Proyecto de Sistemas Informáticos

Título:

Buzz: Desarrollo de juego multiusuario y periféricos utilizando Zigbee para su control.

Autores:

Esteban Alcalde González
José Antonio López García de Paredes
Carlos Valencia Rey

Director:

Luis Piñuel Moreno

Curso:

2006-2007

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

1.INTRODUCCIÓN

1.1 Motivaciones

Zigbee es una nueva tecnología cuyo objetivo es ofrecer otra posibilidad para el desarrollo de aplicaciones inalámbricas. Gracias a su particular arquitectura permite la interactuación de un gran número de dispositivos, lo que es de capital importancia si se pretende desarrollar un entorno multijugador o multiusuario.

Es por esto último por lo que se decidió afrontar el desarrollo del proyecto (cuyo objetivo final es un juego multijugador) utilizando esta nueva tecnología, pero no solo por las bondades de sus características técnicas, que se explicarán en apartados posteriores, sino también por el interés que plantea el aprender a desarrollar aplicaciones (de una manera bastante rápida gracias a la sencillez que ofrece en su programación) con la que posiblemente se convertirá en referencia para muchos otros campos relacionados con la informática o la electrónica, tanto por su bajo coste, su bajo consumo (gracias a la capacidad de dejar dormidos los dispositivos que se encuentran inactivos) como por cubrir servicios no resueltos (emisión de datos con tasas de pequeñas de transferencia), por tecnologías como bluetooth o wifi.

En el presente documento trataremos de explicar de la mejor manera posible el proceso llevado a cabo para aprovechar al máximo todos los puntos fuertes de Zigbee en la elaboración de nuestro proyecto. Comentaremos los distintos problemas que hemos encontrado y como los hemos solucionado, con el objetivo de que sirva de ayuda a futuros interesados.

1.2 Resumen

Español

Hemos organizado la memoria en cinco capítulos: introducción, arquitectura de Zigbee, entorno de desarrollo, desarrollo y conclusiones.

En el capítulo uno, la introducción, a parte de las motivaciones y de este resumen, ofrecemos una pequeña descripción de las distintas tecnologías con funcionalidad parecida a Zigbee como son los infrarrojos, el bluetooth o el wi-fi. A continuación en el punto llamado comparativa, observamos las similitudes y diferencias de estas tres respecto a a la que a nosotros nos atañe, Zigbee.

En el capítulo dos, arquitectura de Zigbee, ofreceremos una descripción detallada de su arquitectura, hablando de la pila que implementa, de las características del IEEE 802.15.4. , de las distintas topologías de red existentes, de los distintos dispositivos necesarios u opcionales, de los requisitos hardware, de cómo funcionan las comunicaciones, de los perfiles, las tecnologías actuales y de posibles aplicaciones.

En el capítulo tres, entorno de desarrollo, daremos una explicación acerca de las herramientas que hemos utilizado para hacer el proyecto, estas son el mplab ICD2 para generar el código, compilarlo y depurarlo, el picdemZ que es donde se cargan los programas generados, eclipse para el desarrollo de la aplicación, los puerto serie para la comunicación con el ordenador y por último los distintos problemas encontrados.

En el capítulo cuatro, desarrollo, explicaremos a fondo tanto el software del pc como el de las placas. En el del PC explicaremos el funcionamiento del buzz, el RS 232 para la comunicación, la funcionalidad, la implementación y las dificultades y soluciones. En el de las placas, a parte de la funcionalidad, implementación y de las dificultades que hemos encontrado y como las hemos solucionado, hablaremos de las dos versiones de la pila que hemos utilizado para implementar la topología malla y la estrella y de las diferencias que tienen entre si.

En el capitulo cinco, conclusión, expondremos las conclusiones que hemos ido obteniendo durante el desarrollo del proyecto, así como de posibles aplicaciones para zigbee en el futuro y de la relevancia que podrá tener en años venideros.

Inglés

We have organized the memory in five chapters: input, architecture of Zigbee, development environment, development and conclusions.

In the first chapter one, the input, to part of the motivations and of this resume, we offer a small description of the different technologies with similar capability to Zigbee like they are infrareds, bluetooth or wi-fi. Next in the called comparative point, we observe the similarities and differences of three regarding to the one that concerns us, Zigbee.

In the chapter two, architecture of Zigbee, we will offer a detailed description of their architecture, speaking of the battery that implements, of the features of IEEE 802.15.4. , of the existing different network topologies, of the necessary or optional different devices, of the requirements hardware, of how the communications work, of the profiles, the current technologies and of possible applications.

In the chapter three we will give an explanation about the tools that we have used to make the project, these they are the mplab ICD2 to generate the code, to compile it and to debug it, the picdemZ that is where the generated programas are loaded, eclipse for the development of the application, the port series for the communication with the computer and lastly the opposing different problems.

In the chapter four, I develop, we will thoroughly explain so much the software of the pc like that of the boards. In that of the PC we will explain the operation of the buzz, RS 232 for the communication, the capability, the implementation and the difficulties and solutions. In that of the boards, to part of the capability, implementation and of the difficulties that we have found and as we have solved them, we will speak of the two versions of the battery that we have used to implement the topology mesh and the star and of the differences that have among if.

In the chapter five, conclusion, we will expose the conclusions that we have gone obtaining during the development of the project, as well as of possible applications for zigbee in the future and of the relevance that he/she will be able to have in coming years.

1.3 ¿Qué es Zigbee?

ZigBee es un protocolo de comunicaciones inalámbrico, similar al bluetooth, y basado en el estándar para redes inalámbricas de área personal (WPANs) IEEE_802.15.4. Surge del fruto de una alianza, sin ánimo de lucro, de más de 200 empresas, la mayoría de ellas fabricantes de semiconductores, con el objetivo de conseguir el desarrollo e implantación de una tecnología inalámbrica de bajo coste.

Con la participación de empresas del renombre de Invensys, Mitsubishi, Honeywell, Philips y Motorola nace Zigbee con el objetivo de crear un sistema estándar de comunicaciones, vía radio y bidireccional, que permita cubrir el vacío que se produce por debajo del Bluetooth.

Entre las características de Zigbee cabe destacar debido a su bajo consumo, su sistema de comunicaciones vía radio (con topología malla) y su fácil integración (se pueden fabricar nodos con muy poca electrónica).

1.4 Alternativas

1.4.1 Bluetooth

Bluetooth es el nombre común de la especificación industrial **IEEE 802.15.1**, que define un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura, globalmente y sin licencia de corto rango. La tecnología *Bluetooth* comprende hardware, software y requerimientos de interoperatividad, por lo que para su desarrollo ha sido necesaria la participación de los principales fabricantes de los sectores de las telecomunicaciones y la informática, tales como: Sony Ericsson, Nokia, Motorola, Toshiba, IBM e Intel, entre otros.

Hablar de Bluetooth es hacerlo de unificación. Traducido al castellano, Bluetooth significa "Diente Azul", el apodo de un rey vikingo que, en el siglo X, logró unir políticamente bajo su reinado a Dinamarca y Noruega, ambas separadas por el mar. Al igual que este rey, Bluetooth une y conecta distintos dispositivos entre sí de forma inalámbrica, solventando uno de los problemas que la vida digital provoca: la masiva proliferación de cables.

Bluetooth proporciona una vía de interconexión inalámbrica entre diversos aparatos que tengan dentro de sí esta tecnología, como móviles (Nokia 6600), consolas (Nokia N-Gage), dispositivos PDA, cámaras digitales, computadoras portátiles, impresoras, o simplemente cualquier dispositivo que un fabricante considere oportuno, usando siempre una conexión segura de radio de muy corto alcance. El alcance que logran tener estos dispositivos es de 10 metros para ahorrar energía ya que generalmente estos dispositivos utilizan mayoritariamente baterías. Sin embargo, se puede llegar a un alcance de hasta 100 metros (similar a Wi-Fi) pero aumentando el consumo energético considerablemente. Para mejorar la comunicación es recomendable que nada físico, como por ejemplo una pared, se interponga.

1.4.2 Infrarrojos

La radiación infrarroja o radiación térmica es un tipo de radiación electromagnética de mayor longitud de onda que la luz visible, pero menor que la de las microondas. Consecuentemente, tiene menor frecuencia que la luz visible y mayor que las microondas.

El nombre de infrarrojo, que significa por debajo del rojo, proviene de que fue observada por primera vez al dividir la luz solar en diferentes colores por medio de un prisma que separaba la luz en su espectro de manera que a ambos extremos aparecen visibles las componentes del rojo al violeta (en ambos extremos).

Aunque sus usos son variados, cabe destacar las redes por inflarrojos. Las redes por infrarrojos permiten la comunicación entre dos nodos, usando una serie de leds infrarrojos para ello. Se trata de emisores/receptores de las ondas infrarrojas entre ambos dispositivos, cada dispositivo necesita "ver" al otro para realizar la comunicación por ello es escasa su utilización a gran escala. Esa es su principal desventaja, a diferencia de otros medios de transmisión inalámbricos

Se utiliza principalmente para realizar intercambio de datos entre dispositivos móviles, como PDA's o móviles, ya que el rango de velocidad y el tamaño de los datos a enviar/recibir es pequeño.

1.4.3 Wi-fi

Wi-Fi (o Wi-fi, WiFi, Wifi, wifi, del inglés Wireless Fidelity) es un conjunto de estándares para redes inalámbricas basados en las especificaciones IEEE 802.11. Creado para ser utilizado en redes locales inalámbricas, es frecuente que en la actualidad también se utilice para acceder a Internet.

Wi-Fi es una marca de la *Wi-Fi Alliance* (anteriormente la *Wireless Ethernet Compatibility Alliance*), la organización comercial que prueba y certifica que los equipos cumplen los estándares IEEE 802.11x

Algunos argumentan que Wi-Fi y las tecnologías de consumo relacionadas son la clave para reemplazar a las redes de telefonía móvil como GSM. Algunos obstáculos para que esto ocurra en el futuro próximo son la pérdida del roaming (capacidad de un dispositivo para moverse de una zona de cobertura a otra), la autenticación más precaria y la estrechez del espectro disponible.

A pesar de dichos problemas, compañías como SocketIP y Symbol Technologies están ofreciendo plataformas telefónicas (reemplazos de centrales y terminales (teléfonos) que utilizan el transporte Wi-Fi.

Uno de los problemas más graves a los cuales se enfrenta actualmente la tecnología Wi-Fi es la seguridad. Un muy elevado porcentaje de redes son instaladas por administradores de sistemas y redes por su simplicidad de implementación sin tener en consideración la seguridad y, por tanto, convirtiendo sus redes en redes abiertas, sin proteger la información que por ellas circulan, existen sin embargo medios para garantizar la seguridad de estas redes. Gracias a la movilidad que ofrece esta tecnología estos dispositivos ofrecen gran comodidad.

1.5 Comparativa

ZigBee es muy similar al Bluetooth pero con algunas diferencias:

Una red ZigBee puede constar de un máximo de 64000 nodos, frente a los 8 máximos de una red Bluetooth.

Tiene un menor consumo eléctrico que el ya de por sí bajo del Bluetooth. En términos exactos, ZigBee tiene un consumo de 30ma transmitiendo y de 3ma en reposo, frente a los 40ma transmitiendo y 0.2ma en reposo que tiene el Bluetooth. Este menor consumo se debe a que el sistema ZigBee se queda la mayor parte del tiempo dormido, mientras que en una comunicación Bluetooth ésto no se puede dar, y siempre se está transmitiendo y/o recibiendo.

Tiene un ancho de banda de 250 kbps, mientras que el bluetooth tiene 1 Mbps.

Debido al ancho de banda de cada uno, uno es más apropiado que el otro para ciertas cosas. Por ejemplo, mientras que el Bluetooth se usa para aplicaciones como el wireless USB, los teléfonos móviles y la informática casera, el ancho de banda del Zigbee aunque menor puede permitir usos como la domótica los productos dependientes de la batería, los sensores médicos, control de periféricos, redes industriales y en artículos de juguetería, en los cuales la transferencia de datos es menor.

Cabe destacar también el hecho de que los módulos ZigBee sean los transmisores inalámbricos más baratos de la historia, y en un futuro podrán ser producidos de forma masiva. Tendrán un coste aproximado de alrededor de los 2 euros, y dispondrán de una antena integrada, control de frecuencia y una pequeña batería. Ofrecerán una solución mucho más económica que el bluetooth porque la radio que utilizan se puede fabricar con muchos menos circuitos analógicos de los que se necesitan habitualmente.

Pero sin duda una de las ventajas mas importantes que actualmente ofrece Zigbee es el sistema estándar de especificaciones que permite desarrollar aplicaciones usando el MICROCHIP Stack de una manera rápida e intuitiva una vez que el desarrollador se ha familiarizado con el entorno.

Respecto a los infrarrojos, como se dijo en el punto anterior su principal inconveniente es la necesidad de que los dos elementos que intentan comunicarse, mantenga línea de visión, a parte de las colisiones que el uso de aparatos diferentes de infrarrojos puede ocasionar entre si, lo que no ocurre si utilizamos Zigbee.

El principal problema de wifi era la seguridad, sin embargo la seguridad ya viene implícita en el Zigbee por lo que no nos dará dolor de cabeza desarrollar sabiendo que podemos usar AES 128 bits y el 802.15.4, que tipo de seguridad cojamos será una decisión nuestra y esto dependerá del uso de las comunicaciones a trata, pero escojamos el tipo de seguridad que cojamos la asignación de claves nos dejara el camino limpio para desarrollar seguros en Zigbee.

Exponemos a continuación una tabla resumiendo en la medida de lo posible lo explicado en los párrafos anterior, dejamos de lado los infrarrojos y nos centramos en las tres mas importantes:

Estándar	Ancho de Banda	Consumo de potencia	Ventajas	Aplicaciones
Wi-Fi	Hasta	400ma	Gran ancho de	Navegar por

	54Mbps	transmitiendo, 20ma en reposo	banda	Internet, redes de ordenadores, transferencia de ficheros
Bluetooth	1 Mbps	40ma transmitiendo, 0.2ma en reposo	Interoperatividad, sustituto del cable	Wireless USB, móviles, informática casera
ZigBee	250 kbps	30ma transmitiendo, 3ma en reposo	Batería de larga duración, bajo coste	Control remoto, productos dependientes de la batería, sensores , juguetería

2.ARQUITECTURA DE ZIGBEE

En la actualidad existe una gran cantidad de estándares para las comunicaciones inalámbricas. Permiten grandes tasas de transferencia para aplicaciones tales como la transmisión de audio, vídeo, datos, etc. Sin embargo, estos estándares no son adecuados para situaciones en las que el consumo energético o la complejidad del dispositivo son vitales. Para ello se ha diseñado ZigBee.

Tanto los sensores como los actuadores u otros dispositivos pequeños de medida o control no requieren un gran ancho de banda, pero si un mínimo consumo energético y una baja latencia. ZigBee es idóneo para la comunicación de estos dispositivos.

Se define ZigBee como una pila de protocolos que permite la comunicación de forma sencilla entre múltiples dispositivos. Especifica diversas capas, adecuándose al modelo OSI. Las capas básicas, física (PHY) y de control de acceso al medio (MAC) están definidas por el estándar IEEE 802.15.4, LR-WPAN (Low Rate – Wireless Personal Area Network). Este estándar fue diseñado pensando en la sencillez de la implementación y el bajo consumo, sin perder potencia ni posibilidades.

El estándar ZigBee amplía el estándar IEEE 802.15.4 aportando una capa de red (NWK) que gestiona las tareas de enrutado y de mantenimiento de los nodos de la red; y un entorno de aplicación que proporciona una subcapa de aplicación (APS) que

establece una interfaz para la capa de red, y los objetos de los dispositivos tanto de ZigBee como del diseñador (aunque parece difícil, es bastante sencillo, sigue leyendo).

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

2.1 Características significativas de ZigBee/IEEE 802.15.4.

Entre muchas otras peculiaridades, podemos entresacar:

IEEE 802.15.4:

- Diversas bandas de trabajo: 2.4 GHz (16 Canales), 915 Mhz (10 Canales), 868 MHz (1 Canal)
- Tasas de transferencia: 250 Kb/s, 40 Kb/s, 20 Kb/s
- Topologías: estrella y p2p (punto-a-punto)
- Direccionamiento MAC recortado (16 bits) y extendido (64 bits)
- Métodos de acceso al canal: CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance)
- Soporta redes slotted (QoS) y non-slotted
- Bajo consumo energético
- Gran densidad de nodos por red
- Radio medio de alcance: 50 m (hasta 500 m, dependiendo del entorno)

ZigBee:

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

2.2 Topologías de red y dispositivos.

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

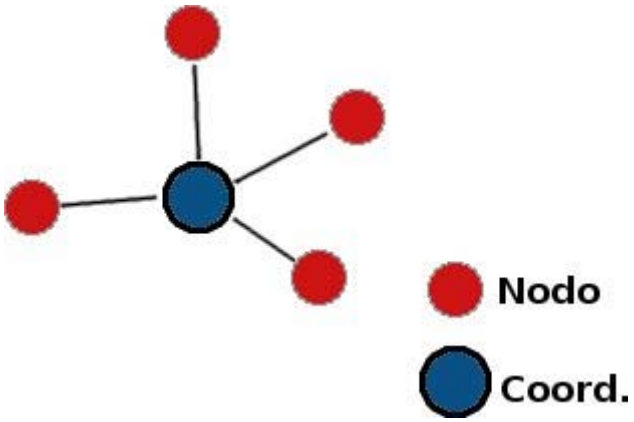
El RFD esta pensado para aplicaciones muy sencillas, como interruptores de iluminación y sensores infrarrojos, que no necesitan enviar o recibir grandes cantidades de información. Solo puede comunicarse con dispositivos FFD. Todo esto permite que pueda ser implementado usando los mínimos recursos posibles,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

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

Para poder tener una red, son necesarios como mínimo dos elementos. Un coordinador (FFD) que creará la red, le asignara el **NWKID** (NetWork IDentifier), y poseerá los mecanismos necesarios para la incorporación y eliminación de nodos en la

red. Además es necesario, como mínimo, un nodo, que puede ser FFD o RFD, con el que comunicarse.

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



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

2.2.1 Tipos de dispositivos IEEE 802.15.4

2.2.1.1 Dispositivo de función completa (FFD)

El dispositivo de función completa FFD, es capaz de recibir mensajes del estándar 802.15.4. Tiene todas las funcionalidades y puede ser utilizado tanto como coordinador de la red, como router o como dispositivo final con el que interactúe el usuario. Por el papel que suele desempeñar es típico que permanezca a la escucha cuando esta ocioso en vez de apagarse para ahorra recursos.

2.2.1.2 Dispositivo de función reducida (RFD)

El dispositivo de función reducida tiene capacidad y funcionalidad limitadas (especificada en el estándar) con el objetivo de conseguir un bajo coste y una gran simplicidad. Básicamente, son los sensores/actuadores de la red. Cuando no están realizando ninguna tarea se quedan en reposo a la espera de ser despertados por alguna tarea.

2.2.2 Tipos de dispositivos ZigBee

2.2.2.1 Coordinador

Uno por red, es el encargado de formar la red y establecer su dirección así como de mantener una tabla con los distintos dispositivos conectados a la red y de las ligaduras entre sus puntos finales. Tiene que ser implementado utilizando obligatoriamente un dispositivo de función completa.

2.2.2.2 Router

Su utilización es opcional, extiende el rango físico de la red permitiendo a mas nodos conectarse a ella y proporciona funciones de monitorización y control. Tiene que ser implementado utilizando obligatoriamente un dispositivo de función completa.

2.2.2.3 Dispositivo final

Puede ser desarrollado utilizando un dispositivo de funcionalidad completa o uno de funcionalidad reducida, suele actuar como sensores y actuadores de la red.

Un protocolo Zigbee de comunicación inalámbrica puede asumir varios tipos de configuraciones pero en todas ellas deben aparecer por lo menos un nodo con funcionalidad completa que haga de coordinador de la red y uno a varios dispositivos

finales. El router aparece tan sólo en algunas topologías de red que se explicarán a continuación.

2.2.3 Topologías

2.2.3.2 Estrella

Como se ha explicado anteriormente la topología tipo estrella consiste en un nodo coordinador y uno o más dispositivos finales, los cuales se comunican únicamente con el. Si un dispositivo final necesita transferir datos a otro dispositivo final, primero los manda al coordinador que se encargará posteriormente de reenviarlos al destinatario.

2.2.3.3 Árbol

Otro tipo de topología es la topología tipo árbol. En esta configuración los mensajes enviados por un dispositivo final serán procesados por el coordinador o por un router intermedio dependiendo del destinatario. Los routers desempeñan dos funciones. La primera es incrementar el número de nodos que se pueden asociar a la red. La segunda extender el rango físico de la red de manera que no todos los dispositivos tengan que estar en el rango del coordinador. Todos los mensajes enviados son rutados por las distintas ramas del árbol.

2.2.3.4 Red

La topología tipo red es parecida a la topología tipo árbol exceptuando el hecho de que los dispositivos FFD pueden rutar mensajes directamente a otros FFD en vez de seguir la estructura del árbol. Los mensajes RFD siguen teniendo que ir a través de su padre. Las ventajas de esta topología son que la latencia del mensaje se ve reducida y la fiabilidad incrementada. La topología tipo árbol y la tipo malla son también conocidas como redes multi-hop, debido a su habilidad para rutar paquetes a través de multiples

dispositivos, mientras que la tipo estrella se conoce como red single-hop. En todas estas configuraciones todos los nodos de la red tienen igual acceso al medio de comunicación.

Existen dos tipos de acceso, con balizas y sin balizas. En una red sin balizas todos los nodos de la red tienen permitido transmitir todo el tiempo mientras el canal este ocioso. En un red con balizas, los nodos solo tienen permitido transmitir en unos periodos de tiempo determinados. El coordinador periódicamente comenzará con una supertrama identificada como una trama de baliza, y todos los nodos de la red tratarán de sincronizarse con esta trama. A cada nodo se le asigna un espacio especifico dentro de la supertrama, durante el cual tiene permitido transmitir y recibir datos. La supertrama puede también contener espacios comunes donde todos los nodos compiten por el canal.

2.3 Requisitos hardware

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

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

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

2.4 Comunicaciones

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

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

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

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

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

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

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

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

La capa APS es la encargada de enviar los PDUs de una aplicación entre dos o mas dispositivos (llamada **APSDE**, APS Data Entity) y de descubrir y enlazar los dispositivos y mantener una base de datos de los objetos controlados, conocida como **AIB**, APS Information Base. Dos dispositivos se enlazan en la AIB en función de los servicios que ofrecen y de sus necesidades. Esto es útil para el direccionamiento indirecto. Así, es posible enviar un paquete a un dispositivo en función de la dirección

de origen, ya que están vinculados. Esta tabla solo puede estar presente en el dispositivo coordinador o en uno designado a tal efecto.

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

2.5 Encaminamiento (routing)

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

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

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

2.6 Perfiles

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

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

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

Cada perfil contiene las descripciones de los dispositivos que incluye, los identificadores de cada **cluster**(y en su caso, sus atributos) y los tipos de servicio ofrecidos. Las descripciones de los dispositivos están definidas por un valor de 16 bits (es decir, hay 65536 posibles descripciones). Los identificadores de cluster son de 8 bits (existen 256 posibles clusters). A su vez, si el tipo de servicio ofrecido es orientado a

pares clave-valor, cada cluster puede contener atributos por valor de 16 bits (o 65536 atributos por cluster). Existen muchas posibilidades dentro de un mismo perfil, y es labor del diseñador del perfil adecuar las necesidades para crear descriptores sencillos y permitir un proceso eficiente de mensajes.

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

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

2.7 Tecnologías actuales

Ya que la ZigBee Alliance solo define un estándar, la pila de ZigBee ha de ser implementada por terceros. Existen diferentes implementaciones, aunque muy pocas, ya que es una tecnología en constante desarrollo.

La empresa Microchip ofrece de forma gratuita el código de su implementación, en lenguaje C. Está diseñada explícitamente para que sea compatible con casi todos los Microcontroladores de Microchip de la familia 18F que tengan un interfaz **LPI** de comunicaciones. Su uso está explicado en el documento AN965. Por desgracia, a fecha de este escrito, no está completamente implementada, ya que carece de algunas funciones, como por ejemplo la seguridad. Además solo puede ser usada para investigación. Si se desea distribuir algún producto con la pila de Microchip, la empresa requiere que seas miembro de la ZigBee Alliance.

En cuanto a la implementación de IEEE 802.15.4, Chipcon, de Texas Instruments, proporciona un integrado que se encarga de las comunicaciones: el CC2420.

La implementación de Microchip necesita un MCU y un transmisor/receptor compatible (el CC2420 por ejemplo). Para proveer una solución completa, Chipcon ha creado el CC2430, que es un CI que integra un MCU, el IEEE 802.15.4 transceiver y la pila ZigBee.

Para evaluar las posibilidades de ZigBee y su pila, Microchip proporciona un kit de desarrollo conocido como PicDem Z.

2.8 Posibles aplicaciones

ZigBee es especialmente indicado para situaciones en las que el consumo energético y/o los costes de implementación son críticos. Por ello, es apropiado para aplicaciones de domótica en las que no se desea o no es posible crear un entramado de hilos de comunicaciones. También es propicio para aquellos casos en los que los elementos controlados o los sensores son móviles. Por ejemplo, si se desea crear un red sensorial para el control de ejemplares de cierta fauna en un espacio delimitado o para controlar las electro-bombas y los sensores de humedad de un invernadero, ZigBee es idóneo.

A pesar de las bondades de la pila, tenemos que ser cuidadosos a la hora de diseñar una posible aplicación y tener en cuenta ciertas consideraciones.

Sabemos que los dispositivos finales (**ED**, End Device) son los que permiten un reducido consumo energético, pero también sabemos que estos solo se pueden comunicar con su coordinador y que la latencia de las comunicaciones es inversamente proporcional al consumo de los dispositivos. Esto se debe a que cuando un ED no está transmitiendo o recibiendo, se desactiva y entra en un estado de bajo consumo o sleep.

Solo despertará de este estado cada cierto tiempo, preprogramado, o cuando una interrupción externa lo despierte. Por ello, todos los mensajes que deban ser enviados a ese ED han de dirigirse a su coordinador, que los mantendrá en una caché hasta que el ED pueda recibirlos. Esto aumenta la latencia y la carga del coordinador, pero permite al ED funcionar con baterías por su bajo consumo. Por supuesto, un dispositivo final RFD no puede desempeñar labores de enrutado, pero ya que es posible tener ED de tipo FFD, técnicamente si es posible tener dispositivos finales que enruten paquetes, aunque esto va en contra de la definición de "dispositivo final". De todos modos, si el dispositivo tiene que encaminar el tráfico de sus vecinos, este ha de ser FFD y no puede permanecer en modo sleep, ya que las rutas que contasen con el no serían útiles.

Por todo ello, si deseamos crear una red extensa con muchos dispositivos, debemos elegir bien cuáles podrán encaminar paquetes y cuales podrán ahorrar energía. Si además queremos hacer esto de forma transparente e independiente del dispositivo, es decir, que no queremos preocuparnos de donde estén los dispositivos y de que tipo sean, sino que la propia red, por su estructura física, lógica y espacial ha de determinar el rol de cada dispositivo, tendremos que proporcionar a cada nodo de la red de funcionalidad completa, de la posibilidad de alimentarse con fuentes "perdurables" de energía y de los algoritmos necesarios para determinar el rol de cada nodo. Algo bastante complejo.

3.ENTORNO DE DESARROLLO

3.1 PIC

3.1.1 PIC18F4620 y familia.

El PIC utilizado, está dentro de la familia PIC18 de microcontroladores, que ofrece un alto rendimiento computacional a un precio económico, con una ostentosa mejora con respecto a las anteriores, en cuanto a la memoria de programa.

La familia PIC18Fxx incluye una serie de características que hacen que se pueda reducir el consumo de energía durante las operaciones. Estas pueden ser:

-Modos RUN alternativos: Bloqueando el controlador desde el Timer1 o el oscilador interno, el consumo puede ser reducido hasta un 90%

-Modos libres múltiples: El núcleo de procesador puede estar parado mientras los periféricos están activos.

- Consumo bajo en módulos de Switch: el requisito de consumo para ambos Timer1 y el Timer del guardián está minimizado.

Otras características del diseño son las siguientes:

-Memoria resistente: Las celdas de las memorias de programa y de datos han sido mejoradas para aguantar entre 100.000 y 1.000.000 de escrituras y borrados, pudiendo llegar a durar hasta más de 40 años.

-Conjunto de instrucciones extendido: Se han creado 8 nuevas instrucciones para manejar direccionamientos indexados para poder posibilitar la utilización de lenguajes de programación como “C”.

-USART direccionable mejorado: este módulo de comunicación serie es capaz de operar a nivel RS - 232 y suministra el soporte para el protocolo LIN de bus

3.1.2 Arquitectura

Los dispositivos se diferencian entre sí en cinco puntos:

- Memoria FLASH de programa (48 Kbytes para dispositivos de PIC18FX525, 64 Kbytes para PIC18FX620).

- A / D channels (10 por 28 pin-devices, 13 por 40/44 pin-device).

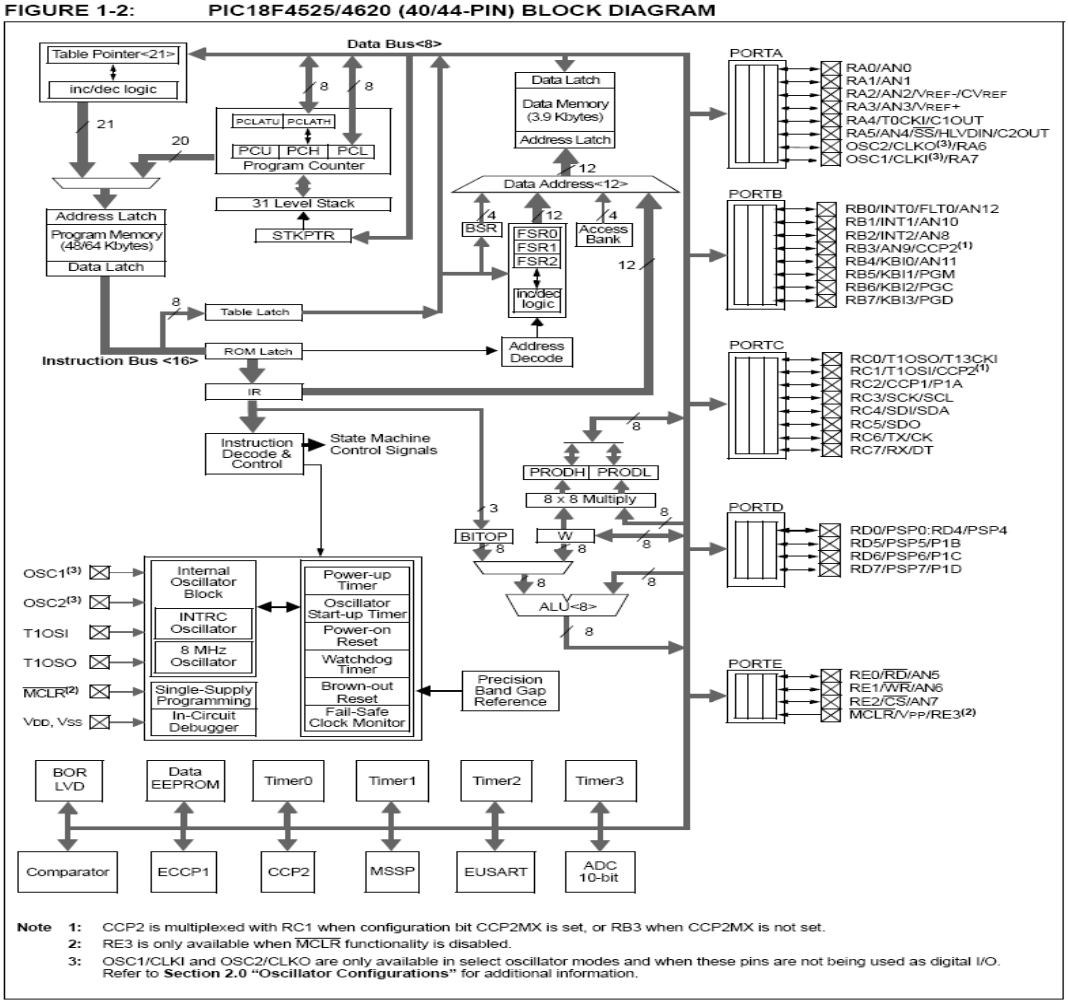
- Puertos de E/S (3 puertos bidireccionales sobre 28 pin-device, 5 puertos bidireccionales sobre 40/44 pin-device).

- CCP y la puesta en práctica de CCP mejorada

- El puerto paralelo esclavo (presente solamente sobre 40/44 pin-devices).

En la figura siguiente podemos ver el diagrama de la estructura interna de los dispositivos PIC18F4525 y PIC18F4620, siendo este último el que ha sido utilizado para la realización del proyecto:

FIGURE 1-2: PIC18F4525/4620 (40/44-PIN) BLOCK DIAGRAM

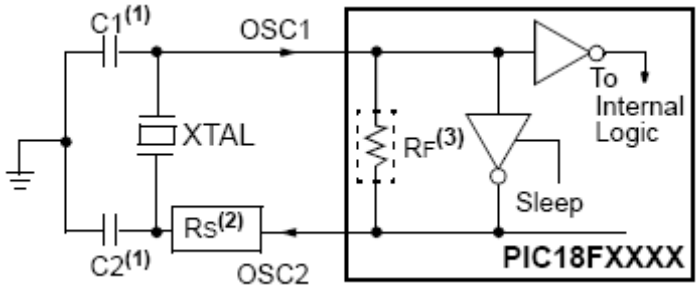


Note 1: CCP2 is multiplexed with RC1 when configuration bit CCP2MX is set, or RB3 when CCP2MX is not set.
2: RE3 is only available when MCLR functionality is disabled.
3: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

3.1.2.1 Osciladores

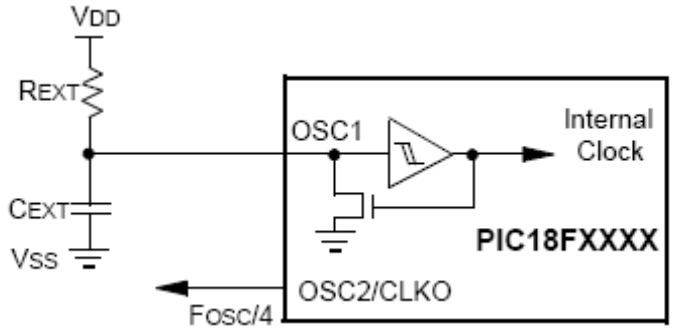
Los dispositivos de PIC18F2525 / 2620/4525/4620 pueden operar en diez modos de oscilador diferentes. El usuario puede programar las partes de configuración FOSC3:FOSC0 para seleccionar uno de estos diez modos.

-Osciladores de cristal/Resonadores cerámicos: cuatro modos conectados a los pin OSC1 y OSC2 para establecer la oscilación.



-Entrada de reloj externo: Requerido para algunos modos de oscilador.

-Oscilador RC: En el modo de oscilador de RC, la frecuencia de oscilador se divide por 4 y está disponible sobre el pin OSC2. Esta señal puede ser usada para los pruebas o para sincronizar otro dispositivo.



-Multiplicador de frecuencia PLL: El circuito Phase Locked Loop (PLL) es proveído como uno alternativa para usuarios que desean usar una frecuencia más baja.

-Osciladores internos: El dispositivo, incluye un oscilador interno que es capaz de generar dos tipos de señales de reloj, esto puede eliminar la necesidad de usar señales externas sobre OSC1 y OSC2.

El registro OSCCON controla varios aspectos de la operación del reloj del dispositivo:

R/W-0	R/W-1	R/W-0	R/W-0	R(1)	R-0	R/W-0	R/W-0
IDLEN	IRCF2	IRCF1	IRCF0	OSTS	IOFS	SCS1	SCS0
bit 7							bit 0

BIT 7 **IDLEN**: BIT de Enable Idle

1 = El dispositivo entra en el modo libre en la instrucción SLEEP.

0 = El dispositivo entra en el modo de Reposo en la instrucción SLEEP.

BIT 6 - 4 **IRCF2:IRCF0**: Selección de frecuencia de los osciladores internos

111 = 8 MHz

110 = 4 MHz

101 = 2 MHz

100 = 1 MHz (3)

011 = 500 Khz.

010 = 250 Khz.

001 = 125 Khz.

000 = 31 Khz.

BIT que 3 **OSTS**: estado del Time out

1 = Tiempo expirado

0 = Tiempo corriendo aún

BIT 2 **IOFS**: La frecuencia de INTOSC es estable o no

1 = la frecuencia de INTOSC es estable

0 = la frecuencia de INTOSC no es estable

BIT 1 - 0 **SCS1:SCS0**: BIT de selección de oscilador

1x = oscilador interno

01 = oscilador de (Timer1) secundario

00 = oscilador principal

3.1.2.2 Gestión de modos de energía

Los dispositivos PIC18F2525/2620/4525/4620 brindan un total de siete modos operativos para una gestión más eficiente de la energía. Hay tres categorías:

- Modos de Run
- Modos libres
- Modo de Reposo

La selección de uno de ellos, necesita dos decisiones: si la CPU va a hacer uso de reloj o no, y cual en caso afirmativo.

Modo Run: En los modos run, el reloj del núcleo y los dispositivos periféricos están activos. La diferencia entre éstos son el origen de reloj.

Modo Sleep: Cuando se entra a este modo desde otro, no se necesita selección de señal de reloj. Una vez despertado, el dispositivo no tendrá señal de reloj hasta que se seleccione.

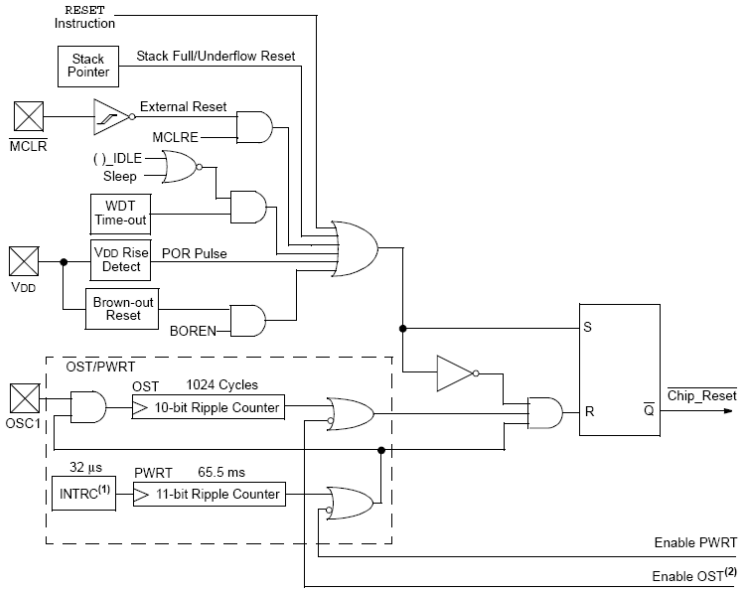
Modo Idle: los modos libres, permiten apagar selectivamente la CPU mientras lo periféricos continúan activos.

3.1.2.3 Reset

Los dispositivos diferencian entre varias clases de Reset:

- Power-on Reset (POR)
- MCLR Reset durante operación normal.
- MCLR Reset durante selección de modo de energía.
- Watchdog Timer (WDT) Reset
- Programmable Brown-out Reset (BOR)
- Instrucción RESET
- Stack Full Reset
- Stack Underflow Reset

En el siguiente diagrama se muestra la arquitectura del control del reset:



3.1.2.4 Organización de la memoria

Existen tres tipos de memorias en los dispositivos PIC:

-Memoria de programa
-RAM
-EEPROM

Como en una arquitectura Harvard, las memorias de programa y la de datos usan buses de datos separados. La EEPROM se puede considerar un periférico, accediendo a ella a través de los registros de control.

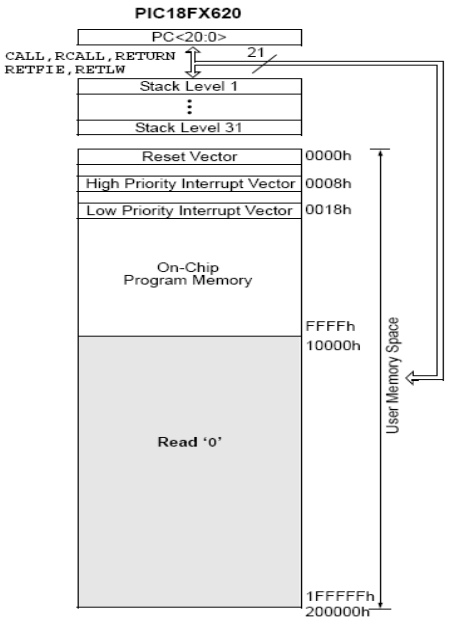
Organización de la memoria de programa:

La memoria tiene un contador de programa de 21-bits, capaz de direccional hasta 2 MBytes. El PIC18F4620 tiene una memoria Flash de 64 KBytes y puede almacenar hasta 32.768 instrucciones.

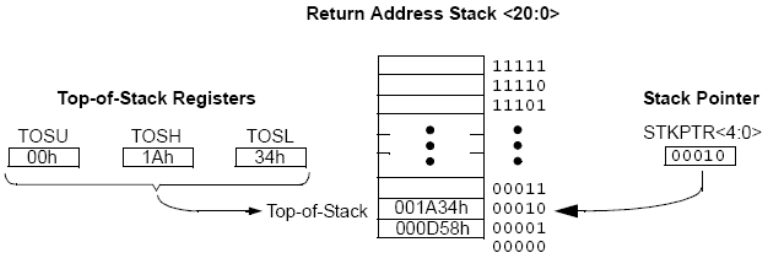
Los dispositivos tienen dos vectores de interrupción.

El contador de programa tiene 21 bits divididos en tres registros de 8 bits. Los bits <0:7> son tanto de escritura como lectura, mientras que los otros dos registros, bits <8:20> no son accesibles directamente. Este registro (el contador) direcciona las instrucciones sobre la memoria. Para evitar una mala alineación con la memoria de instrucciones, el bit 0, se inicializa siempre a '0' y se aumente de dos en dos porque las instrucciones son de 16 bits.

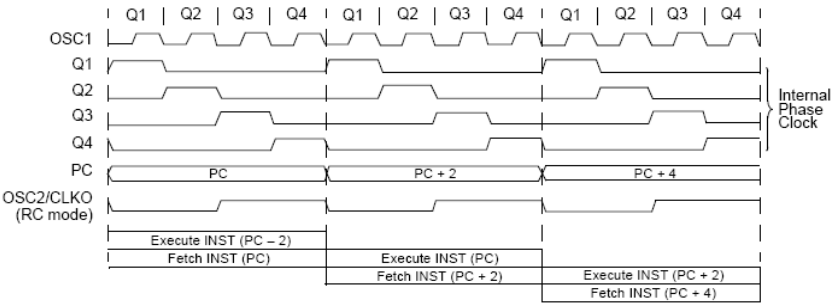
A continuación se muestra el mapa de la memoria:



La pila tiene 31 palabras de 21 bits y no está situada en el espacio del programa o en el de datos, tiene su propia dirección y puede ser escrita y leída. Es posible hacer PUSH y POP usando estos registros. Esta, también hay que decir que permite hasta 31 anidamientos de llamadas a programas e interrupciones.

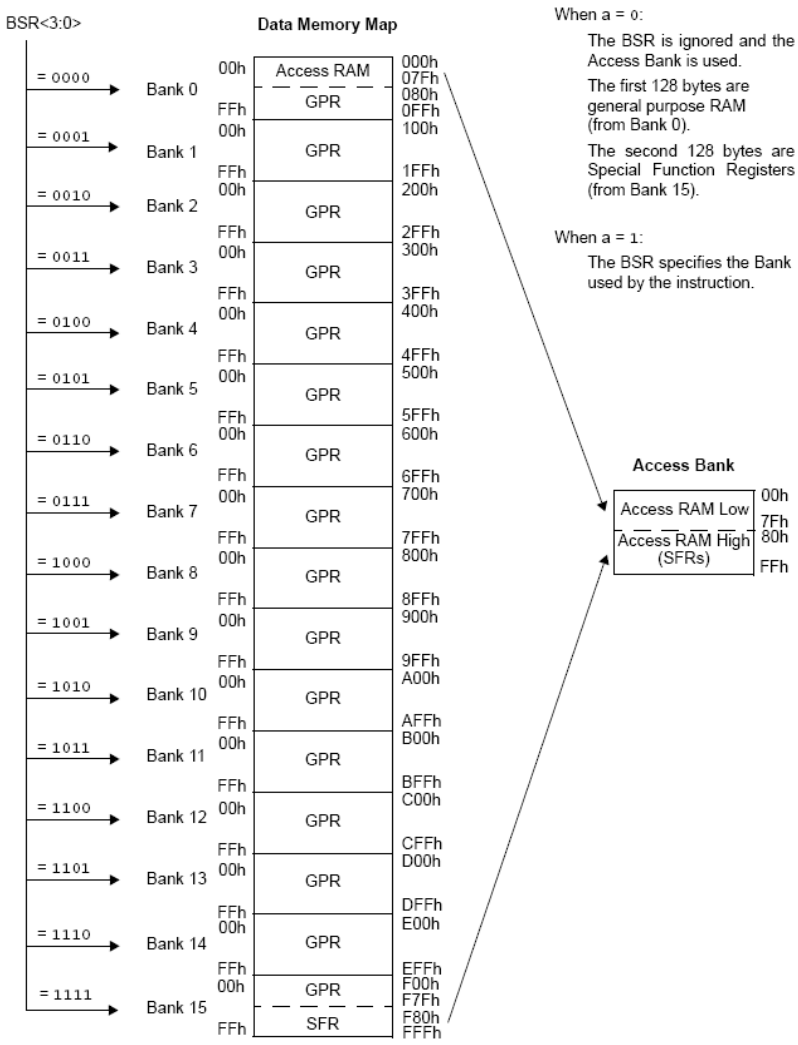


En cuanto a los ciclos de instrucción, la entrada de reloj, ya bien sea de una fuente interna o externa, es dividida internamente en cuatro (Q1, Q2, Q3, Q4). La instrucción se busca y ejecuta entre Q1 y Q4.



Organización de la memoria de datos:

La memoria de datos está implementada como una RAM estática de 12 bits de dirección admitiendo hasta 4096 bytes. Esta, está dividida en 16 bancos de memoria de 256 registros. Permite el acceso a toda la memoria mediante accesos directos, indirectos o indexados.



3.1.2.5 Memoria FLASH de programa

Esta memoria permite borrado, lectura y escritura. La lectura se hace sobre un solo byte cada vez mientras que la escritura se hace sobre bloques de 64 KBytes. Mientras la memoria se está escribiendo o borrando, no puede ser accedida, o cual hace que mientras se están haciendo estas tareas, no se pueda ejecutar el programa.

Los registros de control son los siguientes:

-EECON1: Control de acceso a memoria.

- ECON2: Se usa exclusivamente en secuencias de escritura.

- TABLAT: Mantiene 8-bits de comunicación entre la memoria y la RAM.

- TBLPTR: Usado en instrucciones concretas.

3.1.2.6 Memoria de datos EEPROM

Esta memoria está físicamente separada de la memoria de programa y de la RAM, y está preparada para almacenar los datos durante un largo periodo de tiempo. Solo se permite leer y escribir en ella. Para estas acciones se usan los registros siguientes, ya que no se puede acceder a ellos directamente:

-EECON1: Registro de control de acceso a la memoria
- EECON2: Es una duplicación del EECON1 usado solamente para secuencias.
- EEDATA:
- EEADR: Direccional la memoria en escrituras y lecturas.
- EEADRH: Parte alta del registro EEADR.

3.1.2.7 Multiplicador hardware 8x8

En los PIC18 se incluye un multiplicador 8x8 en la ALU sin signo, que devuelve un resultado de 16 bits que se guardan en el par de registros PRODH:PRODL. Esta operación se realiza en un ciclo y permite reducir el tamaño del código y la dificultad de los algoritmos a realizar.

3.1.2.8 Interrupciones

Existen diferentes fuentes de interrupciones y dos vectores de prioridad para estas que permiten asignarles el nivel de prioridad a estas. Los registros utilizados para el control de interrupciones son los siguientes:

- RCON : Bits que determinan las causas del ultimo reset y vuelta desde el modo SLEEP y modo IDLE.
- INTCON : Contiene varios BIT de prioridad, habilitadores y flag.
- INTCON2 : Contiene varios BIT de prioridad, habilitadores y flag.
- INTCON3 : Contiene varios BIT de prioridad, habilitadores y flag.
- PIR1, PIR2 : Contiene varios bits de flag para interrupciones en periféricos.
- PIE1, PIE2 : Bits que habilitan las interrupciones externas.
- IPR1, IPR2: Bits de prioridad para las interrupciones externas.

En general cada fuente de interrupción tienen tres bits de control:

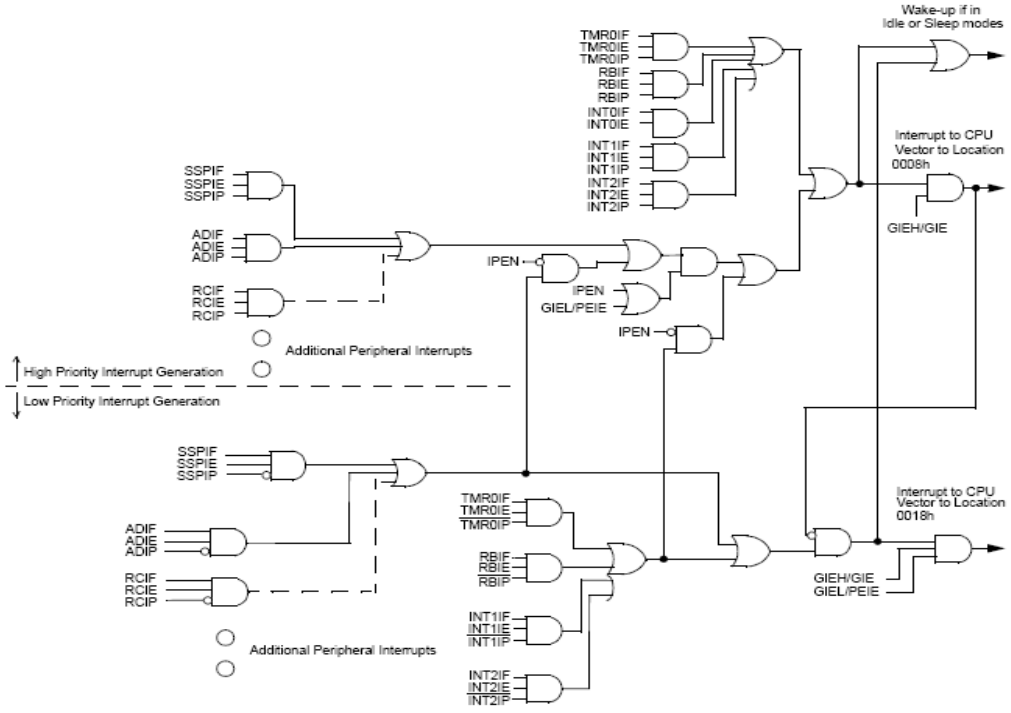
-Un BIT flag para detectar que un evento ha ocurrido.
-Un BIT para habilitar el tipo de interrupción.
-BIT de prioridad, para decidir si tiene alta o baja prioridad.

Las interrupciones pueden venir de las siguientes fuentes:

-INT pins: Son interrupciones externas y pueden despertar al dispositivo de los estados SLEEP e IDLE, estas son INT0, INT1 e INT2. La prioridad se puede establecer para las interrupciones INT1 e INT2.
- Interrupciones por temporizador mediante el registro TMR0.
-Interrupción por cambio en los valores del puerto B.

Cada vez que se produce una interrupción, el valor del contador de programa y el contexto se salvan en la pila, de modo que al terminar de tratar la interrupción, vuelve al contesto en el que se encontraba.

En el diagrama siguiente se puede ver el esquema del sistema de interrupciones:



3.1.2.9 Puertos de entrada/salida

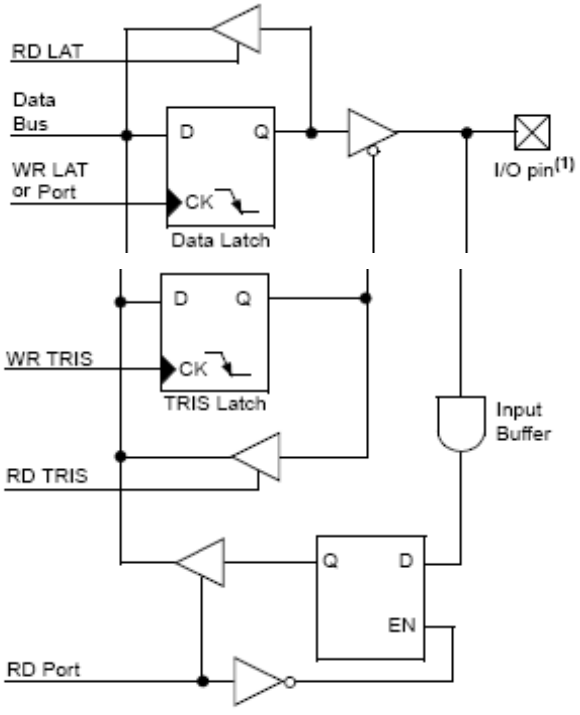
Dependiendo de la estratagema escogida y las características permitidas, hay hasta cinco puertos disponibles. Cuando un dispositivo periférico está activado, ese pin no puede en general ser usado como un pin de E/S para todo uso.

Cada puerto tiene tres registros para su operación. Éstos registros son:

- El registro de TRIS (registro de dirección de datos)
- El registro de puerto (lee los niveles sobre los pins del dispositivo)
- Registro LAT

El registro de LAT de datos es útil para la lectura de operaciones sobre el valor de los pins de E/S.

Modelo simplificado de un puerto de E/S genérico:



Note 1: I/O pins have diode protection to VDD and VSS.

- PUERTO A: El pin RA4 está multiplexado con el módulo de Timer0 de entrada de reloj. RA6 y RA7 está multiplexado con los pins de los osciladores principales.

- PUERTO B: Cuatro de los pins del puerto B, poseen una interrupción en el cambio de sus valores, solo los pins configurados como entradas pueden hacer que se produzca esta interrupción. Para saber si ha cambiado el valor en el puerto, se comparan con los valores antiguos que están almacenados en el registro LATB.

- PUERTO C: Utilizado para diferentes funciones de entrada/salida.

- PUERTO D: Utilizado en funciones con el puerto paralelo.

- PUERTO E: Control del puerto paralelo.

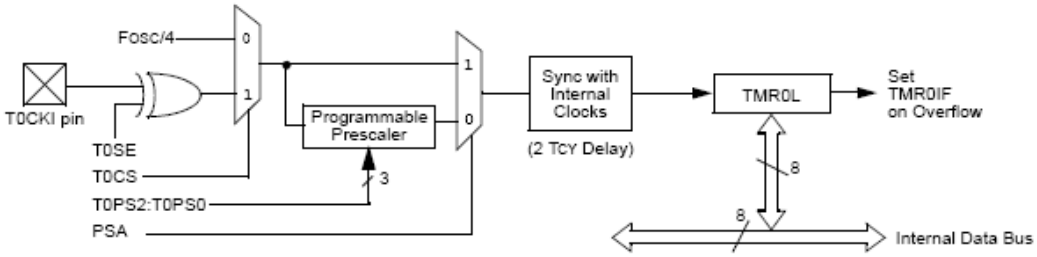
-PUERTO PARALELO: Este puerto se habilita con los bits superiores del registro TRISE, y puede ser leído y escrito asíncronamente desde el mundo exterior. Posee una interfaz con 8 bits de bus, que son leídos desde el registro PORTD.

3.1.2.10 Registro TIMER0

El módulo de Timer0 incluye las siguientes características:

- Operaciones con software seleccionable como un reloj automático.
- Registros legibles y gravables
- 8 BIT dedicados al software programable Prescaler.
- Fuente de reloj seleccionable (interno o externo)
- Borde selector para el reloj externo
- Interrupción en desbordamiento.

El registro de T0CON (registro 11 - 1) controla todos los aspectos de la operación del módulo. Es tanto legible como gravable. Timer0 puede actuar tanto como un contador o como un temporizador. Una fuente de reloj externa puede ser controlada por Timer0; sin embargo, debe cubrir ciertos requisitos para asegurar que el reloj externo puede ser sincronizado con el Reloj de fase interno.



Note: Upon Reset, Timer0 is enabled in 8-bit mode with clock input from T0CKI max. prescale.

La interrupción de TMR0 es generada cuando el TMR0 registra overflow de FFh a 00h.

Además del TIMER0, existen otros tres temporizadores que se usan similármente a este, son : TIMER1, TIMER2, TIMER3.

Hasta aquí se han explicado los componentes hardware más importantes y su distribución dentro del microcontrolador. Se han explicado diversos aspectos, muy importantes como pueden ser la distribución de la memoria o el uso de los principales puertos.

Una parte importantísima de cualquier procesador es el sistema de interrupciones, que también ha sido introducido y que en nuestro trabajo ha sido parte principal.

3.2 CC2420 antena del conjunto PICDEMZ

El CC2420, es un dispositivo periférico, que es utilizado para la comunicación entre distintos PICs. CC2420 puede ser usado juntos con varios tipos de antenas.

La banda 2.4 GHz es la utilizada por este dispositivo, es compartida por muchos sistemas tanto industriales, en oficinas y casas. Utiliza el estándar IEEE 802.15.4. Para la transmisión de datos, este estándar usa un sistema por el cual cada byte es transmitido con dos símbolos, cada uno representa 4 bits.

3.3 MPLAB e ICD2

Para el desarrollo de este proyecto, a parte del PIC y sus periféricos, hemos usado otras herramientas para la generación del código que posteriormente se volcará sobre las placas.

El entorno de desarrollo empleado ha sido el que suministra el fabricante del PIC, que se compone de diferentes utilidades, que a continuación explicaremos.

3.3.1 Compilador C MPLAB C18

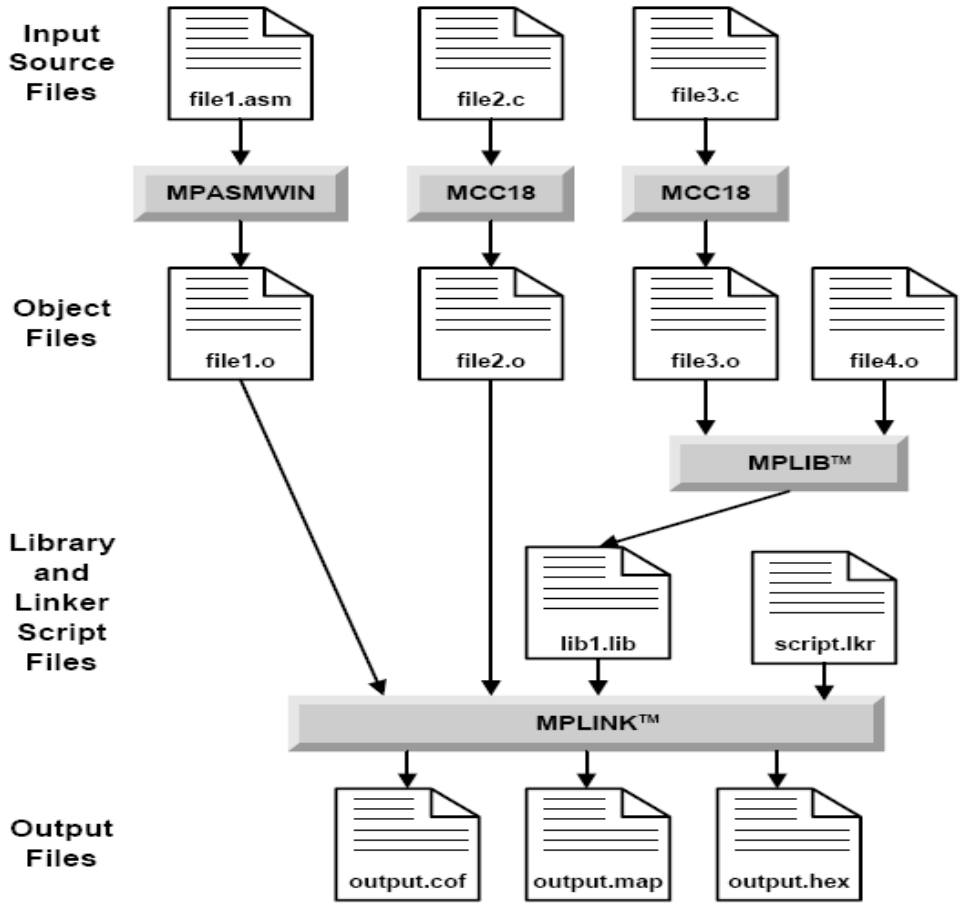
El compilador traduce un código entendible por los humanos, a un código de ceros y unos entendibles por el dispositivo PIC. El código humano se escribe en ANSI C en bloques diferentes, que posteriormente son unidos por el linker con los demás bloques y datos.

3.3.2 MPASM y MPLinker

El módulo MPASM, se ha incluido para poder definir pequeñas partes de código en lenguaje ensamblador, que necesitan ejecutarse rápidamente o bien tienen que ser ejecutadas en un periodo de tiempo estricto.

El MPLinker es el módulo que se ocupa de unir todas las secciones de código para que el resultado final en código máquina sea el deseado.

A continuación mostramos el flujo de ejecución del compilador:



3.3.3 ICD2 (In Chip Debugger).

Este dispositivo, viene acompañando a los dispositivos PIC en el paquete con el que se ha trabajado. Tiene varias dos funciones principales:

- Programación : Permite comunicar el entorno de desarrollo de MPLAB y su código compilado, con un dispositivo PIC, programando en este último el código anteriormente nombrado.

- Depuración: Un vez programado el dispositivo, es posible depurar viendo en el MPLAB, el estado del PIC en depuración, pudiendo poner “break-points”, hacer depuración “paso a paso”,…

Se puede conectar al ordenador de dos formas diferentes, la primera mediante un conector USB a cualquiera de los puertos que se tengan o bien mediante un cable para comunicación por el puerto COM.

Una vez conectado y descargado el firmware desde el MPLAB, se debe conectar el PIC al ICD2 mediante un cable con conectores RJ-11, con seis patillas, para así establecer la conexión que nos permitirá realizar funciones sobre el PIC.

3.3.4 Entorno de desarrollo JAVA.

Además de los anteriores componentes imprescindibles descritos, se ha trabajado en una pequeña aplicación para poder hacer una demostración de las posibilidades de este paquete de desarrollo.

El entorno elegido para crear una pequeña aplicación con gran jugabilidad ha sido Eclipse, que es gratuito y se encuentra disponible en Internet. La aplicación ha sido desarrollada en JAVA.

3.3.5 Comunicación serie.

Una de las partes importantes del proyecto, sin duda ha sido la comunicación serie entre los PIC y el ordenador en tiempo de ejecución y no de compilación, depuración o programación. Para ello hemos utilizado el protocolo RS-232, que consigue la comunicación a través del puerto COM con un cable.

4. DESARROLLO DEL PROTOTIPO

4.1 Desarrollo SW en el PC

4.1.1 RS-232

La comunicación entre el picdemz y el programa en java la realizamos a través del puerto serie en concreto utilizando el RS-232. El RS-232 puede transmitir los datos en grupos de 5, 6, 7 u 8 bits, a unas velocidades determinadas (normalmente, 19200 bits por segundo o más). Después de la transmisión de los datos, le sigue un bit opcional de paridad (indica si el numero de bits transmitidos es par o impar, para detectar fallos), y después 1 o 2 bits de Stop. Normalmente, el protocolo utilizado ser 8N1 (que significa, 8 bits de datos, sin paridad y con 1 bit de Stop). Tanto el aparato a conectar (en nuestro caso el coordinador) como el ordenador tienen que usar el mismo protocolo serie para comunicarse entre si.

El puerto serie es una de las conexiones mas comunes que se realizan en un ordenador permitiendo que éste se conecte con módems, impresoras, escaners, lectores de código de barras, etc. El API de Comunicaciones Java, constituido por el paquete **javax.comm**, que proporciona JavaSoft, no forma parte del JDK, pero añade soporte a Java para dispositivos serie y paralelo de manera que sirve perfectamente a nuestro propósito.

El paquete proporciona soporte para dispositivos serie y paralelo utilizando una semántica semejante a la que se usa con streams y eventos. Para comunicarse con un dispositivo serie a través de uno de los puertos serie de un ordenador, desde una aplicación Java o un applet, es necesario un interfaz. El API de Comunicaciones Java, permite transmitir y recibir datos a través de dispositivos conectados al puerto serie; proporcionando además un conjunto de opciones que permiten la configuración de todos los parámetros asociados a los puertos serie y paralelo. Este API es una

proposición para establecer un método estándar de acceso a los puertos de comunicaciones.

El API de Comunicaciones Java solamente puede controlar puertos de los cuales tenga conocimiento. En la última versión que JavaSoft ha proporcionado de este API, no es necesario que se inicialicen los puertos, ya que en el arranque, el API realiza una búsqueda de los puertos disponibles en la máquina en que se ejecuta y los va incorporando automáticamente.

En caso de que la nomenclatura de los dispositivos no siga la convención habitual se pueden añadir los puertos series explícitamente, éste no era nuestro caso así que nos hemos limitado a inicializar el puerto serie que íbamos a utilizar, aun así ponemos a continuación un código en el que se puede ver como añadir los puertos.

```
// Registro del dispositivo
CommPort ttya = new javax.comm.solaris.SolarisSerial( "ttya","/dev/ttya" );
CommPortIdentifier.addPort( ttya,CommPortIdentifier.PORT_SERIAL );
CommPort ttyb = new javax.comm.solaris.SolarisSerial( "ttyb","/dev/ttyb" );
CommPortIdentifier.addPort( ttyb,CommPortIdentifier.PORT_SERIAL );
```

Para añadir un dispositivo fijar sus características y abrirlo hemos utilizado un código parecido al que se expone a continuación. El ejemplo fija un dispositivo serie determinado para que sea accesible con el nombre **ControlPuertoSerie**. El dispositivo conectado a esta línea tiene una velocidad de 9600 baudios, 1 bit de parada, 8 bits por carácter y no dispone de paridad, y lo que se pretende es proporcionar dos canales, o streams, uno para leer y otro para escribir en el dispositivo conectado a este puerto.

```
InputStream entrada = null;
OutputStream salida;
SerialPort puertoSerie = null;

public ControlPuertoSerie( String dispositivo,int baudios,int timeout ) throws Exception
{
```

```
CommPortIdentifier idPuerto =
  CommPortIdentifier.getPortIdentifier( dispositivo );
puertoSerie = (SerialPort)idPuerto.openPort( "PuertoSerie",timeout );
puertoSerie.setSerialPortParams( 9600,SerialPort.DATABITS_8,
  SerialPort.STOPBITS_1,SerialPort.PARITY_NONE );
puertoSerie.setFlowcontrolMode( SerialPort.FLOWCTRL_NONE );
puertoSerie.enableRcvThreshold( 1 );
puertoSerie.enableRcvTimeout( timeout );
System.out.println( "Dispositivo Serie abierto" );

salida = puertoSerie.getOutputStream();
entrada = puertoSerie.getInputStream();
```

En el caso del API de Comunicaciones Java, la lectura y escritura no se diferencia en nada de cualquier llamada a métodos semejantes para realizar estas mismas tareas en objetos derivados del uso de streams.

Para escribir, se puede hacer:

```
try {
  salida.write( arraySalida,0,longitud );
```

Y para la lectura de datos, es suficiente con hacer:

```
try {
  int b = entrada.read()
```

El cierre de los puertos es un paso sumamente importante porque el API de Comunicaciones Java siempre intenta proporcionar acceso exclusivo a los dispositivos, y si algún canal no se cierra, no estará disponible para otras aplicaciones. Si se quiere utilizar un dispositivo para múltiples usuarios sobre una misma línea serie, es necesario emplear un protocolo que permita multiplexar la información proveniente de cada uno de ellos.

```
try {
  entrada.close();
  salida.close();
  } ...
```

Habría que comprobar el comportamiento del API de Comunicaciones Java ante grandes avalanchas de datos o aplicaciones en tiempo real, en donde sí se puede valorar la bondad de API. En nuestro caso lo hemos probado con cuatro dispositivos funcionando simultáneamente lo que supone una tasa de envío bastante reducida. Por ello no hemos podido probarlo en el sentido, de someterlo a un gran estrés. No obstante esa duda, la aproximación tan disciplinada que proporciona este API de Comunicaciones hará mucho más sencilla la integración de cualquier dispositivo serie como lectores de códigos de barras, impresoras, lectores de tarjetas, y cientos de otros dispositivos.

Los datos que enviamos por los puerto serie son el reflejo de los datos recibidos por el coordinador dependiendo de si en los dispositivos finales se pulsa uno u otro botón. Estos datos son procesados por el programa java, es decir el BuZz que actúa en consecuencia a los estímulos recibidos.

4.1.2 Buzz

Se trata de un juego multijugador en el que el número de jugadores viene determinado por las capacidades físicas del zigbee, esto quiere decir que teniendo en cuenta la posibilidad de utilizar routers el número de jugadores sería prácticamente ilimitado. El objetivo del juego es responder el máximo numero de preguntas de manera acertada. Se van mostrando preguntas por pantalla y los jugadores tratan de responder a la pregunta correctamente, se suma un punto al primer jugador en contestar en el caso de que la respuesta a la pregunta sea la acertada, en caso contrario se pasa a la pregunta siguiente. El juego acaba cuando no tenemos más preguntas. Las preguntas se leen de un fichero de texto y constan de cuatro posibles respuestas, solo una de ellas verdadera, los jugadores seleccionaran la respuesta que crean oportuna utilizando los botones

disponibles en los dispositivos finales. Los dispositivos finales podrán ser configurados antes o después de iniciar la aplicación, pero es necesario conocer el número de jugadores que deseamos que participen antes de comenzar a jugar. Cada vez que un usuario responda, se mostrara un mensaje por pantalla indicando la respuesta seleccionada y si esta es correcta o no, además del número del jugador que contesto. El jugador con más puntos cuando acaben las preguntas es el ganador.

4.2 Desarrollo SW para el PICDEM Z

En esta sección explicaremos la funcionalidad con la que hemos dotado a las placas de desarrollo PicDem-Z, así como las diversas soluciones encontradas a la hora de implementar dicha funcionalidad.

El desarrollo de este software ha sido realizado en dos versiones que se corresponden con dos versiones distintas de la implementación de la pila de red proporcionada por Microchip, sobre las cuales hemos implementado nosotros dos códigos distintos que implementan la capa de aplicación de cada una de estas pilas.
En los dos apartados siguientes explicaremos con detalle cada una de las características de estas dos versiones así como las soluciones aportadas por nosotros.

El motivo fundamental de utilizar dos versiones distintas de la pila de red de Zigbee es que Zigbee es una tecnología nueva y en constante evolución, de tal manera que cuando comenzamos el desarrollo de este proyecto la única versión que distribuía Microchip de su pila de red para Zigbee era la versión 1.0.00 que es sobre la que hemos realizado nuestra primera versión de la aplicación.

Más adelante y con fecha de marzo de 2007, microchip lanza la primera versión de su pila (versión 1.0.3.8) que es reconocida por la Zigbee Alianze como implementación que respeta el estándar del protocolo que esta corporación establece, y es por ello que debido a las motivaciones que provocaron el desarrollo de este proyecto, creímos conveniente ampliar su alcance y desarrollar una nueva aplicación para que

utilizara esta versión y realizar de este modo, un verdadera uso del protocolo de comunicaciones Zigbee.

4.2.1 SW para PICDEM-Z sobre la versión 1.0.00 de la pila de red Zigbee

En principio y antes de comentar nuestra aplicación en esta primera versión, vamos a explicar las características y limitaciones de la versión 1.0 de la pila de Microchip.

Como puntos mas importantes esta versión no esta reconocida por la Zigbee Alianze como una implementación de la pila de Zigbee ya que no dispone de muchas de las características que en principio son exigibles a este protocolo.

Las características principales de esta versión son:

- Esta basado en la versión 0.8 de la especificación de Zigbee
- Soporta una banda de frecuencia de 2.4GHz usando un emisor CC2420
- Soporta dispositivos de funcionalidad reducida y coordinadores (de funcionalidad completa)
- Implementa un almacenamiento no volátil de las tablas de vecinos y de ligaduras en el coordinador
- Soporta redes de topología estrella pero no soporta otros tipos de topologías como malla o árbol.
- Es portable con cualquier procesador de la familia PIC18
- Posee una arquitectura de cooperación a nivel multitarea
- Independiente de la aplicación
- De diseño modulable para incorporar y extraer diferentes funcionalidades.

Al estar basado en una versión no definitiva de la especificación Zigbee carece de diversas características que la especificación definitiva de la Alianza Zigbee propone, como son:

- la ausencia de seguridad en el acceso a la red
- no están disponibles ni los clusters ni las redes peer-to-peer (solo redes cliente-servidor)

- no tiene capacidad de enrutado de mensajes
- no soporta ligaduras de uno a varios.
- Como ya hemos dicho antes no soporta topologías tipo malla o árbol.

Organización del código

La pila está formada por múltiples archivos de código. Muchos de ellos son comunes a todas las aplicaciones Zigbee, mientras que otros son específicos de cada aplicación. Para facilitar el desarrollo y la programación los distintos archivos están organizados en subdirectorios debajo del directorio Source. Estos subdirectorios son: Stack, que contiene todo el código de la pila, DemoCoordApp, que contiene todo el código del coordinador y por último DemoRFDApp, que contiene todo el código de los dispositivos finales. Aunque los distintos archivos soportan cualquiera de las funcionalidades de los módulos de Zigbee, estas deben ser definidas en el archivo Zigbbe.def, es decir, se pueden desarrollar múltiples aplicaciones para nodos Zigbee usando el mismo código pero archivos .def diferentes, que deberán ser incluidos en el path de la compilación según convenga. Dentro del archivo .def encontramos definiciones como las que siguen a continuación:

```
-CLOCK_FREQ: Define la frecuencia del reloj.
-TICKS_PER_SECOND: número de ticks en un segundo.
-USE_CC24240: se usa para indicar que estamos usando ese tranceiver.
-I_AM_COORDINATOR: índica que el nodo es un coordinador
-I_AM_END_DEVICE: índica que el nodo es un dispositivo final.
-MAX_NEIGHBORS: índica el número máximo de nodos que soporta el
coordinador.
```

Dentro de los distintos archivos, vamos a destacar los que hacen referencia a la parte de la aplicación y los que son necesarios modificar según la funcionalidad que le queramos dar al proyecto.

Coordinador: El método principal del coordinador se encuentra en el archivo DemoZCoordApp.c, en él inicializamos la maquina de estados, los flags y todo el hardware de la placa. A continuación comprobamos que no se haya producido ningún error, si éste no se ha producido seguimos con la ejecución y en caso contrario dejamos al dispositivo durmiendo hasta que se solucione. Una vez hecho esto llamamos al método APLInit que es la principal llamada para la inicialización de la pila, la inicialización de la capa MAC se hace de manera independiente llamando al método APLEnable. A continuación creamos la red llamando al método APLNetWorkInit y entramos en el bucle principal que está formado por cuatro estados: el estado SM_APP_INIT en el que se empieza por defecto y que lo único que hace es mostrar un mensaje diciendo que vamos a comenzar a establecer la red y pasa al estado SM_APP_INIT_RUN. En el estado SM_APP_INIT_RUN tratamos de encontrar un canal vacío para establecer la red. Si lo encontramos establecemos la red y pasamos al estado SM_APP_NORMAL_START y en caso contrario volvemos a SM_APP_INIT. En el estado SM_APP_NORMAL_START permitimos la asociación de nuevos dispositivos finales y ligamos sus endpoints para que puedan ser posteriormente utilizados, una vez hecho esto pasamos al estado SM_APP_NORMAL_START donde no permitimos nuevas asociaciones y comprobamos si alguno de los endpoints ligados en el estado anterior ha sido pulsado en cuyo caso llamamos al método S2Task, S3Task.....dependiendo del botón que haya sido pulsado, y donde comprobamos también si recibimos algún mensaje de los dispositivos finales asociados llamando al método D1Task. Estos métodos son iguales para coordinador y dispositivos finales, en nuestro caso la funcionalidad que nos interesa para el coordinador es la de leer los mensajes recibidos por los dispositivos finales. Estos mensajes son mandados de manera indirecta a todos los dispositivos utilizando el método APLSendKVPIndirect donde a parte del mensaje, el identificador del mensaje y otros datos, se manda también el identificador del endpoint que envió el mensaje lo que nos va a servir para saber como procesarlo en el coordinador, donde vamos leyendo byte a byte utilizando los métodos APLGet y realizamos las acciones correspondientes, en nuestro caso mostrar el identificador del que envió el mensaje y el botón que había pulsado.

Si cargamos el programa en la placa y lo ejecutamos, la aplicación entrará automáticamente en un estado de configuración siempre que carguemos un programa nuevo o siempre que pulsemos el botón S3, el led RD0 se encenderá para indicar que

estamos en este estado. Conectamos la placa al ordenador mediante un cable RS232 y abrimos el hyperterminal, tenemos que tener cuidado de que el hyperterminal este correctamente configurado. Veremos a continuación un menú similar a este:

```
******************************************************************************
1. Set node ID...
2. Join a network.
3. Perform quick demo binding (Must perform #2 first)
4. Leave a previously joined network (Must perform #2 first)
5. Change to next channel.
6. Transmit unmodulated signal.
7. Transmit random modulated signal.
0. Save changes and exit.
Enter a menu choice:
```

Lo primero que tenemos que hacer es seleccionar la opción uno para fijar el id del coordinador que es el que viene indicado en la placa y que determina la dirección MAC de la placa. El nodo id forma los 16 bits menos significativos de la dirección MAC formada a su vez por 64 bits. A continuación pulsamos el dos para permitir nuevas asociaciones al coordinador. En este punto podemos encender uno de los RFD para que acceda a la red recién creada, si accede correctamente veremos un mensaje indicándolo, una vez creada la red pulsamos 0 para dejar el coordinador configurado.

Dispositivos finales: El método principal del RFD se encuentra en el archivo DemoZRFApp.c, las tareas iniciales son similares a las que realiza el coordinador y las principales diferencias se encuentran en los estados del bucle principal: el primer estado y en el que empezamos por defecto es SM_APP_INIT donde inicializamos todas las tareas de los endpoint y comprobamos si se ha pulsado S3, de así ser pasamos al estado SM_APP_CONFIG_START para entrar en el modo de configuración, sino se ha pulsado pasamos al estado SM_APP_NORMAL_START. Cuando estamos en normal start lo primero que va hacer el nodo es tratar de unirse a una red a la que se hubiera asociado con anterioridad para ello llamamos al método APLRejoin y pasamos a un estado de espera SM_APP_NORMAL_START_WAIT en el que esperamos a que el proceso de reconexión se complete y comprobamos si se ha completado satisfactoriamente o no. En el primer caso pasaríamos al estado SM_APP_NORMAL_RUN y en el segundo volveríamos a SM_APP_NORMAL_START para repetir el proceso. En SM_APP_NORMAL_RUN comprobamos si hemos pulsado alguno de los botones y pasamos al estado

SM_APP_RUN_XX_TASK (xx indica el nombre del EP (S2, S4,D1..)) donde ejecutamos las tareas asociadas con el método que corresponda(S2TASK, S3TASK...) y pasamos al estado SM_APP_SLEEP. En SM_APP_SEP desabilitamos la pila y dejamos al dispositivo durmiendo tras habernos asegurado de no haber dejado algún mensaje a medias por enviar. El dispositivo se despierta y vuelve al estado SM_APP_NORMAL_RUN cuando pulsamos algún botón.

Si cargamos el programa en la placa o pulsamos S3 entramos en el menú de configuración tal y como hemos explicado antes. En este menú que tiene un formato muy parecido al mostrado anteriormente para el coordiandor, debemos fijar el id del nodo en cuestión y que viene indicado en la placa (pulsamos 1 y a continuación el identificador) y acceder a una red para poder enviar mensajes(pulsamos 2), estas son las tareas más importantes a realizar y una vez hechas debemos pulsar el 0 para guardar los cambios y salir del menú.

Problemas encontrados y soluciones propuestas.

Los principales problemas encontrados con está versión de la pila, han estado más relacionados con la familiarización con la arquitectura de la pila que con la funcionalidad de la misma. El objetivo que se buscaba en ésta primera implementación era conseguir una red tipo estrella que permitiera que varios dispositivos finales se conectarán a un único coordinador lo que pese a la ausencia de seguridad en el acceso a la red, la no disponibilidad de clusters ni redes peer-to-peer (solo redes cliente-servidor), la incapacidad para rutar mensajes y demás inconvenientes solucionados en gran medida en la siguiente versión, hemos podido conseguir.

4.2.2 SW para PICDEM-Z sobre la versión 1.0.3.8 de la pila de red Zigbee

La segunda versión de nuestra aplicación esta desarrollada sobre la implementación de Microchip de la pila de red para Zigbee versión 1.0-3.8 que se ajusta a las especificaciones de la Alianza Zigbee 1.0 con lo que esta implementación si es realmente una aplicación Zigbee propiamente dicha.

Las características que hacen diferente a esta versión de la pila con respecto a la anteriormente utilizada son:

- Esta certificada como implementación del protocolo Zigbee versión 1.0
- Soporta también una frecuencia de transmisión de 2.4 GHz
- Soporta todos los dispositivos permitidos por Zigbee como son los coordinadores, los dispositivos finales y los routers.
- Soporta el resto de características que tenia la versión anterior.

Como limitaciones mas importantes de esta versión, aunque no son requisitos establecidos por el protocolo de Zigbee, tenemos:

- No soporta redes de balizas, esto es solo soporta redes non-slotted en la que todos los dispositivos finales pueden transmitir en cualquier momento y no aquellas en las que los dispositivos tienen unas ranuras de tiempo preestablecidas para transmitir.
- Las direcciones de red de los nodos que abandonan la red no son reasignadas a nuevos nodos, con lo que se pierden.
- No borra las entradas de la tabla de vecinos de los nodos que abandonan la red
- No esta implementada la resolución de conflictos por varios PAN que interfieran entre si.
- La reconstrucción de rutas si existen fallos no esta automatizada.
- No esta permitido que los coordinadores establezcan mas de una red y la alternancia entre estas.

En esta versión la pila de Zigbee esta realizada sobre la implementación de la especificación del protocolo de red IEEE 802.15.4, y añade las características propias de Zigbee como son la formación de redes el rutado, uso de estándares para las funciones, gestión y detección de redes entre otras.

Zigbee utiliza las especificaciones del IEEE 802.15.4 a la hora de implementar las capas MAC de control de acceso al medio y la capa PHY o física. Este estándar define

tres frecuencias de banda para operar aunque en la implementación realizada solo se utiliza la banda de 2,4 con 16 canales (del 11-26), y se utiliza esta solo porque es la que mayor capacidad de transferencia tiene, con tasas de transferencia de 250kbps (aunque en realidad en nuestra aplicación son menores debido a los paquetes de cabecera y los retardos por procesamiento). También utilizamos esta frecuencia para transmitir debido a que es la frecuencia que soporta el emisor que lleva acoplado nuestro PicDemZ.

La máxima longitud de un paquete para IEEE 802.15.4 son 127 bytes incluidos los 16 bit de un código de redundancia cíclica para la asegurar la integridad de los paquetes.

Además utiliza paquetes de confirmación o ACK para detectar si un paquete se ha enviado con éxito. Si el paquete se transmite con el flag de ACK activado y la confirmación no llega en un periodo de tiempo, el emisor vuelve a retransmitir el paquete. Esto solo indica que la capa MAC del receptor ha recibido el paquete con éxito pero no indica si lo ha procesado bien, y puede darse que un paquete lo reciba la capa MAC con éxito pero que debido a las capas superiores ese paquete no se procese con éxito, por lo que el resto de capas debieran también implementar un mecanismo de confirmación para asegurarnos que no se pierde información en la transferencia de datos.

El estándar IEEE 802.15.4 define dos tipos de dispositivos que son los de funcionalidad completa y los de funcionalidad reducida. Los primeros poseen, como su nombre indica, toda la funcionalidad y han de estar conectados a una fuente de potencial fija, como puede ser una toma de corriente, debido a que nunca se pueden dormir porque seran los encargados de procesar los datos o de su enrutado. Los Segundos no poseen toda la funcionalidad y puede haber de muy diversos tipos en una misma red, pudiéndose alimentar con baterías debido a que estos dispositivos cuando permanezcan inactivos pasaran a un estado “sleep” en el que su consumo de energía será mínimo.

Los dispositivos que después utilizara Zigbee se basan en estos y serán de tres tipos:

- Coordinador: de funcionalidad completa, existirá uno por red y serán los encargados de generar esta red y de la gestión de las tablas de ligaduras y de vecinos así como del procesamiento de datos.
- Router: También son dispositivos de funcionalidad completa debido a que tienen que estar activos para realizar el rutado de los paquetes hacia el coordinador.

Son dispositivos opcionales en una red y permiten ampliar el numero de nodos asociados a una red y su alcance. También se pueden utilizar a la hora de realizar funciones de monitorización y control de las funciones de red.

- Dispositivos Finales: Pueden ser de funcionalidad completa o reducida utilizándose normalmente estos últimos debido a que sus funciones principales son las de observar eventos que transmiten al coordinador para que este los procese.

Debido a la posibilidad que ofrece esta implementación de tener los tres tipos de dispositivos del protocolo Zigbee, podemos configurar redes con topologías muy diversas como son la topología estrella, la topología en árbol y la topología malla.

La pila de Zigbee en la versión 1.0.3.8

Las capas físicas y de control de acceso al medio son propias del transmisor/receptor. Como en esta versión no esta soportado el transmisor CC2420 hemos tenido que implementar ambas capas basándonos en las que existían para otros transmisores, con la ayuda de el programa Zena para la configuración de la pila de Microchip, cuyo uso detallaremos mas adelante en el apartado de problemas encontrados en esta implementación.

En esta versión el tratamiento de los end points (terminología Zigbee, a partir de ahora EP) se hace mas general, por lo cual en nuestra implementación pasaremos a tener solo dos EP por dispositivo, uno para la transferencia de datos de aplicación propiamente dichos (el estado de los botones) y otro para tareas de gestión de la red que deberá aparecer en toda aplicación que se implemente sobre esta versión de la pila.

Cada tipo de dispositivo se implementa con una capa de aplicación distinta sobre la misma implementación de la pila, variando eso si, determinados archivos de configuración para definir en cada caso el tipo de dispositivo que se esta desarrollando y sus características.

Las tres aplicaciones tienen en común a nivel de capa de aplicación, la forma en la que se comunican. Esto significa que a la hora de abordar la implementación decidimos utilizar ligaduras, por lo cual todos los dispositivos tendrían que implementar una sección para el tratamiento los EP necesarios. Al igual que en la versión primera de nuestra implementación las comunicaciones se realiza mediante los EP de los distintos dispositivos, pero en este caso hemos reducido el numero de estos a dos, siendo el llamado EP_ZDO obligatorio para la gestión de la red de cualquier aplicación sobre la pila de Microchip en esta versión (mediante este EP se realizan las gestiones propias a la seguridad, comunicaciones para formación de redes, unión a estas redes, enrutamiento, etc) , y el otro llamado EP_LIGHT, el propio de nuestra aplicación, que será donde irán representados los datos que indican el estados de los botones. En esta versión al utilizar solo un EP de aplicación propiamente dicho, tenemos optimizada la tabla de ligaduras, permitiendo la existencia si hiciera falta de muchos mas tipos de dispositivos, aunque para la funcionalidad requerida no era necesario.

Como hemos dicho el EP_LIGHT será el que utilicemos para transferencia de los estados de los botones de las placas de dispositivos finales. Este EP incluye un cluster llamado ONOFFSRC que a su vez incluye cuatro atributos que se corresponden con el estado (pulsado o no) de cada uno de los cuatro botones del kit de desarrollo PicDemZ.

El coordinador leerá este EP cuando se produzca una interrupción por presencia de datos en el transmisor que vienen determinados por una interrupción. En este caso se leerán los datos que llegan al transmisor/receptor y una vez detectado que provienen del EP_LIGHT se procederá a detectar el estado de los botones y saber de esta forma cuales han sido pulsados. Además se determinara el mando del que proceden mediante la lectura de las cabeceras de estos paquetes para extraer su dirección de red, la cual habremos almacenado antes en la tabla de vecinos de este coordinador. Una vez detectados estos dos datos, se mandaran a través del puerto de comunicaciones serie RS232 al PC que se encarga de ejecutar las rutinas correspondientes al juego. A continuación se muestra el código encargado de la rutina de recepción de mensajes.

Esquema de la recepción de datos procedentes de EP_LIGHT en el coordinador

```
 case EP_LIGHT:
                    if ((frameHeader & APL_FRAME_TYPE_MASK) == APL_FRAME_TYPE_KVP)
                    {
                       frameHeader &= APL_FRAME_COUNT_MASK;
                       for (transaction=0; transaction<frameHeader; transaction++)
                       {
                          sequenceNumber         = APLGet();
                          command               = APLGet();
                          attributeId.byte.LSB    = APLGet();
                          attributeId.byte.MSB    = APLGet();
                          if(params.APSDE_DATA_indication.SrcAddress.ShortAddr.Val >= 0x796F)
                             player = params.APSDE_DATA_indication.SrcAddress.ShortAddr.Val - 0x793E;
                          else
                                                                                  player =
params.APSDE_DATA_indication.SrcAddress.ShortAddr.Val - 0x13FF;

          ConsolePutString(&player);

                          //dataType = command & APL_FRAME_DATA_TYPE_MASK;
                          command &= APL_FRAME_COMMAND_MASK;

                          if ((params.APSDE_DATA_indication.ClusterId == OnOffSRC_CLUSTER)&&
                                                                  ((attributeId.Val ==
OnOffSRC_OnOffA) ||(attributeId.Val == OnOffSRC_OnOffB) ||
                                                                  (attributeId.Val ==
OnOffSRC_OnOffC) ||(attributeId.Val == OnOffSRC_OnOffD) ))
                                                                          {

                                                                                  switch
(attributeId.Val)
                                                                                  {

          case OnOffSRC_OnOffA:

          ConsolePutROMString( (ROM char *)"A" );

          break;

          case OnOffSRC_OnOffB:

          ConsolePutROMString( (ROM char *)"B" );

          break;

          case OnOffSRC_OnOffC:

          ConsolePutROMString( (ROM char *)"C" );

          break;

          case OnOffSRC_OnOffD:

          ConsolePutROMString( (ROM char *)"D" );

          break;

          default:break;
                                                                                  }

                             if ((command == APL_FRAME_COMMAND_SET) ||
                                (command == APL_FRAME_COMMAND_SETACK))
                             {
                                // Prepare a response in case it is needed.
                                TxBuffer[TxData++] = APL_FRAME_TYPE_KVP | 1;    // KVP, 1 transaction
                                TxBuffer[TxData++] = sequenceNumber;
```

```
                        TxBuffer[TxData++] = APL_FRAME_COMMAND_SET_RES |
(APL_FRAME_DATA_TYPE_UINT8 << 4);
                        TxBuffer[TxData++] = attributeId.byte.LSB;
                        TxBuffer[TxData++] = attributeId.byte.MSB;

                        // Data type for this- attibute must be APL_FRAME_DATA_TYPE_UINT8
                        data = APLGet();
                        switch (data)
                        {
                           case LIGHT_OFF:
                              ConsolePutROMString( (ROM char *)"Turning light off.\r\n" );
                              MESSAGE_INDICATION = 0;
                              TxBuffer[TxData++] = SUCCESS;
                              break;
                           case LIGHT_ON:
                              ConsolePutROMString( (ROM char *)"Turning light on.\r\n" );
                              MESSAGE_INDICATION = 1;
                              TxBuffer[TxData++] = SUCCESS;
                              break;
                           case LIGHT_TOGGLE:
                              //ConsolePutROMString( (ROM char *)"Toggling light.\r\n" );
                              MESSAGE_INDICATION ^= 1;
                              TxBuffer[TxData++] = SUCCESS;
                              break;
                           default:
                              PrintChar( data );
                              ConsolePutROMString( (ROM char *)" Invalid light message.\r\n" );
                              TxBuffer[TxData++] = KVP_INVALID_ATTRIBUTE_DATA;
                              break;
                        }
                     }
                     if (command == APL_FRAME_COMMAND_SETACK)
                     {
                        // Send back an application level acknowledge.
                        ZigBeeBlockTx();

                        // Take care here that parameters are not overwritten before they are used.
                        // We can use the data byte as a temporary variable.
                        params.APSDE_DATA_request.DstAddrMode =
params.APSDE_DATA_indication.SrcAddrMode;
                        params.APSDE_DATA_request.DstEndpoint =
params.APSDE_DATA_indication.SrcEndpoint;
                        params.APSDE_DATA_request.DstAddress.ShortAddr =
params.APSDE_DATA_indication.SrcAddress.ShortAddr;

                        //params.APSDE_DATA_request.asduLength; TxData
                        //params.APSDE_DATA_request.ProfileId; unchanged
                        params.APSDE_DATA_request.RadiusCounter = DEFAULT_RADIUS;
                        params.APSDE_DATA_request.DiscoverRoute =
ROUTE_DISCOVERY_ENABLE;
                                                                                #ifdef
I_SUPPORT_SECURITY

          params.APSDE_DATA_request.TxOptions.Val = 1;
                                                                                #else

                                    params.APSDE_DATA_request.TxOptions.Val = 0;
                                                                                #endif
                        params.APSDE_DATA_request.SrcEndpoint = EP_LIGHT;
                        //params.APSDE_DATA_request.ClusterId; unchanged

                        currentPrimitive = APSDE_DATA_request;
                     }
                     else
                     {
                        // We are not sending an acknowledge, so reset the transmit message pointer.
                        TxData = TX_DATA_START;
```

```
                        }
                    }
                    // TODO read to the end of the transaction.
                } // each transaction
            } // frame type
            break;
```

En el caso del RFD el EP_LIGHT de comunicaciones se empleara para el envió de los datos referentes al estado de las botoneras de el dispositivo propiamente dicho. El envió de datos se ejecutara por la interrupción de presión sobre un botón, la cual previamente cambiara el estado del atributo correspondiente a este botón en el cluster SrcOnOff que será lo que se mande en el EP. Además de los datos del EP a la hora de trasmitir los paquetes se incluirá en las cabeceras los datos relativos a la dirección de red, así como otros datos referentes al uso de seguridad en la transmisión, petición de confirmación por parte del coordinador, etc.

La rutina de envío del EP_LIGHT en los dispositivos finales tiene mas o menos esta estructura:

```
if (ZigBeeStatus.flags.bits.bDataRequestComplete && ZigBeeReady())
            {

                //
************************************************************************
**
                // Place all processes that can send messages here.  Be sure to call
                // ZigBeeBlockTx() when currentPrimitive is set to
APSDE_DATA_request.
                //
************************************************************************
**
                if ( myStatusFlags.bits.bLightSwitchToggled1 )
                {
                    // Send a light toggle message to the other node.
                    myStatusFlags.bits.bLightSwitchToggled1 = FALSE;
                    ZigBeeBlockTx();

                    TxBuffer[TxData++] = APL_FRAME_TYPE_KVP | 1;    // KVP, 1
transaction
                    TxBuffer[TxData++] = APLGetTransId();
                    TxBuffer[TxData++] = APL_FRAME_COMMAND_SET |
(APL_FRAME_DATA_TYPE_UINT8 << 4);
```

```
                    TxBuffer[TxData++] = OnOffSRC_OnOffA & 0xFF;        // Attribute
ID LSB
                    TxBuffer[TxData++] = (OnOffSRC_OnOffA >> 8) & 0xFF;  //
Attribute ID MSB
                    TxBuffer[TxData++] = LIGHT_TOGGLE;

                    #ifdef USE_BINDINGS
                      params.APSDE_DATA_request.DstAddrMode =
APS_ADDRESS_NOT_PRESENT;
                    #else
                      params.APSDE_DATA_request.DstAddrMode =
APS_ADDRESS_16_BIT;
                      params.APSDE_DATA_request.DstEndpoint = EP_LIGHT;
                      params.APSDE_DATA_request.DstAddress.ShortAddr =
destinationAddress;
                    #endif

                    //params.APSDE_DATA_request.asduLength; TxData
                    params.APSDE_DATA_request.ProfileId.Val = MY_PROFILE_ID;
                    params.APSDE_DATA_request.RadiusCounter =
DEFAULT_RADIUS;
                    params.APSDE_DATA_request.DiscoverRoute =
ROUTE_DISCOVERY_ENABLE;
                                                #ifdef I_SUPPORT_SECURITY

        params.APSDE_DATA_request.TxOptions.Val = 1;
                                                #else
                        params.APSDE_DATA_request.TxOptions.Val = 0;
                                                #endif
                    params.APSDE_DATA_request.SrcEndpoint = EP_LIGHT;
                    params.APSDE_DATA_request.ClusterId = OnOffSRC_CLUSTER;

                    ConsolePutROMString( (ROM char *)" Trying to send light switch
message.\r\n" );

                    currentPrimitive = APSDE_DATA_request;
                }
```

Los routers por su parte, solamente participan en la comunicación como medio de paso y encaminamiento de los paquetes de tal manera que no realizan ningún tratamiento con los datos que se incluyen dentro del EP_LIGHT, pero si procesan las cabeceras para determinar a donde mandar el mensaje. Esta tarea también se realizara mediante comunicaciones por el EP_ZDO que como hemos dicho antes se utiliza para la gestión de la red.

Todas las aplicaciones siguen una estructura básica, en la que se ejecutan una serie de inicializaciones, entre las que se encuentran las propias inicializaciones de la pila de red, así como la de variables del programa para definir los estados iniciales de la ejecución, la habilitación de las interrupciones y la inicialización del watchdog. Posteriormente el programa principal entra en un bucle que hace la transición por los diferentes estados de una aplicación desarrollada con Zigbee.

La variable “currentPrimitive” almacena el estado actual, el cual va cambiando según la función que tenga que realizar el dispositivo. Los estados propuestos por esta versión de la implementación de la pila de Zigbee son los que aparecen el la siguiente tabla, con sus respectivos estados respuesta, aunque no significa que todas las aplicaciones tengan que tener todos estos estados.

Estados típicos de una aplicación y sus estados respuesta

Una vez realizados todos los estados para la generación de una red, unión de nodos y creación de ligaduras, los estados mas importantes son los de transmisión de datos y recepción que pasaremos a explicar brevemente a nivel general.

Al estado de transmisión de datos se llega por medio de la detección de interrupción por pulsación de tecla por lo cual se asigna el estado de transmisión y se cambian los atributos correspondientes. Una vez en este estado se carga en un buffer de transmisión los datos del estado de los EP que se van a transmitir junto con los datos referentes al modo de transmisión y a la dirección de red.

Al estado de recepción de datos se llega por interrupción debida a la presencia de datos en el transmisor/receptor. En este estado, y al estar realizando el uso de ligaduras, se comprueba primero el EP desde el que se esta transmitiendo, de tal manera que si se trata del EP_ZDO se harán todas las tareas referidas a la gestión de red, y si se trata del EP_LIGHT se realizan las acciones para detectar los datos del estado de la botonera y posteriormente se ejecutan las llamadas a las funciones que envían estos datos al PC mediante el puerto serie.

Esta misma recepción de datos en los RFD se lleva a cabo de otra manera. Este tipo de dispositivos necesita la recepción de datos para la gestión de la red. Debido a que este tipo de dispositivos permanecerán la mayor parte del tiempo dormidos, cuando pasen a un estado activo lo primero que harán será preguntar a su coordinador o router si tiene datos pendientes para enviarle, debiendo de esta forma, almacenarse en los routers y coordinadores correspondientes un buffer de mensajes preparados para enviar en el momento de ser requeridos. Si no existen datos preparados los RFD entran en estado dormido, ya que están desocupados y de esta manera ahorran energia, esperándose un determinado tiempo para despertar o la pulsación de una tecla en los mismos, en cuyo caso se despertaran para mandar el mensaje y después harán la petición de datos correspondiente a su padre.

Problemas encontrados y soluciones propuestas.

En esta versión de la implementación los principales problemas con los que nos hemos encontrado ha sido la falta de tiempo, debido a lo reciente de la versión utilizada y las dificultades encontradas debido a que esta versión no era compatible en principio con nuestro kit de desarrollo PicDemZ.

En la fecha de la publicación de esta versión el kit PicDemZ, fabricado por Microchip, había cambiado su transmisor de RF, y la nueva versión de la pila de Zigbee solo era compatible para el nuevo que es el MRF24J40 y para otro emisor/receptor RF que es el UBEC UZ2400. Por ello, y por ser esta una versión tan reciente de la cual no se disponía de prácticamente documentación, esta ha sido la principal dificultad.

Al no ser compatibles hemos tenido que realizar modificaciones en la pila de red, en las capas MAC y PHY para adaptarlas a nuestro emisor/receptor, para lo cual hemos utilizado una herramienta para la configuración de este tipo de aplicaciones, distribuida por Microchip y llamada Zena con la cual hemos conseguido configurar nuestra aplicación para que trabajara con nuestros recursos. Además de utilizar esta herramienta hemos tenido que aplicar soluciones propuestas en el foro de microchip para conseguir esta compatibilidad ya que con la configuración dada por la herramienta Zena tampoco bastaba para el correcto funcionamiento, ya que esta herramienta crea archivos de

aplicación para las capas antes mencionadas, pero había problemas de compatibilidad al intercambiar los archivos por los distribuidos originalmente en la versión 1.0.3.8 de la pila de Zigbee para el emisor/receptor MRF24J40, ya que al ser una pila de red, las funciones de la capa superior, en este caso la capa de red NWK, hacían uso de la nueva implementación de la capa MAC y hubo que solucionar también este problema.

5.CONCLUSIÓN

Como se comento en el apartado motivaciones de la introducción el objetivo inicial del proyecto era diseñar un juego multijugador utilizando una de las tecnologías inalámbricas con mayor proyección de futuro para los próximos años. Pensamos que se han alcanzado las metas que nos fijamos y que hemos sabido sacar partido a las ventajas que Zigbee ofrecía respecto a las otras tecnologías (apartado comparativa de la introducción). Aunque toda la parte inicial del proyecto se realizó sobre la versión 1.0 de la pila de red Zigbee, pues era la única disponible, hemos quedado especialmente satisfecho de la funcionalidad y resultados obtenidos con la versión 1.0.3.8.

Las ventajas de esta segunda implementación son varias respecto a la anterior, no solo el haber obtenido una aplicación que cumple con los requisitos de la especificación 1.0 de Zigbee, y todo lo que ello conlleva, como el uso de seguridad en las comunicaciones, la posibilidad de tener mas tipos de topologías, etc sino que además en esta implementación y por el uso optimizado que hacemos de las ligaduras a la hora de definir los EP, sus clusters y sus atributos, tenemos una mayor capacidad de incluir diversos tipos de nodos(en nuestro caso solo tenemos un tipo de nodo final, pero en una aplicación típica de Zigbee el tipo de dispositivos finales tiende a ser cada vez mayor). Además, por la presencia de routers en nuestra implementación aumentamos en gran medida el numero de dispositivos posibles que se unen a la red, debido a que cada router dispone de su propia tabla de vecinos y que el numero de routers solo vendrá definido por un archivo de configuración en el que se le puede determinar el numero de routers conectados al coordinador y el numero de niveles que queremos permitir. Algunos puntos como la posibilidad de conectar más de 200 mandos a un solo coordinador no ha sido posible contrastarlos, aunque si hemos podido realizar pruebas sobre la utilidad de los routers limitando el número de nodos que se podían conectar al coordinador y utilizando un router para aumentar esta capacidad.

El mayor inconveniente a sido la falta de tiempo y la poca documentación existente para esta versión de la pila de Microchip, junto con la incompatibilidad de esta con el kit de desarrollo que teníamos para la ejecución de nuestro proyecto, la cual nos ha costado bastante resolver, aunque al final creemos tener una aplicación mucho mas completa, con ,muchas mas posibilidades y robusta que la anterior.

Respecto al futuro de Zigbee como ya se ha comentado en puntos anteriores, se espera que sus módulos sean los transmisores inalámbricos más baratos de la historia, lo que permitirá su fabricación de forma masiva. Su coste aproximado estará alrededor de los 2 euros (la radio se puede fabricar con muchos menos circuitos analógicos de los que se necesitan habitualmente), y dispondrán de una antena integrada, control de frecuencia y una pequeña batería. Además su utilidad irá también aumentando de manera que no se limitará únicamente a campos como la domótica o similares donde la tasa de transferencia sea muy baja, de hecho se esta empezando a utilizar Zigbee en campos tan competitivos como la telefonía móvil(el fabricante coreano Pantech & Curitel ha presentado una demo de un teléfono móvil que soporta el protocolo Zigbee), también se puede encontrar en el mercado adaptadores que permiten utilizar la tecnología Zigbee en ordenadores y pdas.

APÉNDICE

Manual de uso

Para poder probar el proyecto, tenemos dos caminos, es decir, las dos versiones de desarrollo que se han implementado.

-Desarrollo de una red ZigBee en forma de estrella.
-Desarrollo de una red ZigBee en forma de malla.

En ambos casos, para poder probarlos, será necesario partir con la ultima versión del MPLAB y del compilador C C18 también actualizado.

Una vez que nos hemos asegurado de que amabas cosas están actualizadas, procederemos a cargar las aplicaciones en las placas.

Red en forma de estrella:

1) Abriremos el MPLAB y seleccionaremos la opción de abrir proyecto.

2) Elegiremos el proyecto del Coordinador (DemoCoordinador). En la versión que hemos generado con el nombre de versión 1.

3) Una vez cargado en el MPLAB, en el árbol de ficheros generado a la izquierda, sobre la raíz pincharemos con el botón derecho y compilaremos.

4) Una vez compilado, y asegurándonos de que el ICD2 y el PIC están conectados, pincharemos sobre la opción Programer y seleccionaremos la

opción ICD2. A continuación volveremos a esa opción y le daremos a programar.

5) Una vez programado, desenchufaremos el PIC y cerraremos el proyecto.

6) Abriremos el proyecto RFD (DemorRFD).

7) Comprobamos que el PIC que vamos a cargar no es el coordinador y repetimos los pasos hecho para el Coordinador para tantos RFD´s se quiera.

8) Una vez terminada la carga de los dispositivos, procederemos a enchufar el cable para comunicación mediante RS-232 al ordenador por el puerto COM y al PIC.

9) Abriremos el HYPERTERMINAL, cuyas opciones hay que modificar, poniendo como frecuencia 19200 baudios y dejando el resto igual. Hay que modificar las opciones para activar el “eco”.

10) Una vez hecho esto, conectaremos el PIC Coordinador y lo encendemos. Aparece un menú, en el cual hay que seleccionar el 1, escribiéndolo en el hyperterminal
    Y a continuación los cuatro dígitos que s especifican en el PIC.
    Tras esto seleccionaremos las opciones 2,3 y por ultimo el 0, quedando establecida una nueva red.

11) A continuación, conectaremos por RS-232 los RFD´s uno a uno y los encenderemos, apareciendo también un menú. Seleccionaremos en cada uno la opción 1 escribiremos sus cuatro dígitos y después la opción 2 y la 0.

12) De esta manera habrá quedado configurada por completo la red en forma de estrella.

13) Pulsamos en cada uno de ellos un botón y comprobamos que escribe el identificador del mando, así con la opción de botón seleccionada.

Red en forma de malla:

1) Abriremos el MPLAB y seleccionaremos la opción de abrir proyecto.

2) Elegiremos el proyecto del Coordinador (DemoCoordinador). En la versión que hemos generado con el nombre de versión 2.

3) Una vez cargado en el MPLAB, en el árbol de ficheros generado a la izquierda, sobre la raíz pincharemos con el botón derecho y compilaremos.

4) Una vez compilado, y asegurándonos de que el ICD2 y el PIC están conectados, pincharemos sobre la opción Programer y seleccionaremos la opción ICD2. A continuación volveremos a esa opción y le daremos a programar.

5) Una vez programado, desenchufaremos el PIC y cerraremos el proyecto.

6) Abriremos el proyecto RFD (DemorRFD) y cambiaremos el BYTE 0 de la dirección MAC que se describe en el archivo ZigBee.def y pondremos un número que no haya sido asignado a ningún RFD.

7) Comprobamos que el PIC que vamos a cargar no es el coordinador y repetimos los pasos hecho para el Coordinador para tantos RFD´s se quiera.

8) Una vez programado, desenchufaremos el PIC y cerraremos el proyecto.

9) Abriremos el proyecto Router (DemorRouter).

10) Comprobamos que el PIC que vamos a cargar no es el coordinador ni un RFD y repetimos los pasos hecho para el Coordinador.

11) Una vez terminada la carga de los dispositivos, procederemos a enchufar el cable para comunicación mediante RS-232 al ordenador por el puerto COM y al PIC.

12) Abriremos el HYPERTERMINAL, cuyas opciones hay que modificar, poniendo como frecuencia 19200 baudios y dejando el resto igual. Hay que modificar las opciones para activar el “eco”.

13) Una vez hecho esto, conectaremos el PIC Coordinador y lo encendemos.
Se autogenerará una red ZigBee.

14) A continuación, encenderemos un RFD, podremos ver quese conecta a la red un nuevo nodo. A continuación encenderemos el router y podremos comprobar que también se conecta. A continuación debemos cambiar el cable de RS-232 al router para poder comprobar como se le une los nuevos RFD´s que se vayan conectando a la red, ya que para probar esto hemos dejado al coordinador con solo 2 vecinos como número máximo. Cuando hayamos acabado, volvemos a conectar el cable RS-232 al Coordina

15) De esta manera habrá quedado configurada por completo la red en forma de malla.

16) Pulsamos en cada uno de ellos un botón y comprobamos que escribe el identificador del mando, así con la opción de botón seleccionada.

Bibliografía

Applying Pic18 Microcontroller(English) –Prentice Hall(Barry B. Brey)

Diseño practico de aplicaciones- Angulo Usategui, J.M.

Páginas de referencia

www.microchip.com

www.forosdeelectronica.com

www.monografias.com

foros.solocodigo.com

www.programacion.net

Además de estas referencias, nos hemos apoyado en multitud de foros de Internet y artículos en la red.

Palabras clave

Zigbee
PIC
MCC18
Microchip
MPLab
Malla
Estrella
Coordinador
Router
RFD