadas de alta capacidad (SD-HC) y no son compatibles con el protocolo SPI. Sin embargo 2 GB de capacidad de almacenamiento es más que suficiente para nuestro propósito ya que se almacenarán datos de texto plano y no archivos de imágenes, videos, música, etc.

4.6.3 Protocolo de comunicación con la tarjeta SD:

El protocolo de comunicación de la SD con el microcontrolador a través de la interfaz SPI se describe a continuación. La secuencia de inicialización, el formato de trama y los identificadores de comandos difieren de los otros modos de comunicación también disponibles , conocidos como SD4-bit y SD 1-bit. El protocolo que se va a utilizar es un protocolo sencillo de comando-respuesta. Todos los comandos son iniciados por el dispositivo maestro (uC). La tarjeta SD responde con al comando con una trama de

respuesta y a continuación, en función del comando recibido, puede ir seguido de un valor que indica el inicio de una transmisión de datos que irá a continuación.

Los comandos para la comunicación con la tarjeta SD están listados de la forma "CMDXX" o "ACMDXX", donde CMD y ACMD se refieren a "comandos generales" y comandos específicos de la aplicación" respectivamente. Siendo XX el número del comando. Los comandos son enviados a la SD en un en una trama de 6-bytes enviados a través del puerto SPI. La trama del comando comienza con un "01" seguido de 6 bits que corresponden al número del comando. A continuación van 4 bytes enviando el bit más significativo al principio., seguidos de los 7 bits del CRC, con un bit de paro al final. Todos los bits de las tramas de comandos son enviados por el pin MOSI (MSB primero). Hay que tener en cuenta que el control CRC es opcional en el modo SPI, estando desabilitado por defecto.

Tabla 4-2: Formato de los comandos de la tarjeta SD.

First Byte			Bytes 2-5	Last Byte	
0	1	Command	Argument (MSB First)	CRC	1

La tarjeta SD responde a cada comando. De forma que cada uno de los comandos tiene un respuesta esperada. El tipo de respuesta usada para un comando en concreto depende solamente del tipo de comando y no del contenido del paquete de datos. Existen 3 tipos de respuesta definidos en el modo SPI: R1, R2 y R3. En las siguientes tablas se describen los valores que pueden tomar los bits:

Tabla 4-3: Respuestas tipo R1 de las tarjetas SD.

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State

Tabla 4-4: Respuestas tipo R2 de las tarjetas SD.

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State
2	7	Out of Range, CSD Overwrite
	6	Erase Parameter
	5	Write Protect Violation
	4	Card ECC Failed
	3	Card Controller Error
	2	Unspecified Error
	1	Write Protect Erase Skip, Lock/Unlock Failed
	0	Card Locked

Tabla 4-5: Respuestas tipo R3 de las tarjetas SD.

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State
2-5	All	Operating Condition Register, MSB First

Existe un modo de transferencia que proporciona un mecanismo para transmitir grandes cantidades de datos desde y hacia la SD. Mientras que el modo normal de comandos/respuestas sólo permite la transmisión de pequeñas cantidades de datos en tamaños fijos y el modo de transferencia permite que las cantidades sean de tamaño arbitrario.

Cuando un comando de transferencia de datos es enviado a la SD. La tarjeta responde con una de las respuestas estándar (R1,R2 o R3). A continuación la transferencia de datos comienza con un indicador del comienzo de trama y a continuación la trama de datos y finaliza con un CRC de 16 bits.

Para la lectura o la escritura de un bloque, la transferencia de datos comienza con un indicador de comienzo de trama constante que es "11111110". A continuación le sigue el bloque de datos (normalmente 512 bytes) y 16 bits de CRC (opcional).

Cada vez que la tarjeta escriba los datos, retornará un bytes indicando el estado de la operación realizada. La respuesta es del tipo "XXX0AAA1" los tres primeros bits no nos interesan, siendo el valor de las A´s las que indican el estado de la SD. "010" implica que los datos fueron aceptados, "101" indica que los datos no fueron aceptados debido a un error de CRC. "110" indica que los datos fueron rechazados debido a un error de escritura.

Finalmente, en caso de lectura de un bloque, si una lectura de un bloque falla, la SD devolverá un byte de error, el formato de este byte es el siguiente.

Tabla 4-6: Errores de lectura de las tarjetas SD.

Bit	Meaning
7	Always '0'
6	Always '0'
5	Always '0'
4	Card Locked
3	Out of Range
2	Card ECC Failed
1	Card Controller Error
0	Unspecified Error

A continuación, en la siguiente tabla, se indican los comandos más relevantes para comunicarse la tarjeta SD.

Tabla 4-7: Comandos importantes de las tarjetas SD.

Command	Argument	Type	Description
CMD0	None	R1	Tell the card to reset and enter its idle state.
CMD16	32-bit Block Length	R1	Select the block length.
CMD17	32-bit Block Address	R1	Read a single block.
CMD24	32-bit Block Address	R1	Write a single block.
CMD55	None	R1	Next command will be application-specific (ACMDXX).
CMD58	None	R3	Read OCR (Operating Conditions Register).
ACMD41	None	R1	Initialize the card.

Implementación.

La implementación del driver está realizado entorno a tres funciones principales:

- Función de inicialización de la tarjeta SD.
- Función para la escritura de un bloque en la tarjeta SD.

- Función para la lectura de un bloque de la tarjeta SD.

Función de inicialización de la tarjeta SD:

Las tarjetas SD requieren un secuencia de inicialización expecífica. La inicialización comienza fijando el funcionamiento de la señal de reloj del bus SPI a 400 kHz. Es un requisito fundamental para asegurar un rango amplio de compatibilidad con tarjetas SD y MMC disponibles. Tras esto, el dispositivo master debe esperar al menos 74 ciclos de reloj antes de realizar ningún intento de comunicación con la tarjeta. Esto permite a la tarjeta inicializar los registros internos antes de que la inicialización de la tarjeta continúe.

A continuación, se manda el comando CMD0 mientras se mantiene el pin SS a bajo. Esta combinación resetea la tarjeta y la pone en modo SPI. Hay que tener en cuenta que a pesar de que el CRC es ignorado generalmente en el modo SPI, el primer comando debe de ir seguido de un CRC válido ya que la tarjeta no se encuentra todavía en modo SPI. El byte de CRC para el comando CMD0 es 0x95.

Tras ésto, la tarjeta debe recibir los comando CMD55 y ACMD41 hasta que el bit de 'idle' vuelva a cero, indicando que la tarjeta SD se ha inicializado correctamente y preparada para responder al resto de comandos generales. A continuación, el comando CMD58 se usa para comprobar que la tarjeta soporta el voltaje de operación del procesador, sin embargo, una vez comprobado que es soportado, se ha optado por eliminar este paso para que el proceso se realice más rapidamente. Tras esto, el reloj del SPI se puede poner al valor máximo para conseguir velocidades mayores, sin embargo este paso se ha suprimido para evitar problemas de compatibilidad.

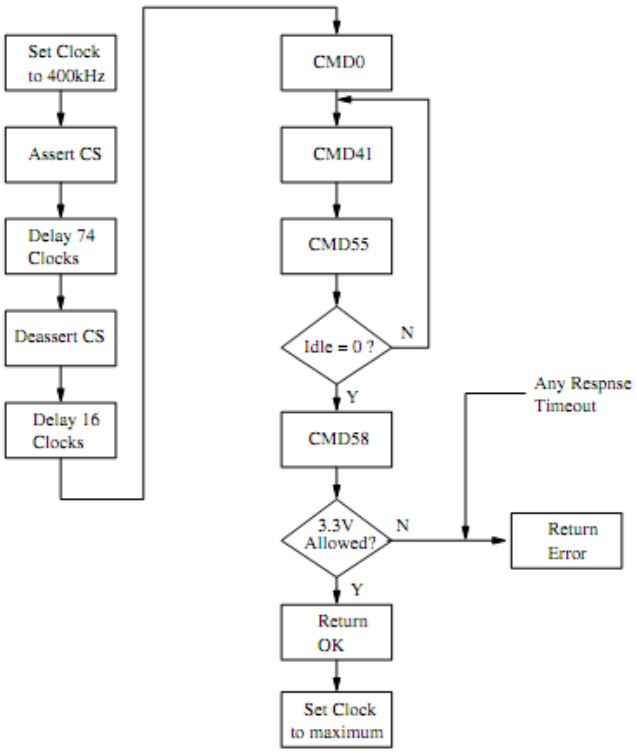


Figura 4-5: Secuencia de inicialización de la tarjeta SD.

Todas las bibliotecas de la SD se encuentran en el Anexo 1-10.

4.6.4 Pruebas de funcionamiento de la tarjeta SD

La comunicación con la tarjeta SD se realiza correctamente y es posible tanto leer de la tarjeta como escribir en ella. Sin embargo, el principal problema de usar este sistema para almacenar datos, es que al no utilizar un sistema de archivos, al insertar la tarjeta en un ordenador, la tarjeta no será reconocida, y deberemos usar un programa que nos permita ver los diferentes sectores de la tarjeta para extraer los datos. Para ello se puede usar el programa HxD:

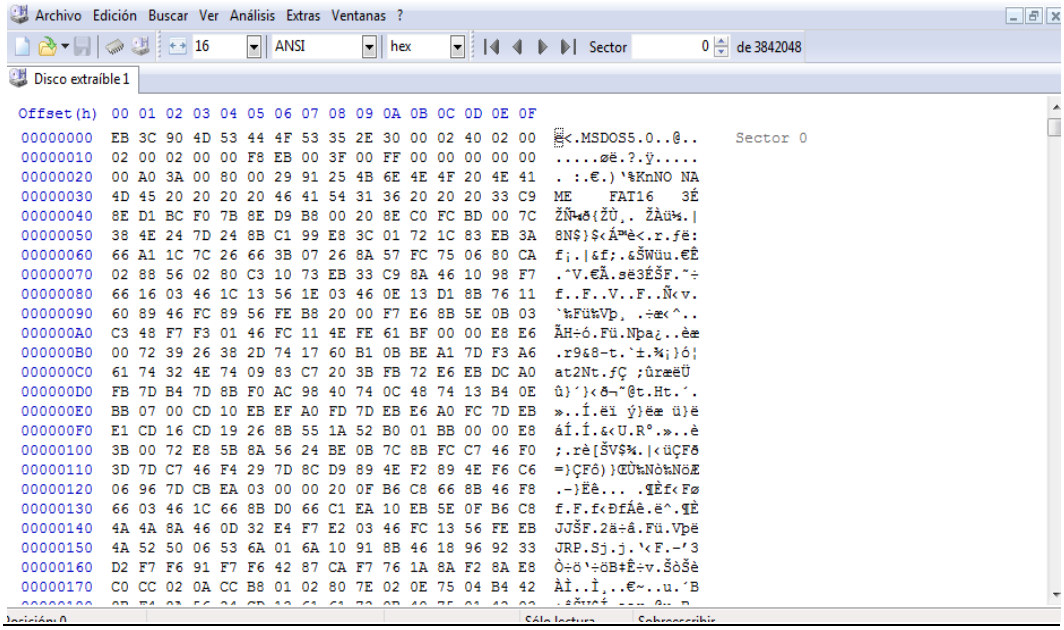


Figura 4-6: Vista de los datos almacenados en la SD con editor hexadecimal.

Este hecho complica sobremanera la extracción de datos de la tarjeta (ver Figura 4-6) , ya que supone analizar cada uno de ellos y convertirlos a un formato adecuado para su posterior tratamiento y/o representación.

Por tanto se va a considerar implementar un sistema de archivos en la tarjeta, que permita al microcontrolador gestionarlo y consiguiendo que sea reconocible en un ordenador para que la extracción de los datos se realice de forma rápida. El problema principal de llevar a cabo esta decisión proviene de las limitadas características técnicas de los dispositivos usados en las redes inalámbricas de sensores. Como ya se ha comentado anteriormente, estas limitadas características técnicas vienen impuestas por la necesidad de un bajo consumo. Es necesario pues conseguir dar soporte a un sistema de archivos conocido, que permita trabajar con los datos desde un ordenador de manera sencilla, con la complicación añadida de las limitadas características técnicas de nuestro dispositivo.

Tras analizar las diferentes alternativas, se considera que la más viable es una implementación de biblioteca FAT diseñadas específicamente para ser usadas en dispositivos con características técnicas muy limitadas, denominada FATfs, el hecho de barajar esta elección es debido a la facilidad de portarla a diferentes dispositivos ya que está implementada de tal forma que es totalmente independiente del hardware donde se vaya a utilizar, sólo serán necesarias una serie de funciones a bajo nivel que permitan la comunicación con el dispositivo.

4.6.5 Biblioteca FATfs

FATFs es un módulo para sistema de archivos FAT genérico diseñado para ser usador en pequeños dispositivos embebidos. Está escrito por ChaN, con licencia de código libre y puede ser descargado gratuitimente de su página web: "*http://elm-chan.org/fsw/ff/00index_e.html*". El sistema FATFs está escrito enteramente en ANSI C y está separado completamente de la capa de entrada y salida del dispositivo. Por tanto, es fácilmente portable independientemente del hardware en que se quiere a implementar. La última revisión disponible de las bibliotecas es la r0.08b y data de Enero de 2011.

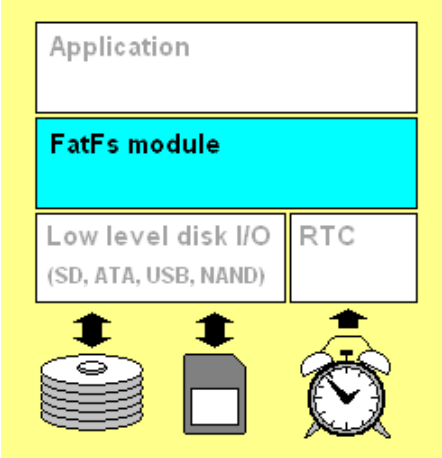


Figura 4-7: La biblioteca FATfs es independiente del hardware donde se utilice.

El hecho de que el sistema sea completamente independiente del hardware (ver Figura 4-7) es crucial, ya que permite que el port a diferentes dispositivos se pueda realizar de manera sencilla y rápida. Todos los pasos llevados a cabo para realizar el port se encuentran en el Anexo 1-11.

4.6.5.1 Pruebas de Funcionamiento del sistema de archivos FATfs.

El código implementado para comprobar el correcto funcionamiento del sistema de archivos FATfs en la tarjeta SD se encuntra en el Anexo 1-12. Tras compilar y cargar la placa MEWiN con el código del anexo, se comprueba introduciendo la tarjeta en un PC y que se ha creado el fichero y los datos son los correctos.

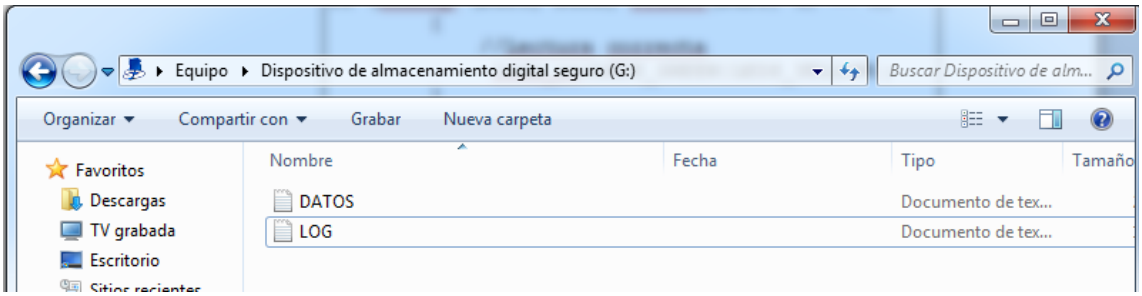


Figura 4-8: Ficheros almacenados en la SD

Comprobamos como en el explorador de archivos aparece el fichero creado con nombre "*DATOS.txt*"(ver Figura 4-9).

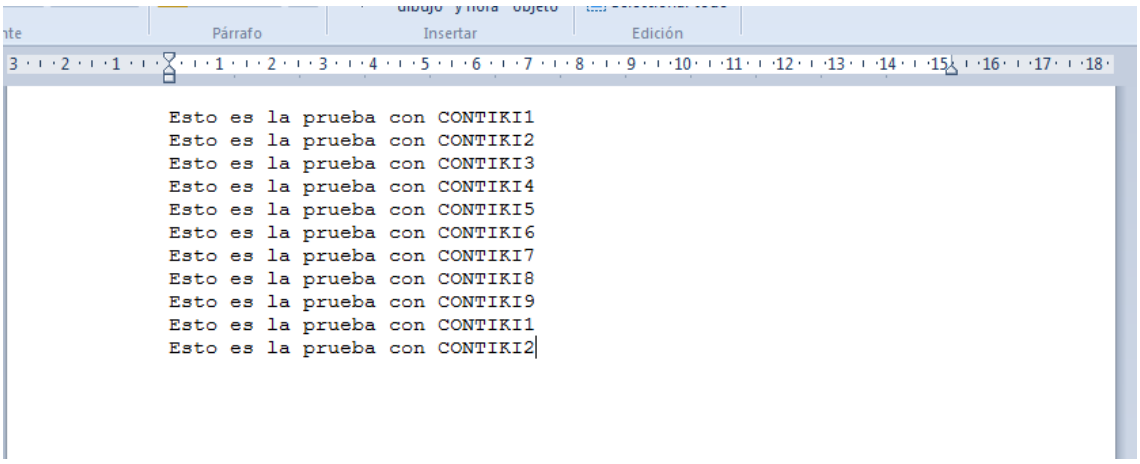


Figura 4-9: Ejemplo de los datos almacenados en la tarjeta SD en un visor de textos de un PC

Así pues, se ha demostrado que la biblioteca FATfs es totalmente funcional, con toda la ventaja que supone utilizar un sistema de archivos que permita ver de forma muy sencilla todos los datos almacenados por dispositivo en un ordenador simplemente introduciendo la tarjeta en un lector de tarjetas SD.

4.7 Módulo RTC de la placa MEWiN.

El circuito integrado DS1339(ver Figura 4-10) corresponde a un reloj en tiempo real ("RTC", Real Time Clock) de bajo consumo. Las direcciones y los datos son

transferidos de forma serie a través del bus I2C. El reloj proporciona: segundos, minutos, horas, día de la semana, día del mes, mes y año . Es capaz de detectar años bisiestos. El reloj puede operar en formato 12 o 24 horas con indicador de AM/PM. El sistema dispone de una batería de backup, de forma que el sistema automáticamente cambia de alimentación si la fuente principal falla.

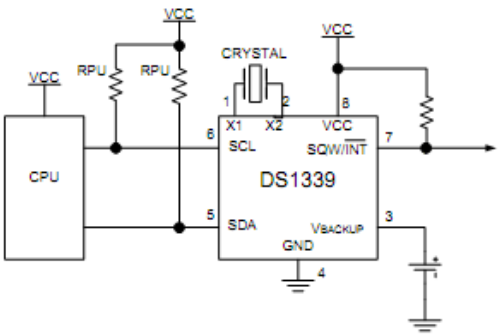


Figura 4-10: Circuito asociado al RTC.

El dispositivo dispone de una serie de registros que se deberemos escribir para fijar la hora actual la primera vez que programemos el microcontrolador. Una vez guardada la hora, cada vez que se desea saber la hora actual, se deberá solicitar la lectura de los registros asociados a cada uno de los campos a través de la interfaz I2C.

Tabla 4-8: Registros asociados al RTC.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{\text{AM/PM}}$ / 20 Hour	10 Hour	Hour				Hours	1–12 +AM/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99

Como se puede comprobar en la Figura 4-11, cada uno de los registros dispone de 8 bits, el primero de ellos tiene la dirección 00 y corresponde a los segundos, el último de los registros dispone de la dirección 06 y corresponde al año.

En el Anexo 1-13 se encuentran las bibliotecas que implementan las funciones del RTC de la placa MEWiN. A continución se presentará una prueba de funcionamiento del sistema completo, haciendo uso del RTC y de la SD de manera conjunta. Todo el código de la aplicación del prueba del sistema se encuentra en el Anexo 1-14. Tras compilar, cargar

y ejecutar el programa en una placa MEWiN con una tarjeta SD insertada.Comprobamos que los datos han sido almacenados (ver Figura 4-12).

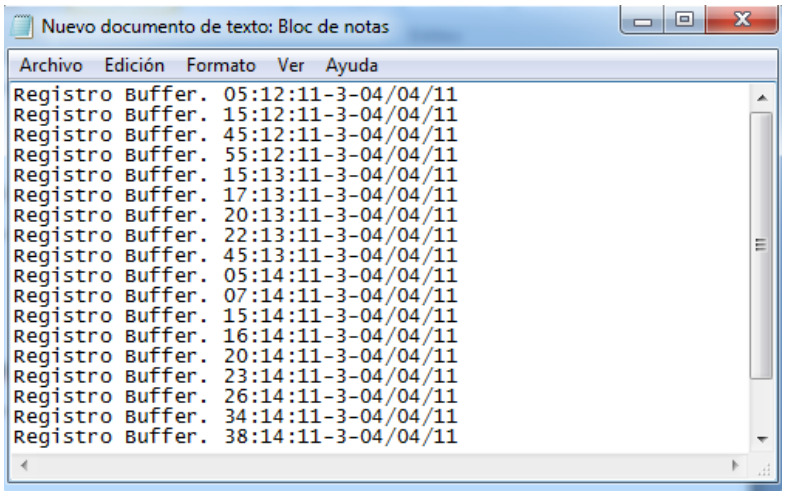


Figura 4-11: Log con los datos almacenados en la tarjeta SD.

El funcionamiento del sistema es el adecuado, de esta forma se ha demostrado que los componentes software desarrollados en Contiki funcionan correctamente y nuestro dispositivo y sus periféricos ya pueden ser programados de manera muy sencilla en este sistema operativo tan popular en las redes de sensores inalámbricas.

4.8 Modulo de radio en Contiki

El módulo de radio CC2420 es uno de los más utilizados en los motes comerciales actuales (TelosB,Z1, etc.). En Contiki, es el único módulo de radio soportado oficialmente en la actualidad.

En este apartado se pretende dar soporte al módulo de radio CC2520, el cual no está soportado por el sistema operativo Contiki. La complejidad de esta labor es muy elevada, ya que a pesar de que el modelo anterior del módulo de radio de Texas Instruments CC2420 si está plenamente soportado por Contiki, los cambios entre el módulo CC2520 y CC2420 son muy elevados y apenas se pueden aprovechar las bibliotecas implementadas del CC2420 para el CC2520.

El módulo de radio CC2520 del mote MEWiN, se puede considerar como un periférico conectado al microcontrolador por medio del protocolo SPI, la labor del módulo de radio es la de recibir y enviar datos a través de la interfaz radio. De esta forma, cuando

se desee enviar un paquete a través de la interfaz radio, el microcontrolador enviará hacia el módulo de radio una serie de comandos así como los datos que se deseen enviar.

Al igual que sucedía con el RTC, el módulo de radio, dispone de una serie de registros que deberán ser escritos y leídos por parte del microcontrolador. Por tanto, el primer paso para afrontar esta labor consiste en intentar inicializar el módulo de radio en Contiki, es decir, comunicarse con el módulo de radio y escribir una serie de registros que definirán las características de la conexión: Potencia de transmisión, canal, encriptación, etc. Todos los pasos llevados a cabo para inicializar el módulo de radio, están descritos en el Anexo 16. Para llevar a cabo la inicialización del módulo de radio, es necesario lograr comunicarse con el módulo de radio a través del protocolo SPI. Debido a que se está usando como plataforma base el mote Z1, el primer paso será cambiar los puertos del microprocesador en los que están conectados al módulo Z1 por los usados por el mote MEWiN. Como ya se ha comentado, todos los pasos para llevar a cabo el cambio de puertos en la inicialización del dispositivo se encuentran descritos exhaustivamente en el Anexo 1-16.

Tras realizar los pasos indicados, se puede comprobar cómo se ha inicializado correctamente el módulo de radio. El siguiente paso sería portar todas las funciones que se usan en Contiki con el módulo CC2420 (ver Anexo 1-15) para hacerlas compatibles con el CC2520, sin embargo esta labor es muy compleja y no ha sido posible llevarla a cabo en el tiempo en que se ha estado desarrollando el presente proyecto., en el anexo, se encuentran enumeradas y descritas los archivos más importantes que deberían servir de punto de partida para conseguir compatibilizar el módulo de radio CC2520 en Contiki..

Capítulo 5

Conclusiones y trabajos futuros

En el siguiente apartado se obtendrán las conclusiones obtenidas tras el desarrollo del proyecto. Así como se presentarán las dificultades surgidas durante el mismo. Por otro lado se mostrarán las líneas futuras que podrían seguir trabajos asociados a este proyecto.

En el siguiente proyecto, se ha pretendido dar soporte a la plataforma MEWiN en el sistema operativo Contiki. La labor ha sido compleja pero satisfactoria. Las principales dificultades encontradas radicaban en la escasa documentación existente del propio Contiki. Esa es una de los principales escoyos a la hora de trabajar en este sistema operativo.

Se ha demostrado que la plataforma MEWiN es parcialmente funcional dentro de Contiki, ya que todos los dispositivos de el sistema MEWiN han sido adaptados a Contiki y son totalmente funcionales excepto el módulo de radio. Es decir, se han desarrollado las funciones y bibliotecas necesarias para que puedan ser utilizados por posibles usuarios futuros de la plataforma de manera muy sencilla. Con la salvedad del módulo de radio, del cual sólo ha sido posible conseguir su inicialización. Este módulo de radio a pesar de llevar varios años en el mercado no tiene compatibilidad con este S.O. Esto es debido la complejidad y laboriosidad de crear unos drivers completamente funcionales para el módulo de radio CC2520 en Contiki. A pesar de que existe un port parcial que ha sido desarrollado la universidad Johns Hopkins para otro sistema operativo como TinyOS, este no es completamente funcional a fecha de hoy (05/09/2011), por tanto, se puede concluir que en la actualidad, el módulo de radio, a pesar de llevar varios años en el mercado, no es totalmente compatible con ninguno de los dos sistemas operativos para redes de sensores inalámbricas más conocidos (Contiki y TinyOS).

Por tanto actualmente la comunicación entre dos dispositivos MEWiN a través del módulo de radio usando el sistema operativo Contiki no es posible. Como líneas futuras se podría plantear la creación de los drivers para el módulo de radio de forma escalonada en diferentes capas para conseguir dar soporte completo a la plataforma y de esta forma obtener un dispositivo compatible con IPv6, que es una de las principales bazas de Contiki.

Esta labor a pesar ser realmente compleja, permitiría al dispositivo, además de comunicarse por radio con otros dispositivos, hacer uso de la pila de protocolos IPv6 desarrollada en Contiki, con todas las ventajas que ello conllevaría como se pudo ver en los ejemplos de funcionamiento mostrados en el capítulo 3.1.5.

ANEXO I

En el siguiente apartado se pretende describir detalladamente los pasos seguidos para llevar a cabo la implementación y desarrollo software del presente PFC sobre el sistema operativo Contiki.

1. Instalación de Contiki.

1) Se Añaden los repositorios de tinyos y se borran los antiguos si los hubiere.

```
deb http://hinrg.cs.jhu.edu/tinyos lucid main
```

2) Se actualiza la cache de los repositorios:

```
sudo apt-get update
```

3) Instalamos tinyos

```
sudo apt-get install tinyos-2.1.1
```

4) Se añade a `~/.bashrc` o `~/.`en el directorio home para que se carguen los parámetros de tinyos al arrancar la consola.

```
source /opt/tinyos-2.1.1/tinyos.sh
```

Instalación Contiki y las bibliotecas del microcontrolador MSP430F2618, para ello necesitamos:

1. GCC (latest version is 4.4.4).
2. Binutils (latest version is 2.20.1)
3. Libc support.
4. Python-serial support (BSL).

Para instalarlas de la forma más sencilla se usará como patrón la guía y repositorios proporcionados por el fabricante del mote Zolertia Z1:

En primer lugar se necesita instalar las herramientas necesarias para compilar el mote Z1, para ello se necesitan las sisguientes bibliotecas:

- GCC (latest version is 4.4.4).
- Binutils (latest version is 2.20.1)
- Libc support.
- Python-serial support (BSL).

Comprobamos la version de las bibliotecas que tenemos:

dpkg -l 'gcc' 'binutils*' | grep ii*

Instalar o actualizar a la última versión:

sudo apt-get install subversion build-essential libcurses cvs texinfo python-serial*

Instalar la msp430 toolchain y arrancamos el script para el parcheo y la instalación:

wget http://sourceforge.net/projects/zolertia/files/contiki-enviroment/mspgcc4-20100815.tar.bz2
tar -xvjf mspgcc4-20100815.tar.bz2
cd mspgcc4-20100815
sudo sh buildgcc.sh

Cuando lo indique se seleccionará sólo las opciones donde aparezcan GCC y LibC.

Ahora añadimos el mspgcc4 a nuestras variables de entorno.

echo "export PATH=/opt/msp430-gcc-4.4.4/bin:$PATH" >> ~/.bashrc

Cerrar el terminal y lo volver a abrir para actualizar los cambios. Comprobar el PATH escribiendo:

echo $PATH

Ahora que el entorno para Contiki está preparado, descargar la última versión de Contiki de cvs:

cvs -d:pserver:anonymous@contiki.cvs.sourceforge.net:/cvsroot/contiki login
cvs -z3 -d:pserver:anonymous@contiki.cvs.sourceforge.net:/cvsroot/contiki co contiki-2.x

Si se quiere actualizar en algún momento Contiki sólo hace falta escribir:

cd contiki-2.x
cvs update -dP

2. Instalación de MSP430 USB-DEBUG -interface.

Una vez descargada e instalada la aplicación de la página web del autor:

http://mspdebug.sourceforge.net/download.html

Sólo es necesario ejecutar una serie de comandos para poder cargar código en la aplicación.

Ejecutar 'dmesg' y se comprueba si se ha reconocido el MSP430-FET430UIF y le ha asignado algún puerto:
En caso de que no le haya sido asignado ningún puerto, hacer lo sigiente:

```
# echo 2 > /sys/bus/usb/devices/x-y.z/bConfigurationValue
```

El cual asigna un valor a la variable de configuración del puerto usb.
Siendo x-y.z el valor obtenido al ejecutar el comando 'dmesg'.

3. Crear la Plataforma MEWiN en Contiki.

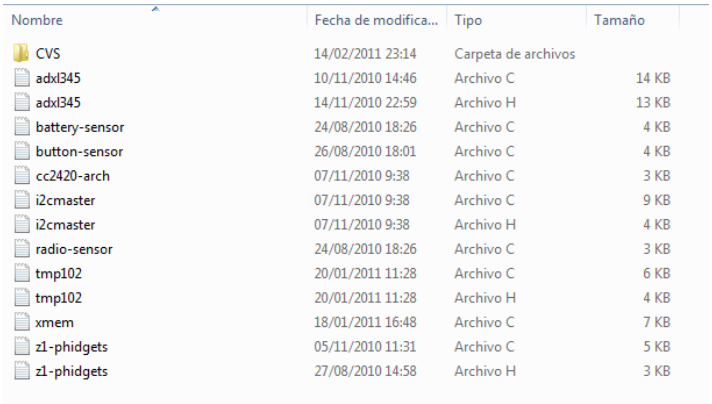
Se debe crear una carpeta llamada MEWiN dentro del directorio platform.

En la carpeta "\contiki-2.x\platform\mewin" deberemos cambiar los archivos:

-"*contiki-z1-main*" por "*contiki-mewin-main*".

-Se deberá buscar dentro de los archivos "*contiki-conf*" y "*platform.conf*" la palabra z1 y sustituirla por MEWiN.

-Dentro de la carpeta *"/dev/"*:



Nombre	Fecha de modifica...	Tipo	Tamaño
CVS	14/02/2011 23:14	Carpeta de archivos	
adxl345	10/11/2010 14:46	Archivo C	14 KB
adxl345	14/11/2010 22:59	Archivo H	13 KB
battery-sensor	24/08/2010 18:26	Archivo C	4 KB
button-sensor	26/08/2010 18:01	Archivo C	4 KB
cc2420-arch	07/11/2010 9:38	Archivo C	3 KB
i2cmaster	07/11/2010 9:38	Archivo C	9 KB
i2cmaster	07/11/2010 9:38	Archivo H	4 KB
radio-sensor	24/08/2010 18:26	Archivo C	3 KB
tmp102	20/01/2011 11:28	Archivo C	6 KB
tmp102	20/01/2011 11:28	Archivo H	4 KB
xmem	18/01/2011 16:48	Archivo C	7 KB
z1-phidgets	05/11/2010 11:31	Archivo C	5 KB
z1-phidgets	27/08/2010 14:58	Archivo H	3 KB

Figura A-1: Estructura de la carpeta dev

Se deberán borrar todos los archivos. Esto es debido a que estos archivos corresponden a las diferentes bibliotecas que dan soporte al acelerómetro, sensor de temperatura, memoria flash, sensor de batería y sensores phidgets que contiene el dispositivo zolertia Z1 y que no se usarán en nuestro disposito.

-Por último, se debe eliminar éstos archivos del fichero de configuración "*Makefile*":

```
CFLAGS+=-Os -g
CLEAN += *.z1 symbols.c symbols.h

ARCH=msp430.c leds.c watchdog.c xmem.c \
     spix.c cc2420.c cc2420-aes.c cc2420-arch.c cc2420-arch-
sfd.c\
     node-id.c sensors.c button-sensor.c cfs-coffee.c \
     radio-sensor.c uart0x.c uart0-putchar.c uip-ipchksum.c \
     checkpoint-arch.c slip.c slip_uart0.c z1-phidgets.c

CONTIKI_TARGET_DIRS = . dev apps net
ifndef CONTIKI_TARGET_MAIN
CONTIKI_TARGET_MAIN = contiki-z1-main.c
endif
```

Figura A-2: Makefile sin modificar.

Para que quede de la siguiente forma:

```
CFLAGS+=-Os -g
CLEAN += *.z1 symbols.c symbols.h

ARCH=msp430.c leds.c watchdog.c \


CONTIKI_TARGET_DIRS = . dev apps net
ifndef CONTIKI_TARGET_MAIN
CONTIKI_TARGET_MAIN = contiki-z1-main.c
endif
```

Figura A-3: Makefile modificado.

4. Compilación y carga de la aplicación en MEWiN.

Compilar el código indicando que cree el .ihex:

make 'nuestra_aplicacion' TARGET=MEWIN nombre.ihex

Ejecutar 'dmesg' y ver si ha reconocido el MSP430-FET430UIF y le ha asignado algún puerto. En caso de que no le haya asignado ningún puerto, hacer lo sigiente:

 # echo 2 > /sys/bus/usb/devices/1-2.1/bConfigurationValue

(NOTA: 1-2.1 es el valor que tiene el dispositivo al ejecutar dmesg)

Ejecutamos 'mspdebug'(x es el valor del puerto asignado):

mspdebug uif -j -d /dev/ttyUSBx

Cargamos el archivo .ihex:

prog nombre.ihex

Arrancamos el programa:

run

Resetear dispositivo:

reset.

El programa funciona correctamente, demostrando que se ha compilado y cargado correctamente en el dispositivo.

5. Comunicación de la placa MEWiN con el PC a través de la UART.

Dentro del archivo *'contiki-mewin-main'*, que se encuentra en la carpeta platform/mewin (o platform/z1 si todavía no se ha creado la plataforma mewin).

Debemos añadir en el main de la aplicación , tras :

```
msp430_cpu_init();
clock_init();
leds_init();
leds_on(LEDS_RED);
clock_wait(100);
```

La siguiente linea :

```
uart0_init(BAUD2UBR(115200));
```

En la cual se inicializa la uart y se configura a una velocidad de 115200 baudios.

Ya sólo resta añadir en nuestra aplicación cada vez que deseemos enviar algo por la Uart lo siguiente:

```
printf("Texto que se quiera enviar");

//Incluso podemos ver el valor que toman las variables:
```

```
printf("La variable de tipo u8t 'a' vale: %u.\n", a);
```

Por tanto, se puede comprobar cómo la depuración a través de la uart es muy sencilla, sólo se necesita un programa terminal configurado a: 115200 bps, 8N1 y sin paridad.

6. *Esquemático del mote Z1.*

En primer lugar se comprobará la conexión de los leds de la placa Z1:

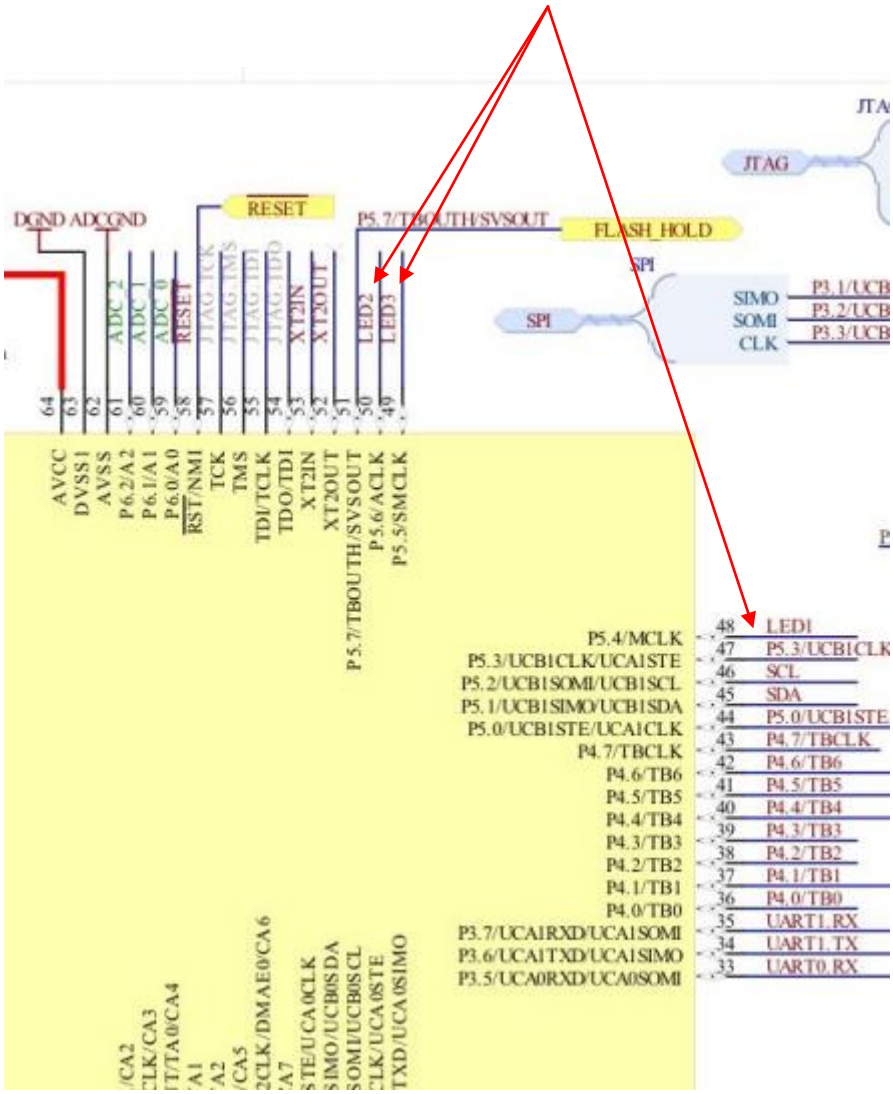


Figura A-4: Conexionado del microcontrolador del mote Zolertia.

7. Cambios en Contiki para el funcionamiento de las directivas de los LEDS.

El archivo de cabecera "platform-conf.h" se encuentra dentro de la carpeta *platform/z1* en el directorio raiz de Contiki:

```
/* LED ports */

#define LEDS_PxDIR P5DIR

#define LEDS_PxOUT P5OUT

#define LEDS_CONF_RED    0x10

#define LEDS_CONF_GREEN  0x40

#define LEDS_CONF_YELLOW 0x20
```

Se puede Comprobar que las dos primeras definiciones definen la directiva de P5DIR y P5OUT, indicando que los LEDs se encuentran en el puerto 5.

A continuación , comprobamos como el puerto del primer led (ROJO) es el P5.4 ya que 0x10 es en binario 0x00010000 y como el bit menos significativo corresponde al puerto 5.0, el quinto bit menos significativo será el correspondiente al puerto P5.4.

Usando esta misma deducción se puede comprobar que el segundo led (VERDE) corresponde al puerto el P5.6 y el tercer led (AMARILLO) al puerto P5.5 como habíamos comprobado en la hoja de características anteriormente.

En este caso, la placa MEWiN dispone de cuatro leds de estado, que se deberán direccionar apropiadamente.

Por tanto, para los tres primeros leds, bastará con cambiar dentro del archivo de configuración antes citado los valores de los puertos por los correspondientes en la placa MEWiN:

```
/*Modifico los puertos para que funcionen en Mewin*/
/* LED ports */
#define LEDS_PxDIR P4DIR
#define LEDS_PxOUT P4OUT
#define LEDS_CONF_RED    0x01
#define LEDS_CONF_YELLOW 0x02
#define LEDS_CONF_GREEN  0x04
```

En nuestro caso, las conexiones a la placa MEWiN se realizan a través de los puertos:

- LED_ROJO:P4.0
- LED_AMARILLO:P4.1
- LED_VERDE:P4.2

Ahora sólo resta añadir las funciones y los puertos para actuar sobre el último led, para ello debemos modificar los archivos "leds.c" y "leds.h" que se encuentran dentro de core/dev/ en la carpeta raiz de Contiki.

Elementos añadidos en el archivo "leds.h":

```
#ifdef MEWIN
#define LEDS_ALL    0x0F
#define LEDS_RED_2    8
#else
#define LEDS_ALL    7
#endif

.

.

.

#ifdef MEWIN
void leds_red_2(int onoroff);
#endif
```

En este archivo se definen en caso de que estemos compilando la plataforma MEWiN, los leds. Es decir, si trabajamos con MEWiN, como dispone de cuatro leds, debemos de activar los cuatro bits menos significativos "0x0F". En otro caso, como la mayoria de las plataformas desarrolladas para Contiki disponen de 3 leds, se activarán los tres bits menos significativos "0x07".

También se definen la nueva función que activará el cuarto led, en este caso se le ha llamado " leds_red_2" el 2 es debido a que el cuarto led es de color rojo y ya existe otro con este mismo color(LED1).

Elementos añadidos en el archivo "leds.c":

```
*-----------------------------------------------------------------------*/
#ifndef MEWIN
void
leds_on(unsigned char ledv)
{
  unsigned char changed;
  changed = (~leds) & ledv;
  leds |= ledv;
  show_leds(changed);
}
/*-----------------------------------------------------------------------*/
void
leds_off(unsigned char ledv)
{
  unsigned char changed;
  changed = leds & ledv;
  leds &= ~ledv;
  show_leds(changed);
}
#else
void
leds_off(unsigned char ledv)
{
  unsigned char changed;
  changed = (~leds) & ledv;
  leds |= ledv;
  show_leds(changed);
}
/*-----------------------------------------------------------------------*/
void
leds_on(unsigned char ledv)
{
  unsigned char changed;
  changed = leds & ledv;
  leds &= ~ledv;
  show_leds(changed);
}
void leds_red_2(int o) { o?leds_on(LEDS_RED_2):leds_off(LEDS_RED_2); }
#endif
```

En caso de que no estemos compilando para la plataforma MEWiN dejamos todo tal y como estaba definido anteriormente. Si se compila para MEWiN se añade la nueva función "void leds_red_2(int o)" y se invierten las funciones de on y off ya que los leds en este caso se activan negados.

8. Prueba de funcionamiento de los LEDS en la placa MEWiN con las directivas de Contiki.

En este momento ya sería posible utilizar las interfaces proporcionadas por Contiki para llevar a cabo la activación de los leds, incluido el cuarto led de nuestra placa. Para comprobar el funcionamiento, se ha desarrollado una pequeña aplicación que hace uso de éstas funciones.

```
#include "contiki.h"

#include <io.h>

#include <signal.h>

#include <sys/unistd.h>
#include "msp430.h"
/*---------------------------------------------------------------------------*/
PROCESS(blink_process, "Blink");
AUTOSTART_PROCESSES(&blink_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(blink_process, ev, data)
{

 PROCESS_BEGIN();

 leds_init();
 leds_off(LEDS_ALL);
  while(1) {
   static struct etimer et;

   etimer_set(&et, 0.5*CLOCK_SECOND);
   PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
   leds_on(LEDS_RED_2 | LEDS_RED | LEDS_GREEN | LEDS_YELLOW);
   etimer_set(&et, 0.5*CLOCK_SECOND);
   PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
   leds_off(LEDS_ALL);

  }

PROCESS_END();

}
```

Para compilar el código y cargarlo en nuestro dispositivo se deben de seguir los siguientes pasos:

Compilamos el código como si fuera un mote Z1 indicando que nos cree el .ihex:

make 'nuestra_aplicacion' TARGET=MEWIN nombre.ihex

Ejecutamos 'dmesg' y vemos si ha reconocido el MSP430-FET430UIF y le ha asignado algún puerto.

Ejecutamos 'mspdebug'(x es el valor del puerto asignado):

mspdebug uif -j -d /dev/ttyUSBx

Cargamos el archivo .ihex:

prog nombre.ihex

Arrancamos el programa:

run

Resetear dispositivo:

reset.

En este ejemplo, se hace uso tanto de los timers proporcionados por Contiki como de las funciones para el manejo de los leds. El funcionamiento del dispositivo es el esperado. Cada 0.5 segundos se encienden todos los leds, permanecen 0.5 segundos encendidos y se vuelven a apagar a los 0.5 segundos, así sucesivamente. Se ha podido comprobar por tanto que es posible compilar nuestra plataforma y hacer uso de funciones sencillas para el manejo de los diferentes dispositivos de los que consta la placa.

El manejo de los leds se realiza de forma sencilla con sencillas llamadas a funciones, ésta era una de las funcionalidades básicas que se pretendían conseguir, tras esto se pasará a analizar el funcionamiento de la configuración y lectura del RTC, así como el almacenamiento y lectura de datos desde la tarjeta SD.

9. Características de las tarjetas SD.

Secure Digital (SD) es un formato de tarjeta de memoria Inventado por Panasonic. Se utiliza en dispositivos portátiles tales como cámaras fotográficas digitales, PDA, teléfonos móviles, ordenadores portátiles e incluso videoconsolas (tanto de sobremesa como la Wii o la PlayStation 3 (primeros modelos), como portátiles como la Nintendo 3DS), entre muchos otros.

Estas tarjetas tienen unas dimensiones de 32 mm x 24 mm x 2,1 mm. Existen dos tipos: unos que funcionan a velocidades normales, y otros de alta velocidad que tienen tasas de transferencia de datos más altas. Algunas cámaras fotográficas digitales requieren tarjetas de alta velocidad para poder grabar vídeo con fluidez o para capturar múltiples fotografías en una sucesión rápida.

Los dispositivos con ranuras SD pueden utilizar tarjetas MMC, que son más finas, pero las tarjetas SD no caben en las ranuras MMC. Asimismo, se pueden utilizar directamente en las ranuras de CompactFlash o de PC Card con un adaptador. Sus variantes MiniSD y MicroSD se pueden utilizar, también directamente, en ranuras SD mediante un adaptador.Las normales tienen forma de ◘. Hay algunas tarjetas SD que tienen un conector USB integrado con un doble propósito, y hay lectores que permiten que las tarjetas SD sean accesibles por medio de muchos puertos de conectividad como USB, FireWire y el puerto paralelo común. Las tarjetas SD también son accesibles mediante una disquetera usando un adaptador FlashPath.

Como la mayoría de los formatos de tarjeta de memoria, el SD está cubierto por numerosas patentes y marcas registradas, y sólo se puede licenciar a través de la Secure Digital Card Association (Asociación de la Tarjeta Secure Digital). El acuerdo de licencia actual de esta organización no permite controladores de código abierto para lectores de tarjetas SD, un hecho que genera consternación en las comunidades de código abierto y software libre. Generalmente, se desarrolla una capa de código abierto para un controlador SD de código cerrado disponible en una plataforma particular, pero esto está lejos de ser lo ideal. Otro método común consiste en utilizar el antiguo modo MMC, donde se requiere que todas las tarjetas SD soporten el estándar SD.

Esto significa que SD es menos abierto que CompactFlash o los llaveros USB, que pueden ser implementados libremente (aunque requieren costes de licencia por las marcas registradas y logotipos asociados, pero aun así resulta mucho más abierto que XD o Memory Stick, donde no hay disponible ni documentación pública ni implementación documentada).

Explicación técnica:

Todas las tarjetas de memoria SD y SDIO necesitan soportar el antiguo modo SPI/MMC que soporta la interfaz de serie de cuatro cables ligeramente más lenta (reloj, entrada serial, salida serial y selección de chip) que es compatible con los puertos SPI en muchos microcontroladores.

Muchas cámaras digitales, reproductores de audio digital y otros dispositivos portátiles, probablemente utilicen exclusivamente el modo MMC. La documentación para este modo se puede comprar en MMCA por 500,00 USD; sin embargo, la documentación parcial para SDIO es libre y existe documentación libre disponible para tarjetas de memoria como parte de las hojas de especificación de algunos fabricantes.

El modo MMC no proporciona acceso a las características propietarias de cifrado de las tarjetas SD y la documentación libre de SD no describe dichas características. La información del cifrado es utilizada primordialmente por los productores de medios y no es muy utilizada por los consumidores quienes típicamente utilizan tarjetas SD para almacenar datos no protegidos.

10. Bibliotecas para la comunicación con la tarjeta SD.

1. Función para la inicialización de la tarjeta SD.

```
void initMMC (void)
{

  //raise SS and MOSI for 80 clock cycles
 //SendByte(0xff) 10 times with SS high
 //RAISE SS
 int i;
 char response=0x10;

initSPI();
CS_HIGH();

 for(i=0;i<=9;i++)
   spiSendByte(0xff);

 CS_LOW();
 //Send Command 0 to put MMC in SPI mode
 mmcSendCmd(0x00,0,0x95);
 //Now wait for READY RESPONSE
 response = mmcGetResponse();
```

```
 /*
 while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
 UCA0TXBUF = 13;
 while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
 UCA0TXBUF = 13;
 while (!(IFG2&UCA0TXIFG));              // USCI_A0 TX buffer ready?
 UCA0TXBUF = response;
 */
 if(response != 0x01)
 {
  response = response;
 }
//LA RESPUESTA TIENE QUE SER 0X01
//       debug_printf("no responce");

 while(response != 0x00)
 {
//  debug_printf("Sending Command 1");
  CS_HIGH();
  spiSendByte(0xff);
  CS_LOW();
  //mmcSendCmd(0x01,0x00,0xff);
  //response=mmcGetResponse();
  mmcSendCmd(55,0x00,0xff);
  response=mmcGetResponse();
  CS_HIGH();
  spiSendByte(0xff);
  CS_LOW();
  mmcSendCmd(41,0x00,0xff);
  response=mmcGetResponse();
 }
 CS_HIGH();
 spiSendByte(0xff);
```

2. Función para la escritura de un bloque en la tarjeta SD.

Esta función permite escribir un bloque de datos en la tarjeta SD, la función toma como argumentos:

- buff: Puntero a los datos que queremos escribir.
- token: Token de datos para indicar el fin de los mismos.

Como en el caso anterior se puede comprobar, todo el código está comentado. Cabe destacar que en primer lugar se debe de enviar el token de datos (0xFD). Tras esto se envían los 512 bytes de datos, y al acabar se envía el CRC, en función de si la respuesta es la correcta(0x05) devolvemos TRUE o FALSE.

```
BOOL xmit_datablock (const BYTE *buff,BYTE token  )
{
        BYTE resp, wc;

        if (wait_ready() != 0xFF) return FALSE_t;

        xmit_spi(token);            //enviamos el token de fin de datos
        if (token != 0xFD) {        /* Token de datos?? */
                wc = 0;
                do {                /* Se envian los 512 bytes a la MMC */
                        xmit_spi(*buff++);
                        xmit_spi(*buff++);
                } while (--wc);

                xmit_spi(0xFF);             /* CRC (Dummy)*/
                xmit_spi(0xFF);
                resp = rcvr_spi();          /* Respuesta de la SD */
                if ((resp & 0x1F) != 0x05)          //Correcta??
                {
                        return FALSE_t;     //No
                }
        }
        return TRUE_t;                              //Si
}
```

3. Función para la lectura de un bloque de la tarjeta SD.

Esta función permite leer un bloque de datos en la tarjeta SD, la función toma como argumentos:

- buff: Puntero donde queremos guardar los datos que vamos a leer de la SD.
- btr: Numero de bytes que vamos a leer(debe ser un número par).

Como en el caso anterior, todo el código está comentado. Cabe destacar que en en este caso, no se indica en ningún momento el sector que se quiere leer, para ello se utiliza una función a más alto nivel, la cual enviará (antes de llamar a esta función) el comando a la SD indicando el sector que se desea leer.

```
BOOL rcvr_datablock (
        BYTE *buff,
        UINT btr
)
{
        BYTE token;
        Timer1 = 10;
        do {
/* Wait for data packet in timeout of 100ms */
                token = rcvr_spi();
        } while ((token == 0xFF) && Timer1);
        if(token != 0xFE) return FALSE_t;
```

```
/* If not valid data token, return with error */

        do {                        /* Receive the data block into buffer */
                *buff++ = rcvr_spi();
                *buff++ = rcvr_spi();
        } while (btr -= 2);
        rcvr_spi();                                 /* Discard CRC */
        rcvr_spi();

        return TRUE_t;                  /* Return with success */
}
```

Estas 3 funciones comentadas son las más representativas, sin embargo existen diversas funciones a más bajo nivel. Se ha considerado no explicar todas ellas detenidamente en el proyecto por el cuantioso tiempo que supondría, a continuación se describirán brevemente el resto de funciones a bajo nivel existentes:

- void initSPI(void);

  Inicializa la USART en modo SPI.

- unsigned char send_cmd (unsigned char cmd,unsigned long arg);

  Envía el comando especificado en cmd por el bus spi a la tarjeta SD. El argumento 'arg' es un valor que especificará el sector que se desea leer o escribir en caso de que el comando enviado sea de este tipo.

- unsigned char rcvr_spi (void);

  Tras el envío de un comando hacia la SD, debemos de llamar a rcvr_spi, para comprobar la respuesta del dispositivo y comprobar si existe algún problema

- void xmit_spi (unsigned char data);

  Es una función a muy bajo nivel que envía un byte por el puerto SPI, en este caso la llamada a send_cmd consta de varias llamadas a ésta función.

- char MMC_cardPresent (void);

  Esta función comprueba que la SD está insertada en el SLOT.

- char MMC_cardWriteProtected (void);

Esta función comprueba si la SD insertada tiene activada la protección contra escritura.

Además se han añadido dos funciones para habilitar y deshabilitar la SD cuando no sea necesario su uso:

```
#define SD_enable() P1SEL &= ~0x02,P1DIR |= 0x02,P1OUT &= ~0x02;

#define SD_disable() P1OUT |= 0x02;
```

11. Sistema de archivos FATfs

El paquete de librerías FATFs necesita una serie de funciones a alto nivel que le permitan manejar la SD y gestionar el sistema de archivos. Las funciones necesarias son las siguientes:

- Una función que inicialice la tarjeta SD y la ponga en modo SPI.

  La función "*initMMC()*" es totalmente válida sin necesidad de realizar ningún cambio.

- Funciones para lectura y escritura de bloques de datos en la SD.

  Se pueden usar las funciones "*rcvr_datablock*" y "*xmit_datablock*" sin realizar apenas cambios.

- Una función para enviar los comandos a la SD.

  La función "*send_cmd*" es adecuada para llevar a cabo esta labor.

- Una función para recibir los datos con las respuestas proporcionadas por la tarjeta SD.

  La función *"rcvr_spi"* cumple perfectamente con este propósito.

- Función que indique el estado de la SD señalando si está o no ocupada.

En este caso no disponemos de ninguna función adecuada para ello por esto la implementaremos:

```
BYTE wait_ready (void)
{
        BYTE res;

        rcvr_spi();
        do
                res = rcvr_spi();
        while ((res != 0xFF));
return res;
}
```

Simplemente esperamos hasta recibir un 0xFF desde la SD, lo cual indicará que está lista.

Éstas son todas las funciones necesarias para poder comenzar a portar el sistema FATFs. A continuación se explicarán los diferentes archivos por los que está formada la biblioteca FATfs y las funciones que implementa cada uno de esos archivos:

- "integer.h": Este archivo incluye las definiciones y tipos de datos usados de forma más común en la libreria.

```
typedef signed int      INT;
typedef unsigned int    UINT;
typedef signed char     CHAR;
typedef unsigned char   UCHAR;
typedef unsigned char   BYTE;
typedef signed short    SHORT;
typedef unsigned short  USHORT;
typedef unsigned short  WORD;
typedef signed long     LONG;
typedef unsigned long   ULONG;
typedef unsigned long   DWORD;
typedef enum { FALSE_t = 0, TRUE_t=1 } BOOL;
```

- "diskio.h": Esta biblioteca incluye las funciones de bajo nivel necesarias para que se pueda llevar a cabo la inicialización y manejo del sistema de archivos. En el nivel de aplicación final sólo deben ser usadas las funciones **disk_initialize** y **disk_status**, el resto de funciones son usadas por la biblioteca tff.h para llevar a cabo operaciones más complejas.

1. ***DSTATUS disk_initialize (BYTE Drive);***

Descripción: Inicializa la Unidad FAT.
Retorno: Estado de la inicialización.
Parámetro: Valor de la unidad que vamos a inicializar.

2. ***DSTATUS disk_status (BYTE Drive);***

Descripción: Devuelve el estado de la unidad.
Retorno: Estado de la inicialización.
Parámetro: Número de la unidad que queremos comprobar.

*El valor de retorno en ambas funciones es !(***STA_NOINIT**) en caso de que el dispositivo esté inicializado correctamente.
Nota: La función disk_initialize() sólo debe ser llamada una vez por la aplicación, si queremos volver a montar una unidad previamente desmontada, se ha de llamar a la función f_mount().

3. ***DRESULT disk_read (BYTE** Drive**,BYTE*** Buffer**,DWORD** SectorNumber**,BYTE** SectorCount **);***

Descripción: Lectura de un sector del disco.
Retorno: Resultado de la lectura.
Parámetros:

Drive: Número de la unidad.
Buffer: Puntero a array de Bytes donde se almacenará el dato leido.
SectorNumber: Especifica el sector a comenzar en el Bloque de Direcciones Lógicas.
SectorCount: Especifica el número de sectores a leer. Debe ser un valor entre 1 y 128.

4. ***DRESULT disk_write (BYTE** Drive**, const BYTE*** Buffer**,DWORD** SectorNumber**, BYTE** SectorCount**);***

Descripción: Escritura de un sector del disco.
Retorno: Resultado de la lectura.
Parámetros:

Drive: Número de la unidad.
Buffer: Puntero a array de Bytes que se quiere escribir.
SectorNumber: Especifica el sector a comenzar en el Bloque de Direcciones Lógicas.
SectorCount: Especifica el número de sectores a escribir. Debe ser un valor entre 1 y 128.

5. ***DRESULT disk_ioctl (BYTE** Drive**,BYTE** Command**, void*** Buffer **);***

Descripción: Usada en la función de sincronización de la unidad FAT.
Retorno: Resultado de la consulta.
Parámetros:

Drive: Número de la unidad.
Command: Comando a realizar.
Buffer: Buffer dependiente del comando usado, será NULL en nuestro caso.

DRESULT:

RES_OK (0):
Resultado OK.
RES_ERROR:
Error durante la operación solicitada.
RES_PARERR:
Parámetro inválido.
RES_NOTRDY:
El dispositivo no se ha inicializado.

- "tff.h": Contiene todas la funciones que se van a utilizar para escribir, leer, borrar, etc. en los ficheros.

1. ***FRESULT f_mount (BYTE** Drive**, FATFS*** *FileSystemObject* ***);***

Descripción: Registra o deja de registrar un área de trabajo para el sistema FATfs.
Retorno: Resultado del proceso de montaje.
Parámetros:

Drive: Número de la unidad.
FileSystemObject: Puntero al área de trabajo.

FRESULT:

FR_OK (0)
Función ejecutada satisfactoriamente.
FR_INVALID_DRIVE
El número de dispostivo es incorrecto.

Para dejar de registrar un área de trabajo determinada(DRIVE) se debe especificar el FileSystemObject a NULL.
No se accede a la tarjeta en ningún momento, únicamente se inicializa el area de trabajo y registra su dirección en la tabla interna.

2. ***FRESULT f_open (FIL**** *FileObject**, const TCHAR**** *FileName**, BYTE* *ModeFlags**);***

Descripción: Crea un Objeto de tipo fichero que puede ser usado para acceder al fichero.
Retorno: Resultado del proceso.
Parámetros:

FileObject: Puntero al objeto de estructura de ficheros a ser creado.

Filename: Puntero a la cadena de caracteres que contendrá el nombre del fichero a ser creado o abierto.

ModeFlags:

(Se pueden combinar varios flags con el operador binario "|")

Valor	Descripción
FA_READ	Especifica la lectura del fichero.
FA_WRITE	Especifica la escritura del fichero.
FA_OPEN_EXISTING	Abre el fichero. Si no existe devuelve error.
FA_OPEN_ALWAYS	Abre el fichero. Si no existe se crea. Usar en conjunción con f_lseek .
FA_CREATE_NEW	Crea un nuevo fichero.Devuelve FR_EXIST si ya existe el fichero indicado.
FA_CREATE_ALWAYS	Crea un nuevo fichero. Si ya existe lo trunca y lo sobreescribe.

FRESULT:

FR_OK (0)
La función se realizó correctamente. El objeto creado es correcto.
FR_NO_FILE
No se encuentra el fichero.
FR_NO_PATH
No se encuentra la ruta.
FR_INVALID_NAME
El nombre del fichero es inválido.
FR_INVALID_DRIVE
El número de la unidad es incorrecto.
FR_EXIST
El fichero ya existe.
FR_DENIED
Acceso denegado, diversos motivos:

- Intento de escribir/crear un fichero de sólo lectura.
- La tabla de directorios está completa.

FR_NOT_READY
Disco/Unidad no disponible.
FR_WRITE_PROTECTED
Dispositivo protegido contra escritura.
FR_DISK_ERR
Fallo al ejecutar la función debido a un error en el dispositivo.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_ENABLED
No está habilitada el area de trabajo en el dispositivo.
FR_NO_FILESYSTEM
No hay un volumen FAT válido en el dispositivo.
FR_LOCKED
Función rechazada debido a la politica de compartición de archivos.

3. ***FRESULT f_close (FIL****FileObject***);***

Descripción: Cierra un fichero abierto previamente.
Retorno: Estado de la acción.
Parámetro: FileObject que se quiere cerrar.

FRESULT:

FR_OK (0)
FileObject cerrado correctamente.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.

4. ***FRESULT f_read (FIL****FileObject***, void****Buffer***,UINT***ByteToRead***, UINT****ByteRead***);***

Descripción: Lee datos de un fichero.
Retorno: Estado de la acción.
Parámetros:
FileObject: FileObject que se quiere cerrar.
Buffer: Puntero al buffer para almacenar los datos leidos.
ByteToRead: Número de Bytes a leer.
ByteRead: Puntero a la variable para almacenar el número de Bytes leidos del fichero.
FRESULT:

FR_OK (0)
Lectura correcta.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.

5. ***FRESULT f_write (FIL**** *FileObject**, const void**** *Buffer**,UINT* *ByteToWrite**, UINT**** *ByteWritten**)***

Descripción: Escribe los datos de un fichero.
Retorno: Estado de la acción.
Parámetros:
FileObject: FileObject que se quiere cerrar.
Buffer: Puntero al buffer con los datos que se quieren escribir.
ByteToWrite: Número de Bytes a escribir.
ByteWritten: Puntero a la variable para almacenar el número de Bytes escritos en el fichero.
FRESULT:

FR_OK (0)
Lectura correcta.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.

6. F R E S L

T f_lseek (FIL* FileObject,DWORD Offset);

Descripción: Mueve el puntero de lectura/escritura de un FileObject abierto una cantidad determinada por Offset.

Retorno: Estado de la acción.
Parámetros:
FileObject: FileObject del cual se quiere modificar su puntero de lectura/escritura.
Offset: Cantidad de Bytes que se quiere desplazar el puntero.
FRESULT:

FR_OK (0)
Lectura correcta.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.

FR_INVALID_OBJECT
El FileObject es inválido.
FR_NOT_ENOUGH_CORE
Tamaño insuficiente de la tabla de enlaces para el fichero seleccionado.

7. FRESULT f_sync (FIL* FileObject);

Descripción: Borra la caché almacenada correspondiente a un fichero.
Retorno: Estado de la acción.
Parámetros:
FileObject: Puntero al FileObject que se quiere borrar su caché.
FRESULT:

FR_OK (0)
Lectura correcta.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.

8. F

ESULT f_unlink (const TCHAR* *FileName***);**

Descripción: Elimina un objeto.
Retorno: Estado de la acción.
Parámetros:

FileObject: Puntero al FileObject que se quiere borrar.

FRESULT:

FR_OK (0)
Lectura correcta.
FR_NO_FILE
No se puede encontrar el fichero.
FR_INVALID_DRIVE
El número de la unidad es inválido.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.
FR_WRITE_PROTECTED
El medio tiene protección contra escritura.
FR_NOT_ENABLED
El disco lógico o tiene área de trabajo. **FR_NO_FILESYSTEM**
No hay un volumen FAT válido en el dispositivo.
FR_LOCKED
Función rechazada debido a la política de compartición de archivos.

9. ***FRESULT f_rename (const TCHAR**** *OldName*, ***const TCHAR**** *NewName*);**

Retorno: Estado de la acción.
Parámetros:

OldName: Cadena de caracteres que queremos cambiar.
NewName: Nuevo valor de la cadena de caracteres.

FRESULT:

FR_OK (0)
Lectura correcta.
FR_NO_FILE
No se puede encontrar el fichero.
FR_INVALID_DRIVE
El número de la unidad es inválido.
FR_DENIED
Acceso denegado, el fichero se ha intentado abrir sin haber seleccionado el modo lectura.
FR_DISK_ERR
Error en el disco.
FR_INT_ERR
Error interno o fallo en la estructura FAT.
FR_NOT_READY
Disco/Unidad no disponible.
FR_INVALID_OBJECT
El FileObject es inválido.
FR_WRITE_PROTECTED
El medio tiene protección contra escritura.
FR_NOT_ENABLED
El disco lógico o tiene área de trabajo. **FR_NO_FILESYSTEM**
No hay un volumen FAT válido en el dispositivo.
FR_LOCKED
Función rechazada debido a la política de compartición de archivos.
FR_INVALID_NAME
El nombre del fichero es inválido.
FR_EXIST
El nuevo nombre está colisionando con uno ya existente en la unidad.

Comprobación del funcionamiento de la tarjeta SD usando el sistema de archivos FAT16.

El sistema FAT 16 cumple con los requisitos, ya que soporta un tamaño máximo de archivo de 2GBytes , coincidiendo con la capacidad máxima de la tarjeta SD que podemos utilizar, de acuerdo a las condiciones que se explicaron previamente. Además este sistema de archivos permite un número máximo de archivos de 65.517. Siendo más que suficiente para nuestro propósito de usar nuestra dispositivo como un datalogger.

Para conseguir usar las bibliotecas Fatfs necesitamos añadirlas a nuestro directorio, para ello, se copian todos los archivos dentro de la carpeta *dev* en el directorio *"/platform/mewin/"* . Y se añaden las siguientes lineas al archivo Makefile dentro de la carpeta "platform/mewin":

```
ARCH=msp430.c leds.c watchdog.c uart0-putchar.c uart0x.c mmc.c
tff.c\
```

12. Prueba de funcionamiento del sistema de archivos FATfs.

A continuación se presentará y explicará un código de pruebas que permita comprobar el correcto funcionamiento de nuestro dispositivo:

```
#include "contiki.h"
#include "dev/leds.h"
#include "diskio.h"
#include "tff.h"
#include "mmc.h"


/*---------------------------------------------------------------------------*/
PROCESS(blink_process, "Blink");
AUTOSTART_PROCESSES(&blink_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(blink_process, ev, data)
{
 //PROCESS_EXITHANDLER(goto exit;)
 PROCESS_BEGIN();

 leds_init();

 leds_off(LEDS_ALL);

 static struct etimer et;

 printf("Hola 1");

FIL F;
FRESULT fres;
WORD bytesWritten;
WORD bytesRead;
char buf1[]="Esto es la prueba con CONTIKI\n";
char buf2[200];
//Anado, File handling variables
FATFS fatfs;
unsigned char res;
unsigned char max_tries=5;
const char filename[] = "fftest2.txt";
sd_enable();
res = disk_initialize(0);
if(!(res==STA_NOINIT))
{
      if ((fres = f_mount(0, &fatfs)) == FR_OK)
      {
       fres = f_open (&F, filename, FA_WRITE | FA_OPEN_ALWAYS);
       if (fres == FR_OK)
          {
           fres = f_lseek(&F, (&F)->fsize);
           if(fres != FR_OK)
             leds_on(LEDS_RED);
```

```
          else
          {
          fres = f_write (&F, buf1, sizeof (buf1) - 1, &bytesWritten);
          if (fres != FR_OK)
            leds_on(LEDS_YELLOW);
          }
          f_sync (&F);
          f_close (&F);
         }
       fres = f_open (&F, filename, FA_READ );
       if (fres == FR_OK)
       {
            //Pongo el puntero en la zona donde se comenzÃ³ a escribir por última vez
            fres = f_lseek(&F, (&F)->fsize - (sizeof(buf1)-1));

            if(fres != FR_OK)
              leds_on(LEDS_GREEN);
            else
            {
              fres = f_read (&F, buf2, sizeof(buf1), &bytesRead);
              if (fres != FR_OK)
                leds_on(LEDS_RED_2);

              f_sync (&F);
              f_close (&F);
            }
       }
       if (memcmp (buf1, buf2, sizeof(buf1)-1) == 0)
       {
          //Lectura correcta
           leds_on(LEDS_GREEN|LEDS_YELLOW);
       }
       else
       {
          //Fallo lectura
          leds_on(LEDS_RED_2|LEDS_RED);
       }
       //Renombramos el archivo
       fres = f_rename(&filename[0], "newname.txt");
       /*
       //Para Borrar algun archivo
       fres = f_unlink(&filename[0]);
       */
           }
}
  printf("Hola 2");
  etimer_set(&et, 3*CLOCK_SECOND);
  PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
  leds_off(LEDS_ALL);
  sd_disable();
 /*
exit:
  leds_off(LEDS_ALL);
*/
PROCESS_END();
}
```

A continuación se comentará brevemente el funcionamiento del programa anterior:

- En primer lugar se incluyen las cabeceras de los archivos, para hacer uso de los LEDs, las bibliotecas básicas de Contiki y las funciones para el manejo de la tarjeta SD con el sistema FAT16.

```
#include "contiki.h"
#include "dev/leds.h"
#include "diskio.h"
#include "tff.h"
#include "mmc.h"
```

- Se inicializan los leds y el timer que se van a usar para temporizar.

```
leds_init();
leds_off(LEDS_ALL);
static struct etimer et;
```

- Se inicializan las variables necesarias para el sistema de archivos, así como las cadenas de caracteres que contendrán el nombre del archivo que se va a crear(filename) , la cadena con los datos que se van a escribir(buf1) y la cadena de caracteres para almacenar los datos leidos(buf2).

```
FIL F;
FRESULT fres;
WORD bytesWritten;
WORD bytesRead;
char buf1[]="Esto es la prueba con CONTIKI\n";
char buf2[200];
//Anado, File handling variables
FATFS fatfs;
unsigned char res;
unsigned char max_tries=5;
const char filename[] = "DATOS.txt";
```

- Se inicializa el sistema FatFs:

```
res = disk_initialize(0);
```

- Si la inicialización es correcta, se escribe en la SD. Para ello:

    1. Se sitúa el puntero de escritura en el último carácter escrito:

    ```
    fres = f_lseek(&F, (&F)->fsize - (sizeof(buf1)-1));
    ```

    2. Se encienden los LEDS para comprobar:

```
if(fres != FR_OK)
    leds_on(LEDS_GREEN);
```

3. Se escribe en el fichero los datos almacenados en la variable buf1:

```
f_write (&F, buf1, sizeof (buf1) - 1, &bytesWritten);
```

4. Se sincroniza el fichero y se cierra:

```
f_sync (&F);
f_close (&F);
```

- A continuación se comprueba que lo que se ha escrito corresponde con lo que se va a leer del fichero, para ello:

1. Se pone el puntero de escritura en el primer caracter escrito:

```
fres = f_lseek(&F, (&F)->fsize);
```

2. Se encienden los LEDS para comprobar:

```
if(fres != FR_OK)
    leds_on(LEDS_RED);
```

3. Se lee del fichero los datos almacenados en la variable buf1:

```
fres = f_read (&F, buf2, sizeof(buf1), &bytesRead);
```

4. Se sincroniza el fichero y se cierra:

```
f_sync (&F);
f_close (&F);
```

5. Se comprueba que la lectura es la correcta, comparándo el valor leido con el valor inicial y encendemos los leds para detectar un error.

```
if (memcmp (buf1, buf2, sizeof(buf1)-1) == 0)
      {
         //Lectura correcta
          leds_on(LEDS_GREEN|LEDS_YELLOW);
      }
     else
      {
         //Fallo lectura
         leds_on(LEDS_RED_2|LEDS_RED);
      }
```

13. Bibliotecas creadas para el funcionamiento del RTC en Contiki.

Dentro de la carpeta "*platform/mewin/dev/",* se ha creado una biblioteca llamada "i2cmaster.c" para gestionar la escritura y lectura de datos del RTC de forma sencilla. La biblioteca consta de las siguientes funciones:

void i2c_init():

Inicializa el Bus I2C en el microcontrolador.

```
void
i2c_init() {

  UCB0CTL1 = UCB0CTL0 = 0;
  //AsignaciÃ³n de pines del I2C
  P3SEL |= 0x06;
  //Deshabilitar la USCI_B0
  UCB0CTL1 |= UCSWRST;
  //Configurar un I2C master en modo sÃncrono
  UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;
  //Configurar la frecuencia de 100KHz
  UCB0CTL1 = UCSSEL_2 + UCSWRST;
  //UCB0BR0 = I2C_PRESC_400KHZ_LSB;
  UCB0BR0 = 16;
  UCB0BR1 = 0;
  //Configurar la direcciÃ³n del slave...104d
  UCB0I2CSA = 0x68;
  //Habilitar la USCI_B0
  UCB0CTL1 &= ~UCSWRST;
}
```

u8_t i2c_busy(void):

Comprueba si está ocupado el bus.

```
u8_t
i2c_busy(void) {
  return (UCB0STAT & UCBBUSY);
}
```

void read_RTC(u8_t *rx_buf):

Lee datos del RTC y los guarda en rx_buf, esta función llama internamente a otra más compleja, el valor 7 corresponde a los siete registros que se desean leer y que corresponden a los: segundos, minutos, horas, dia de la semana, día del mes,mes y año

```
u8_t
```

```
read_RTC(u8_t *rx_buf) {

i2c_receive_n (7, rx_buf);

}
```

void write_RTC(u8_t *tx_buf):

Escribe los registros del RTC, en función del puntero que se le pase como argumento. El puntero tiene en este caso 8 posiciones ya que la primera corresponde a la dirección del primer registro que se desea escribir.

```
void
write_RTC(u8_t *tx_buf) {

i2c_transmit_n (8, tx_buf);

}
```

i2c_enable(), i2c_disable():

Que habilitan y deshabilitan respectivamente el chip del RTC cuando no sea necesario su uso y evitar un consumo extra innecesario.

```
#define i2c_enable() P1SEL &= ~0x02,P1DIR |= 0x02,P1OUT &=
~0x02;

#define i2c_disable() P1OUT |= 0x02;
```

14. Prueba de funcionamiento completo del sistema usando la SD y el RTC.

Con este ejemplo se pretende comprobar que la placa MEWiN es capaz de funcionar como un datalogger, es decir gracias a la tarjeta SD y el RTC, el equipo es capaz de guardar en un archivo de texto plano , la hora en la que se tomó la medida de un sensor determinado y la medida correspondiente. Para el posterior uso y análisis de los datos tomados. De esta forma se pretende confirmar que es dispositivo es totalmente funcional y que todas las bibliotecas realizadas en Contiki funcionan correctamente.

Inicializamos las variables a utilizar:

```
PROCESS_THREAD(blink_process, ev, data)
{
 //PROCESS_EXITHANDLER(goto exit;)
 PROCESS_BEGIN();

 leds_init();
 leds_off(LEDS_ALL);
 static struct etimer et;
FIL F;
FRESULT fres;
WORD bytesWritten;
WORD bytesRead;
char buf2[38];
char buf1[38];

//Anado, File handling variables
FATFS fatfs;
unsigned char res;
unsigned char max_tries=5;
const char filename[] = "fftest2.txt";
int i;
```

Se inicializa la variable del buffer donde se guardarán los datos leidos del RTC:

```
u8_t buffer1[]= { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

Se inicualiza un array con la hora y fecha actuales(se puede comentar una vez que se ha escrito por primera vez en el RTC):

```
 //Fijo hora en caso de que sea necesario
 u8_t u8HoraFecha [8];
 u8HoraFecha[0] = 0;
 u8HoraFecha[1] = 0x00;  //segundos
 u8HoraFecha[2] = 0x35;  //Minutos
 u8HoraFecha[3] = 0x10;  //Horas
 u8HoraFecha[4] = 0x01;  //dÃa de la semana
```

```
u8HoraFecha[5] = 0x14;  //dÃa del mes
u8HoraFecha[6] = 0x03;  //mes
u8HoraFecha[7] = 0x11;  //aÃ±o
```

Se escribe en el RTC la hora actual(se debe comentar una vez que se ha escrito por primera vez en el RTC):

```
i2c_enable();
i2c_init();
write_RTC(&u8HoraFecha[0]);
while (i2c_busy());
```

Se lee del RTC la hora actual y la enviamos por el puerto serie:

```
CS_HIGH();
i2c_enable();
i2c_init();
read_RTC(&buffer1[0]);
while (i2c_busy());
for (i=0;i<7;i++)
        PRINTF ("Registro Buffer %d:%X\n",i,buffer1[i]);
CS_LOW();
i2c_disable();
```

Se activa la hora actual enla variable buf1:

```
memset (buf1,' ',38);
 sprintf (buf1, "Registro Buffer.%02X:%02X:%02X-%02X-
%02X/%02X/%02X\n",buffer1[2],buffer1[1],buffer1[0],buffer1[3],buffer1[4],buffer1[5],buffer1[6]
);
```

Se activa la SD y se escribe el valor de la hora:

```
ActivoSD();
//leds_on(LEDS_RED);
res = disk_initialize(0);
//leds_on(LEDS_RED_2);
if(!(res==STA_NOINIT))
{
          if ((fres = f_mount(0, &fatfs)) == FR_OK)
          {

      //leds_on(LEDS_GREEN);
      fres = f_open (&F, filename, FA_WRITE | FA_OPEN_ALWAYS);
      if (fres == FR_OK)
        {
         fres = f_lseek(&F, (&F)->fsize);
         if(fres != FR_OK)
           leds_on(LEDS_RED);
         else
         {
```

```
    fres = f_write (&F, buf1, sizeof (buf1) - 1, &bytesWritten);
    if (fres != FR_OK)
      leds_on(LEDS_YELLOW);
    }
    f_sync (&F);
    f_close (&F);
   }
```

Se comprueba que la lectura/escritura en la SD es correcta:

```
fres = f_open (&F, filename, FA_READ );
if (fres == FR_OK)
{
     //Pongo el puntero en la zona donde se comenzÃ³ a escribir por ultima vez
     fres = f_lseek(&F, (&F)->fsize - (sizeof(buf1)-1));

     if(fres != FR_OK)
       leds_on(LEDS_GREEN);
     else
     {
       fres = f_read (&F, buf2, sizeof(buf1), &bytesRead);
       if (fres != FR_OK)
         leds_on(LEDS_RED_2);

       f_sync (&F);
       f_close (&F);
     }
}


    if (memcmp (buf1, buf2, sizeof(buf1)-1) == 0)
{
   //Lectura correcta
    leds_on(LEDS_GREEN|LEDS_YELLOW);
}
else
{
   //Fallo lectura
   leds_on(LEDS_RED_2|LEDS_RED);
}

 /*
//Renombramos el archivo
fres = f_rename(&filename[0], "newname.txt");
/*
//Para Borrar algun archivo
fres = f_unlink(&filename[0]);
*/
    }
```

}

Se desactiva la SD y finaliza la ejecución del programa (en 3 segundos).

```
DesactivoSD();
leds_on(LEDS_YELLOW);
printf("Hola 2");
etimer_set(&et, 3*CLOCK_SECOND);
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
leds_off(LEDS_ALL);
```

15. Módulo de Radio CC2420 en Contiki.

En este aparatado se pretende analizar las diferentes bibliotecas que proporciona Contiki para utilizar este módulo de radio.

Dentro del directorio "core/dev/" (ver Figura A-5) se encuentran los diferentes archivos que actúan de drivers para permitir a las funciones de nivel superior comunicarse con el módulo de radio:

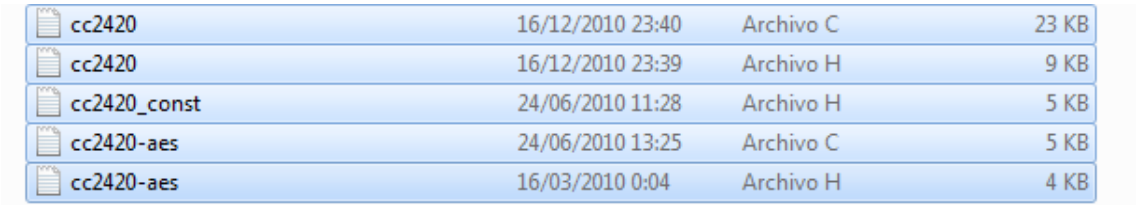


Figura A-4: Archivos del módulo de radio CC2420 en Contiki

- "cc24020.c":

  Este archivo contiene la mayoría de funciones a bajo nivel que es preciso modificar.
  Entre ellas, se destacarán las más importantes:

  1. La función para inicializar el módulo de radio.
  2. La función para encender el módulo de radio.
  3. La función para apagar el módulo de radio.
  4. El proceso "cc2020_process" que se ejecutará en un hilo independiente para ejecutarse en conjunto con la aplicación de usuario.
  5. La función de lectura del módulo de radio.
  6. La función de escritura en el módulo de radio.

- "cc2420":
  Es el archivo de cabecera del "cc2420.c", contiene las definiciones para comunicarse con el módulo de radio.

- "cc24const.h":
  Esta archivo de cabecera, contiene las diferentes definiciones de los comandos a enviar al módulo de radio, así como el valor de los registros del mismo.

- "cc2420aes.c" y "cc2420.h":
  Estos dos archivos contienen las funciones que permiten dotar a la conexión de encriptación aes.

16. Inicialización del módulo de radio CC2520 en Contiki.

Como se ha comentado, para llevar a cabo la inicialización del módulo de radio de la placa MEWiN en Contiki, es necesario tener conocimiento de los puertos (ver Figuras A-6 y A-7) a los que conectan el módulo de radio al microcontrolador.

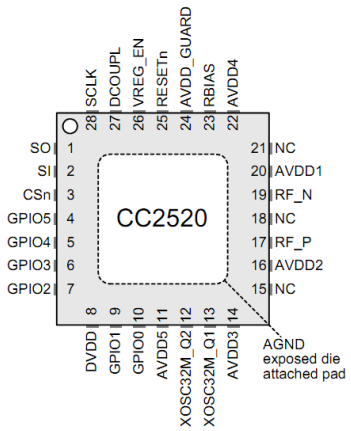


Figura A-5: Puertos del módulo de radio CC2520

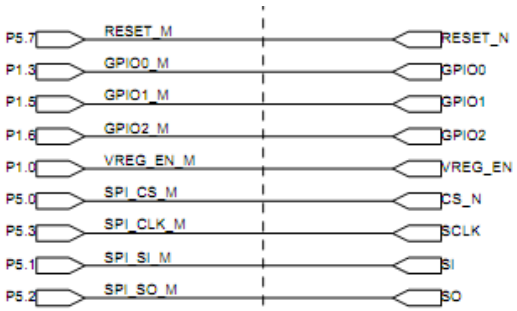


Figura A-6: Conexiones al módulo de radio en la placa MEWiN.

A continuación se describen las conexiones que permiten conectan al módulo de radio con el microcontrolador:

- GPIO0-2:
  Entradas/Salidas de interrupción del módulo de radio.

- RESET_N:
  Entrada hacia el módulo de radio que permite resetear el mismo.

- VREG_EN:
  Sirve para habilitar el regulador de voltaje en el módulo de radio, que se habilitará en corrientes superiores a 32 mA.

- SCLK,SI,SO,CS_N:
  Son las conexiones usadas por el protocolo SPI para llevar a cabo la comunicación con el periférico(ver Capítulo 2, Apartado 13.1).

Para llevar a cabo la inicialización del módulo de radio CC2520, se ha utilizado como punto de partida la inicialización que ese hace del mismo en la pila ZigBee de Texas Instruments. A continuación se expone una aplicación de ejemplo que inicializa el módulo de radio en Contiki y activa uno de los LEDs en caso de que la inicialización se haya realizado correctamente:

```
#include "contiki.h"
#include "dev/leds.h"
#include <signal.h>


#if 1
#define PRINTF(...) printf(__VA_ARGS__)
#else
#define PRINTF(...)
#endif
```

```
#define CC2520_MDMCTRL0 0x046
#define CC2520_MDMCTRL1 0x047

typedef struct {
u8_t reg;
u8_t val;
}regVal_t;

static regVal_t regval[] =
{
CC2520_MDMCTRL0,0x85,
CC2520_MDMCTRL1,0x14,
0,0x00
};


PROCESS_THREAD(blink_process, ev, data)
{

  PROCESS_BEGIN();

  regVal_t* p;
  u8_t val1,val2;
  val1=0;
  val2=0;


  int s = splhigh();                  /* Deshabilitacion de  las interrupciones */
    UCB1CTL1 = 0;                     //Inicializo el BUS SPI
    UCB1CTL0 = 0;
    UCB1CTL1 |= UCSWRST;
    UCB1CTL0 |= UCMST | UCSYNC | UCCKPH | UCMSB;
    UCB1CTL1 |= UCSSEL0|UCSSEL1;
    UCB1BR0  = 0;
    UCB1BR1  = 0;
    P5DIR |= BV(0);
    P5SEL |= BV(1)|BV(2)|BV(3);
    //asm("NOP"); asm("NOP"); asm("NOP"); asm("NOP"); P5OUT |=BV(0);
    UCB1CTL1 &= ~UCSWRST;

  //Bus SPI
  P5OUT &=~BV(7);//Activo Reset
  P1OUT &=~BV(0);//Activo Vreg
  P5DIR |= BV(7);//Salida Reset
  P1DIR |= BV(0);//Salida Vreg

  P1SEL &=~BV(3);// Configuracion del puerto de la interrupción
  P1OUT &=~BV(3);
  P1DIR &=~BV(3);

splx(s);              /* Re-enable interrupts. */

  P1IE &=~BV(3);//Se Desabilitan las Interrupciones
  P5OUT &=~BV(7);//Se Activa el Reset
  P5OUT |=BV(0);//Bus SPI
```

```
 P1OUT &=~BV(0);//Activo Vreg

//Espero 1100us/////////
halMcuWaitUs(1100);

P5DIR &=~ BV(2);
P1OUT |= BV(0);

//Espera 200us
halMcuWaitUs(200);//200
//Activo Reset
P5OUT |= BV(7);

u8_t i;
i=100;
P5OUT &=~ BV(0);
while(i>0 && !(P5IN & BV(2)))
{
//clock_delay(10);
halMcuWaitUs(10);
--i;
}
//Se espera hasta que el modulo de radio esté listo para //comunicarse con el
//Se activa la señal CS del bus SPI
P5OUT |= BV(0);
if(i<=0)
        leds_on(LEDS_RED_2);
else
{
leds_on(LEDS_YELLOW);
p=regval;
while(p->reg!=0)
{
CC2520_MEMWR8(p->reg,p->val);
p++;
//Se envían los datos de los registros por SPI
}
//Se lee el registro
val2 = CC2520_MEMRD8(CC2520_MDMCTRL0);
PRINTF("VALOR LEIDO%c",val2);

if (val2==0x85)   //Se comprueba el valor
        leds_on(LEDS_GREEN);
else
        leds_on(LEDS_RED);
}
PROCESS_END();
}
```

Todo el código anterior se encuentra comentado para una mejor comprensión del mismo. Realizando un resumen :

- Se inicializa la interfaz SPI y se deshabilitan las interrupciones.

- Se establece se establece la comunicación con el módulo de radio.
- Se envían los valores de los registros de inicialización del módulo de radio.
- Se lee uno de los registros del módulo de radio.
- Se comprueba que el valor coincide con el que hemos escrito.
- Se enciende el LED verde en caso afirmativo.