UNIVERSIDAD DE CASTILLA - LA MANCHA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES

CIUDAD REAL

PROYECTO FIN DE CARRERA Nº 09-09-200909

DISEÑO Y CONSTRUCCIÓN DE UN MICROROBOT CON CAPACIDAD PARA COMUNICACIÓN INALÁMBRICA

Autor:
ENRIQUE HOLGADO DE FRUTOS

Director:
FRANCISCO RAMOS DE LA FLOR

Co-Director:
JAVIER GARCÍA-ESCRIBANO SÁNCHEZ-PAULETE

DICIEMBRE DE 2009

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

La consecución de este proyecto pone punto y seguido a una de las mejores etapas de mi vida, en las que no sólo he crecido como ingeniero, también lo he hecho como persona. Ha sido un trabajo duro, incluso a veces nada agradecido. Pero me ha enseñado que hay que luchar por lo que se quiere conseguir y cuando se consigue es muy satisfactorio. Es un trabajo realizado por mí, pero no podría haber sido posible sin la gente que me ha apoyado a lo largo de su ejecución.

La persona que sin duda más me ha ayudado es Paco, mi director de proyecto D. Francisco Ramos de la Flor, me ha escuchado cuando lo he necesitado, ha perdido mañanas, tardes y porque no decirlo. . . noches enteras ayudándome y siempre lo ha hecho sin mirar el reloj, sin preocuparle nada más. Aún sabiendo que tenía que acabar su tesis, siempre lo ha pospuesto para más tarde por ver que me inquietaba e intentar solucionarlo. Sin él todo esto no habría sido posible. Por eso es la primera persona a la que debo agradecérselo.

Agradecer también a D. Javier García-Escribano Sánchez-Paulete, mi co-director de proyecto, el tiempo que me ha dedicado cuando lo he necesitado y con él a todos los demás profesores que en mayor o menor medida me han ayudado en lo que han podido. A todos ellos gracias.

Dar las gracias a todos mis compañeros que han compartido conmigo 5 años de malos y buenos momentos Antonio, Alberto, Ana, Andrés, Luis, Marcos, José Manuel, Guille, Gonzalo y todos los que me dejo que son muchos, espero no se enfaden.

A mis amigos que han escuchado todos mis problemas y no han hecho otra cosa más que apoyarme y demostrarme que las cosas se solucionan con esfuerzo y dedicación Álvaro, Borja, Chente, Javi, Calata, Dori y a todo mi equipo Nacho, Antonio, José Manuel son. . . mi segunda familia y a Sonia, porque es una de las personas que me ha aguantado cuando las cosas han ido mal y siempre ha sabido que decirme para apoyarme y demostrarme que las cosas al final salen.

Por último, a mi familia, porque ellos me han apoyado de manera incondicional, siempre. A mi padre porque no ha pasado un día que no me preguntara. . . ¿se movió?¿Cuándo me lo enseñas? Tengo ganas de verlo, porque ha conseguido enseñarme que con esfuerzo puedes llegar donde te propongas. A mi hermana siempre con una corta pero divertida visita al laboratorio porque pasaba por allí, un 'toc' 'toc' o cualquier cosa, porque es especial. A mi madre, porque ella más que nadie sin decirlo es la persona que más se ha preocupado porque sea feliz, y me ha dicho las palabras exactas cuando era necesario.

A todos ellos y a los que me dejo porque son muchos, gracias sin vosotros esto. . . no hubiera sido posible.

Memoria

UNIVERSIDAD DE CASTILLA-LA MANCHA

E.T.S DE INGENIEROS INDUSTRIALES

1

INTRODUCCIÓN

En este primer capítulo se expondrán los motivos que han desencadenado la construcción y posterior programación de *UCLMin*, el primer microrobot autómata desarrollado dentro del Grupo de Robótica, Automática y Mecatrónica (GRAM) de la Escuela Técnica Superior de Ingenieros Industriales de la UCLM en Ciudad Real. El proyecto ha sido desarrollado durante el curso 2008-2009, realizado por Enrique Holgado de Frutos bajo la supervisión de Francisco Ramos de la Flor.

1.1. Motivación

Un robot es una máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas[1]. Son dispositivos que desempeñan tareas automáticamente, ya sea de acuerdo a supervisión humana directa, a través de un programa predefinido o siguiendo un conjunto de reglas generales, utilizando técnicas de inteligencia artificial; generalmente estas tareas reemplazan, asemejan o extienden el trabajo humano, como el ensamblado en líneas de manufactura, manipulación de objetos pesados o peligrosos, trabajo en el espacio, etc.

[1]Definición dada por la Real Academia Española(RAE)

Existen diferentes tipos y clases de robots, entre ellos con forma humana, animal, de plantas o incluso de elementos arquitectónicos pero todos se diferencian por sus capacidades. Nosotros nos vamos a centrar en los móviles que son los que nos interesan. Estos robots tienen la capacidad de desplazarse por un entorno de manera autónoma sin necesidad de estar conectado a una unidad de control que lo guíe.

Los robots móviles se pueden clasificar por el tipo de locomoción utilizado, en general, los tres medios de movimiento son: por ruedas, por patas y orugas. Cabe señalar que aunque la locomoción por patas y orugas han sido ampliamente estudiadas, el mayor desarrollo se presenta en los Robots Móviles con Ruedas (RMR).

"Un sistema electromecánico controlado, que utiliza como locomoción ruedas de algún tipo, y que es capaz de trasladarse de forma autónoma a una meta preestablecida en un determinado espacio de trabajo."

Dentro de los atributos más relevantes de los RMR, destacan su eficiencia en cuanto a energía en superficies lisas y firmes, a la vez que no causan desgaste en la superficie donde se mueven y requieren un número menor de partes y menos complejas, en comparación con los robots de patas y de orugas, lo que permite que su construcción sea más sencilla.

Son precisamente estos argumentos los que motivan el análisis de este tipo de robots. Estas características resultan de gran interés a la hora de construir robots capaces de realizar tareas de modo cooperativo.

Otro concepto fundamental es el de cooperación. La cooperación consiste en el trabajo en común llevado a cabo por parte de un grupo de personas o entidades mayores hacia un objetivo compartido, generalmente usando métodos también comunes, en lugar de trabajar de forma separada en competición.

La cooperación aparece de manera natural en cualquier sociedad o grupo de individuos que, de forma individual no pueden conseguir un objetivo. Un ejemplo claro de esto es el que se lleva a cabo en los enjambres de abejas en los que cada una de ellas tiene un papel único y a la vez fundamental para el buen desarrollo de la colmena. En una colmena es importante desde la reina que es la única fértil, hasta la obrera que se encarga de trasportar polen para poder obtenerse la miel con la que serán alimentados los nuevos miembros del panal.

Figura 1.1: Cooperación en un panal

Al igual que las abejas, los robots pueden ser programados para realizar, de forma cooperativa entre varios de ellos, tareas que uno solo no podría llevar a cabo[[*Bayindir y Sahin, 2005*]]. Por ello necesitamos un robot con capacidad de comunicarse con otros robots y/o computadores que organicen dichas tareas y con la suficiente potencia de cálculo como para realizar algoritmos sencillos que le permitan esquivar obstáculos sin necesidad de un comando previo o en tiempo real desde el ordenador, coordinación con otros dispositivos, etc.Para realizar dicho intercambio de información entre robots y con una unidad central de coordinación, lo ideal es realizar una comunicación inalámbrica, la cual permite una mayor autonomía además de reducir los obstáculos que aparecerían si la comunicación fuese por cable.

Según el artículo de Efraín Mariscal García [*Mariscal García, 2005*], podríamos llegar a definir los robots cooperativos de la siguiente forma.

"Los Robots cooperativos son un grupo de robots capaces de realizar cooperando unos con otros una actividad. Cada robot sabe interrelacionarse con los demás cuando estos necesiten apoyo de él, y también puede pedir ayuda de los demás cuando sea requerida."

Los robots que forman parte de un sistema multi-robot son simples en términos de diseño y control(individualmente), y menos costosos que los sistemas de un sólo robot especializado. Dichos sistemas están orientados a resolver problemas en los cuales la participación de un solo robot no es suficiente o resulta ser muy costosa, en términos de diseño y tiempo. Podemos ver en la Figura 1.2 lo sencillo que es levantar una columna entre dos robots cooperando:

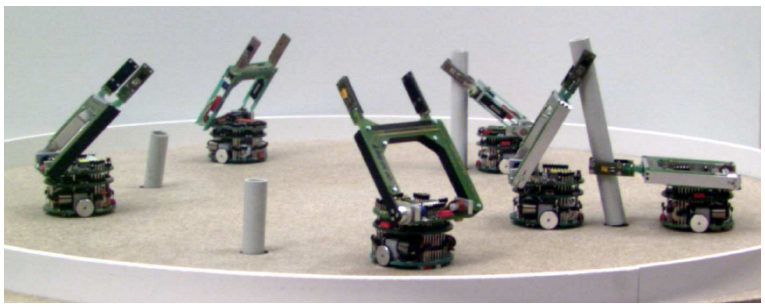


Figura 1.2: Robots cooperando

Este proyecto se engloba dentro de unas experiencias previas en la creación de una línea de investigación en robótica cooperativa. La motivación para crear este robot móvil es tener el prototipo a partir del cual se realizarán dichas tareas.

El presente proyecto presenta el diseño del RMR que constituye la unidad fundamental sobre la que se pretende basar una línea de investigación en robótica de cooperación que aborde los problemas fundamentales que se presentan en la realización de tareas por parte de un conjunto de entidades autónomas (robots móviles).

Estos robots deben tener capacidad para moverse por su entorno, y reconocer lo que les rodea gracias al conjunto de sensores del que se les va a dotar. Se buscará obtener la mayor información posible del entorno por el que se mueven. Para ello necesitamos que tengan algún tipo de conexión entre ellos (la mencionada comunicación inalámbrica) y con la unidad central de coordinación que en este caso será un ordenador. Éste podrá recibir, almacenar, distribuir y posteriormente utilizar los datos obtenidos para organizar las tareas requeridas.

1.2. Objetivos

El objetivo principal del proyecto es el de diseñar y construir un robot de tamaño reducido con capacidad para interactuar con el entorno y a su vez que pueda enviar y recibir información de manera inalámbrica con un PC. Realizar esta conexión es el primer paso para una fase futura, en la que la misma conexión nos permitirá hacer interactuar varios robots.

Como objetivos secundarios, y partiendo de un diseño ya existente, se han desarrollado en él, los siguientes avances:

- Sistema de alimentación autónomo mediante baterías de larga duración Litio-Polímero (LiPo).
- Utilización de una interfaz actual y moderna para conectar físicamente con el PC en tareas de programación-configuración.

- Por último, se busca que el robot sea barato y de sencillo montaje, permitiendo así construir más de uno y poder hacerlos interactuar.

1.3. Definición

Se van a estudiar las mejores soluciones ya existentes en el mercado para los problemas que plantea el diseño y la construcción de un microrobot. Desde el controlador que se va a utilizar al módulo inalámbrico que se va a montar, pasando por la manera de conectarlo al PC o los sensores necesarios para que sea autónomo.

Aunque el objetivo es claro, la placa de control del mismo es lo más genérica posible. De esta manera no sólo se puede manejar un robot sino que, en un momento dado, la placa puede llegar a ser plataforma de trabajo para mover hasta cuatro servos de un mecanismo cualquiera.

La ejecución del proyecto se ha dividido en una serie de fases:

- En una inicial se ha estudiado el estado del arte en cuanto a plataformas comerciales de robótica móvil de las características necesarias, de esa manera hemos seleccionado la estructura básica del mismo.
- Una vez seleccionada la estructura más adecuada a nuestras necesidades entramos a valorar la cantidad de información que se va a enviar y recibir de manera inalámbrica ya que eso nos ayudará a seleccionar componentes como el módulo de radiofrecuencia y el protocolo a utilizar.
- Al tener definida la comunicación inalámbrica y sus necesidades podemos cubrirlas con las especificaciones del micro y decidir cuál es el más aconsejable para la placa.
- Tras haber desarrollado la placa de manera genérica, se procederá a realizar una aplicación más específica. La aplicación a desarrollar es el movimiento autónomo del robot y la monitorización de los sensores que lleva acoplados.
- Finalmente se realiza una aplicación donde el microcontrolador gobierna las diferentes funciones de un Automated Guided Vehicle (AGV, Vehículo guiado automáticamente) [*Berman, Schechtman y Edan, 2009*]. Podemos definir los AGV en la industria como vehículos equipados con dirección automática que van por un camino prefijado, y que se paran en cada máquina o estación de ensamblaje para carga y descarga de mercancías, bien sea de forma manual o automática.

1.4. Estructura de la memoria

Este proyecto se va a dividir en siete capítulos con distintos subapartados. La descomposición de estos capítulos ha sido efectuada de la siguiente forma:

- El capítulo uno comprende la introducción efectuada hasta este punto, comprendiendo la motivación que ha llevado al autor a realizarlo, el objetivo final del mismo, la definición del proyecto en la que se hace un breve resumen de los pasos seguidos hasta la consecución de los objetivos, y la propia estructura de la memoria.
- El capítulo dos corresponde a los antecedentes en la tecnología existente en robótica móvil tanto dentro de la misma ETSII de Ciudad Real como otras plataformas que de alguna manera han influido en el diseño final de UCLMin, el primer robot cooperativo del GRAM.
- El capítulo tres mostrará tanto la estructura física, o arquitectura hardware, que se propone para el diseño y construcción de nuestro robot autómata, UCLMin, como la arquitectura del programa utilizado con el mismo, o arquitectura software.
- El capítulo cuatro aborda los elementos introducidos en la placa para darle autonomía, facilidad de conexión con el PC tanto de manera inalámbrica como por cable.
- El capítulo cinco trata sobre la programación del robot, se parte de una programación básica y se llega al control de envío de señales desde el robot hasta la unidad de y en concreto del módulo inalámbrico.
- El capítulo seis corresponde al presupuesto necesitado para la construcción y posterior desarrollo del robot y su comunicación con el PC.
- El capítulo siete expone los resultados y conclusiones obtenidas y se proponen distintas líneas de investigación para el futuro.

Además de los capítulos de la memoria se deben mencionar los distintos anexos realizados:

El anexo A corresponde a los esquemáticos de las placas que van montados sobre el robot, para facilitar la compresión del la descripción del Capítulo 4.

El anexo B intenta recoger a modo de resumen un tutorial de cómo utilizar las distintas aplicaciones necesarias para el diseño y programación del robot.

El anexo C explica de manera más extensa los microcontroladores estudiados, una explicación de ellos y una comparativa a modo de resumen de sus características más generales.

El anexo D describe los tipos de comunicación inalámbrica estudiadas. Se hace una comparación de las características que más nos interesan para el proyecto.

El anexo E es el más extenso ya que incluye los códigos de programación, las librerías y archivos de definiciones necesarias para que la comunicación inalámbrica sea posible.

El anexo F está compuesto por las hojas de características de los elementos auxiliares de la placa reguladores, conectores, etc.

El anexo G trata el tema de la conversión Binario a Hexadecimal, ya que los microcontroladores trabajan con los dos sistemas y se hacen cambios de sistema a lo largo de los programas.

2

ANTECEDENTES DE ROBOTS AUTÓNOMOS

En este capítulo se hará un repaso a distintas plataformas de robótica móvil comerciales y, las que podemos encontrar en la Universidad de Castilla la Mancha.

2.1. UCLM

Los antecedentes en la ETSII de Ciudad Real sobre robótica cooperativa son inexistentes. Sin embargo, sí hay experiencias previas en el campo de la robótica móvil:

2.1.1. QuiXote

El primer robot móvil con el que se ha trabajado en la ETSII de Ciudad Real es el QuiXote. El proyecto inició en 1999 y duró tres años (hasta 2001).Este robot, comprado a la empresa Real World Inc., se programó con la intención de convertirlo en un guía autónomo de museos y edificios públicos. Para ello había que darle unas ciertas características:

- Capacidad para interaccionar con humanos, para que pudiera hacerlo se le introdujeron elementos de reconocimiento y síntesis de voz. Esto le permitía:
  - Ejecutar órdenes sencillas
  - Mantener conversaciones
  - Informar del estado del robot

  El software que se utilizó tanto para el reconocimiento como para la síntesis es de IBM: IBM Via Voice.
- Interfaz amigable, se le dotó de un aspecto humanoide(Figura 2.1). Se diseño una cabeza en fibra de vidrio sobre la que se hizo un recubrimiento de fibra epoxi y se pintó de un color similar al de la piel. Además se le añadió una boca con un analizador de espectros de frecuencia y una matriz de led que simulaba la pronunciación de las palabras.

Figura 2.1: Aspecto humanoide del Quixote

- Capacidad de movimiento ante obstáculos imprevistos. Se dotó de un sistema de navegación basándose en un mapa del entorno en el que el robot tenía situados y localizados los obstáculos fijos (paredes, muebles, etc.). Para los obstáculos imprevistos el robot iba dotado de una serie de sensores que le permitían en un momento dado esquivarlos.

- Teleoperación desde la web, el QuiXote es capaz de recibir información sobre el entorno a través de las cámaras que lleva incorporadas, él mismo comprime la imagen a jpg y la envía cada 5 segundos. Esto permite a través de comandos de movimiento y un programa implementado en java mover el robot en tiempo real.
- Poder hacer una localización en el entorno. Además poseía un sistema de localización basado en balizas(Figura 2.2), el robot buscaba la imagen, la centraba y ampliaba y calculaba su posición. A partir de eso generaba una trayectoria por la sala.

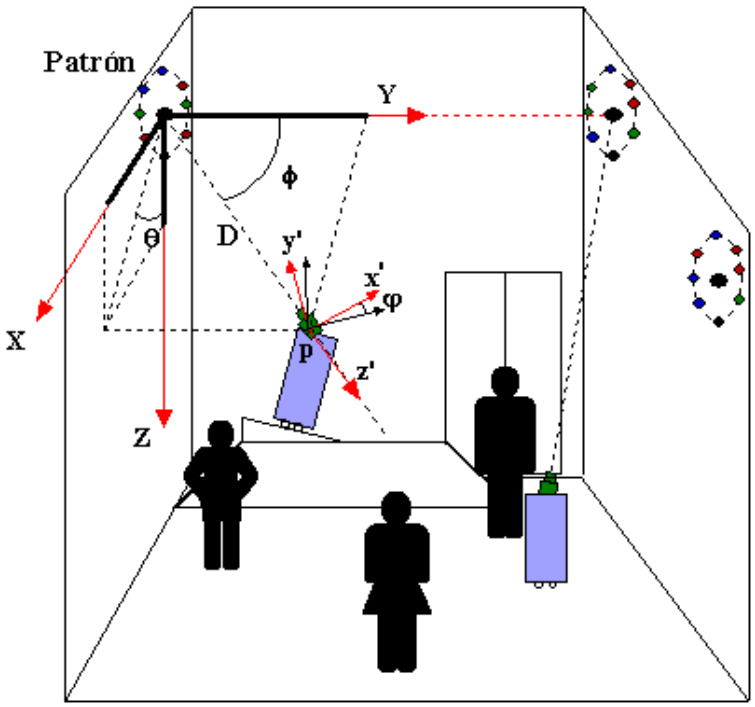


Figura 2.2: Método de localización por balizas del QuiXote

Para que el robot pudiese tener esas características y poder moverse por el edificio a mostrar de manera autónoma e independiente estaba dotado de distintos elementos:

- 2 cámaras SONY XC-999
- Cinturones de sensores de distintos tipos:
  - Ultrasonidos
  - Infrarrojos
  - Colisiones
- Encoders para los tipos de movimientos:
  - Traslación
  - Rotación
- 2 Pentium MMX 233 MHz
- SO Linux Red Hat 5.2
- Tarjeta multipuerto serie AccessBus
- Ethernet bridge Aironet

Figura 2.3: QuiXote

2.1.2. AGV de Miguel D. García Jiménez

Corresponde al [*pfc Nº 08-09-200182, 2008*], el cual se llevó a cabo en el curso académico 2007-2008. En la aplicación objeto de este proyecto, se pretende la implantación de la controladora desarrollada hasta ahora en un vehículo radio control para transformarlo en un AGV[1]. El vehículo radio control a transformar era el de la Figura2.4.Tanto el camión como los palés[2] de carga están a escala 1:20.

[1]Un AGV es un Automatic Guide Vehicle, o lo que es lo mismo un Vehículo guiado automáticamente
[2]Éstos cumplen con el estándar del palé europeo (800x1200 mm)

Figura 2.4: Foto del camión teledirigido

El tamaño era importante en esta controladora, ya que debe incluirse en el camión y, como se muestra en la imagen 2.4, el único espacio disponible para esto ocurra era la cabina. Las medidas del hueco disponible en su interior era 154x85 mm.

Para este proyecto se utilizó un microprocesador de la marca Microchip de la familia de los 18F, en concreto el 18F2455. Este micro dispone de transceptor de USB interno, pero se le suprimieron las comunicaciones por el protocolo USB con el fin de poder aprovechar estos pines como entradas y salidas.

Para alimentar el camión se utilizaron dos baterías en serie de 6 V, ya que con una no era suficiente porque se necesitaba una tensión superior al corte que era de 5 V. Al colocar las dos baterías en serie teníamos una alimentación de 12 V.

El AGV poseía además varios tipos de sensores que podemos describir:

- *Sensores infrarrojos*, estos sensores se utilizaron para la placa auxiliar que el AGV utilizaba para seguir líneas. Los sensores utilizados para este proyecto son de la marca OPTEC y el modelo es el OPB608-B. Estos sensores consisten en un diodo infrarrojo emisor y un fototransistor NPN. Éstos van montados sobre un encapsulado de resina epoxi filtrante para reducir la luminosidad ambiental no deseada. Esta resina que los recubre es una resina muy utilizada en electrónica por sus propiedades aislantes y en recubrimiento de circuitos integrados para la prevención de cortocircuitos, humedad, polvo, etc. La placa que se montó contaba con tres sensores infrarrojos como los descritos.

- *Sensores ultrasonidos* para estos sensores se utilizó un módulo comercial. Éstos venían montados en su propio integrado. El módulo es el SFR08 adquirido en `www.superrobortica.com`. Estos sensores se conectan por I2C al microcontrolador. La aplicación más importante en medición de distancia era al

aplicarlo a la robótica, utilizándolos como sistemas sónar para impedir colisiones.

2.1.3. Robot trepador

Este proyecto [[*Fernández y Feliu, 2008*]] comenzó en Diciembre de 2006 y sigue trabajándose en él en la actualidad. La aplicación objeto de este proyecto, es desarrollar un robot trepador que pueda transportar un sensor de ultrasonidos para realizar las medidas de espesor en toda la superficie de un tanque de almacenamiento de combustible. Se debe prestar una especial atención al diseño de esta máquina, para prevenir una posible caída del robot, para asegurar la estabilidad y para permitirle escalar. En esta sección se detallará el diseño del sistema robótico, que se encuentra dividido en dos partes:

- Las ruedas
- El cuerpo del vehículo

2.1.3.1. Diseño de las ruedas

Las ruedas se diseñan de forma que la fuerza ejercida sea la máxima posible. Con el mismo material y el mismo volumen de un imán, se pueden obtener fuerzas de adherencia diferentes. Se componen de varios elementos, en primer lugar se tiene un cilindro de Nylon en el centro de la estructura, éste tiene 12 orificios que serán los encargados de alojar los imanes de neodimio-hierro-boro; dos anillos de acero para conducir el flujo magnético; y tres anillos de goma alrededor de la estructura de nylon para incrementar la fricción entre las ruedas y la superficie de rodadura. Con el uso de la estructura de nylon con pequeños imanes se aumenta el radio de la rueda con un reducido incremento de peso[3]. Podemos ver la rueda en la Figura 2.5

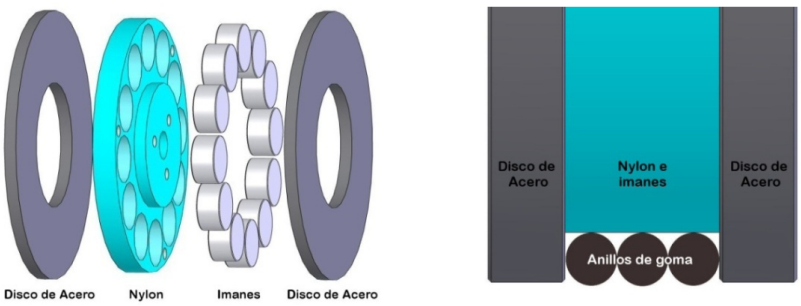


Figura 2.5: Estructura de la rueda del robot trepador

[3]ya que la densidad del nylon es de 1.15 g/cm^3.

Cuando las ruedas están en contacto con una superficie ferromagnética aparece una fuerza (fuerza magnética) entre los anillos de acero y la superficie ya que el flujo magnético se cierra. Como resultado del contacto, se produce una deformación en los anillos de goma, generando una fuerza opuesta a esa fuerza magnética. Como podemos ver en la siguiente Figura 2.6

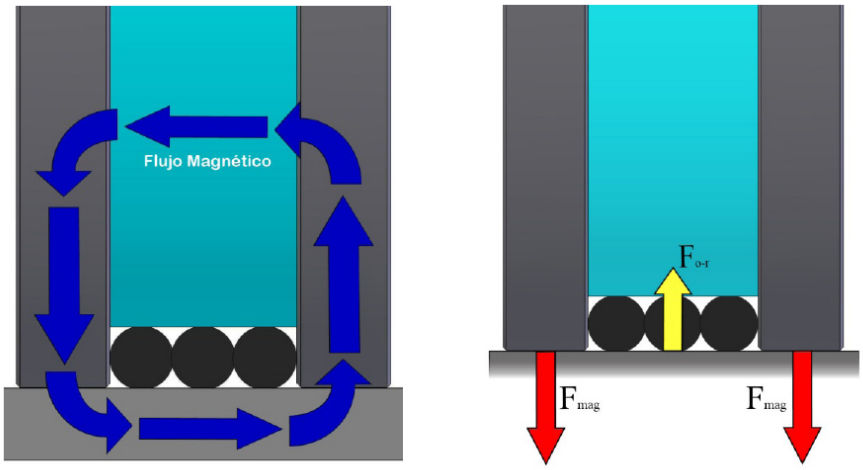


Figura 2.6: Fuerzas generadas en las ruedas

2.1.3.2. El cuerpo del vehículo

Para el cuerpo del robot se ha elegido una configuración en triciclo debido a su ligereza, simplicidad, su gran superficie de operación del sensor de ultrasonidos, su adaptabilidad a superficies no planas y la falta de necesidad de usar un cordón umbilical. El diseño consiste en tres unidades de ruedas motorizadas e independientes, cada una de ellas lleva instalado un motor de corriente continua de la marca Maxon. La rueda frontal tiene un motor adicional que le permite girar sobre su propio eje para dirigir el robot como podemos ver en la siguiente Figura 2.7

El tamaño total del robot es alrededor de 800x750x250 mm^3. El área central se utiliza para la instalación del sistema de inspección, que consiste en un robot cartesiano con un sensor ultrasonidos rodante. Podemos verlo en la Figura 2.7. Está compuesto de tres guías lineales más rápida y reduciendo el número de movimientos del robot provocando una reducción del consumo de energía debida al movimiento. La fuente de alimentación y los componentes electrónicos están montados en una plataforma que se encuentra por encima del sistema de inspección.

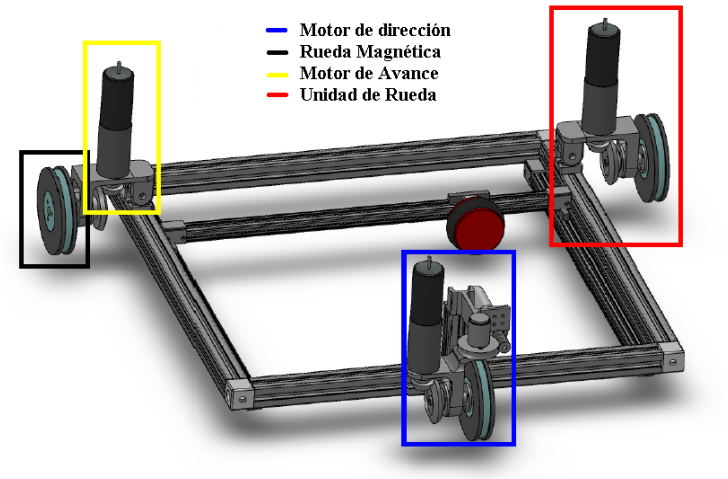


Figura 2.7: Robot con el sistema de inspección

La masa estimada del robot es de unos 10 Kg incluyendo las unidades de rueda y la estructura, a lo que habrá que añadir los sensores, controles y baterías, incrementando la masa hasta unos 21 Kg. Se puede hacer una estimación de los pesos como en la Tabla 2.1

Componentes	**Peso(Kg)**
Partes estructurales	5,2
Partes estructurales	3
3 Ruedas magnéticas	3
3 Motor CC	0,15
Motor de dirección	0,4
Sensor de ultrasonidos	1,8
Guías lineales motorizadas	2
Baterías	6
Peso total	**21,55**

Tabla 2.1: Tabla de pesos de los componentes

2.2. Skybot

Este robot(Figura 2.8) ha sido diseñado, construido y programado íntegramente en la Universidad Autónoma de Madrid (UAM) por Juan González Gómez, profesor ayudante en la Escuela Politécnica Superior en la UAM, en colaboración con Andrés Prieto-Moreno Torres entre otros.

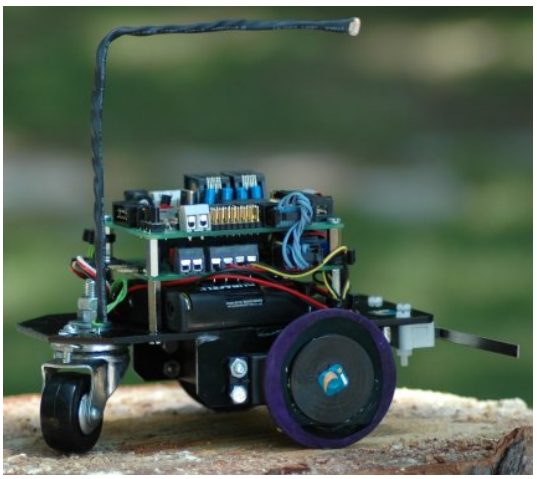


Figura 2.8: Skybot

El Skybot se desplaza mediante dos ruedas motrices y una rueda loca (o castor wheel). Está dotado de sensores para reaccionar ante estímulos del medio. Es un robot "hola mundo" ya que está pensado para aquellos que quieren iniciarse en el mundo de la robótica y los microcontroladores. Por ello, a partir de este robot es muy fácil desarrollar otros más complejos.

La estructura mecánica está compuesta por cuatro piezas de metacrilato, dos servos Futaba 3003 a los que previamente se les ha quitado la electrónica para que actúen como motores de corriente continúa y una rueda loca, la cual nos da estabilidad y una mayor facilidad en el manejo. Ha sido diseñada por Andrés Prieto-Moreno Torres. Se persiguió el objetivo de hacer una estructura fácilmente replicable en diferentes materiales como por ejemplo madera, PVC, etc.

Las piezas de la estructura se unen entre ellas con pegamento de contacto y los motores se sujetan mediante tornillos normales de métrica cuatro.

En cuanto a la electrónica el Skybot utiliza las tarjetas SKYPIC como procesadora y SKY293 como tarjeta de potencia. Ambas se unen formando una torre, colocándose encima de la estructura de metacrilato. Esta electrónica es independiente de la estructura.

Los sensores con los que el Skybot trabaja son siete, de tres tipos distintos:

- *Sensores infrarrojos* (Figura 2.9), estos sensores son los CNY70 es un sensor de infrarrojos de corto alcance basado en un emisor de luz y un receptor, ambos apuntando en la misma dirección, y cuyo funcionamiento se basa en la capacidad de reflexión del objeto, y la detección del rayo reflectado por el receptor. El Skybot lleva 4.

  El CNY70 tiene cuatro pines de conexión. Dos de ellos se corresponden con

el ánodo y cátodo del emisor, y las otras dos se corresponde con el colector y el emisor del receptor. Los valores de las resistencias son típicamente 10K ohmios para el receptor y 220 ohmios para el emisor.

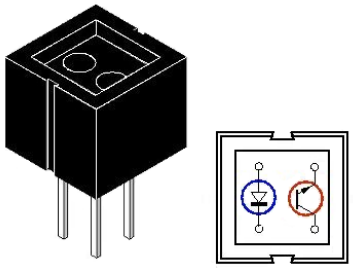


Figura 2.9: Sensores infrarrojos CNY70

El único inconveniente del sensor es la necesidad de tener que situarlo muy próximo al objeto para detectar correctamente la reflexión. Por lo demás, es una solución muy buena para la detección de línea e incluso para emplearlo como encoder para la medición de las vueltas dadas por las ruedas del robot.

- *Bumpers o sensores de contacto* (Figura 2.10), es un conmutador que tiene dos posiciones con muelle de retorno hacia la posición de reposo y con una palanca de accionamiento. Esta palanca será de mayor o menor longitud en función del modelo de bumper.

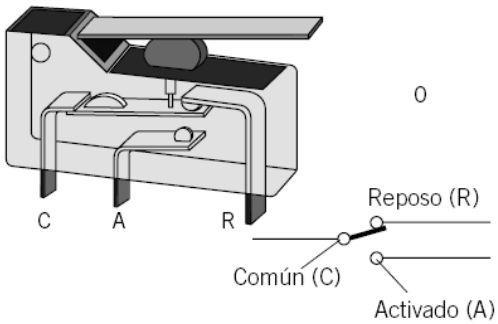


Figura 2.10: Sensores de contacto o bumpers

Cuando el bumper está en reposo, sin accionar, el terminal o patilla común (C) y el terminal R están en contacto. Cuando se aplica presión sobre la palanca, el terminal C entra en contacto con el terminal A: el bumper pasa a la posición activo. En ese momento se oye un clic que nos indica el contacto entre terminales, lo que ocurre cuando la palanca llega prácticamente al final de su recorrido. El Skybot lleva dos.

- *LDR*[4] *o sensor de luz* (Figura 2.11), son como su nombre lo indica, resistencias cuyo valor varía de acuerdo al nivel de luz al que están expuestas. El valor de resistencia eléctrica de un LDR es bajo cuando hay luz incidiendo en él (puede descender hasta 50 Ω) y muy alto cuando está a oscuras (varios $M\Omega$). El Skybot lleva uno.

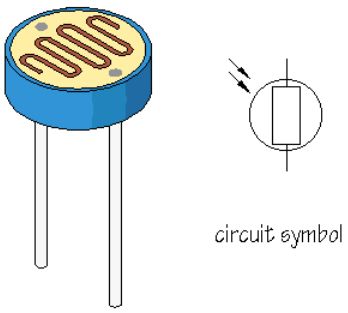


Figura 2.11: Sensor LDR

La electrónica de este robot ha sido la base de desarrollo para nuestro UCLMin como ya se detallará en los Capítulos 3 y 4.

2.3. Otras plataformas

En esta sección se describen otras plataformas que se han estudiado previamente al desarrollo del UCLMin.

2.3.1. SR1

El SR1 (Figura 2.12) es un robot multifuncional de desarrollo y aprendizaje. El SR1 es la plataforma idónea donde hacer todo tipos de proyectos desde un simple guiado por colisión, hasta un avanzado robot radio controlado con sistema de telemetría y capaz de enviar audio, vídeo y datos de forma inalámbrica de la misma forma que lo haría un robot como los que se mandan a explorar el espacio.

[4](Light Dependent Resistor, o Resistencia Dependiente de la Luz)

Figura 2.12: Robot SR1

El robot SR1 cuenta con un chasis lo suficientemente robusto para proteger todos los componentes mecánicos y electrónicos del robots mientras se desplaza en cualquier entorno interior. El chasis admite ampliaciones como plataformas de carga, techos con sensores, motores dc, ruedas de sumo, etc. Desde el punto de vista de la electrónica, se ha buscado un compromiso entre versatilidad de funciones y facilidad de programación que le permita disponer de gran cantidad de sensores, además de poder incluir accesorios extras como cámaras, servos, etc.

Se ha perseguido que todo el conjunto una vez montado permita hacer modificaciones, configuraciones, ampliaciones, cambios de sensores, etc. de forma muy sencilla.

El resultado obtenido es un robot compacto, robusto y duradero con múltiples y avanzadas posibilidades. Además de su versatilidad y su gran cantidad de accesorios que incluyen ruedas todo terreno, torreta móvil, cámara inalámbrica, radio módem, etc.

Normalmente este tipo de robot solo permite hacer pequeños montajes electrónicos ya que emplean placas de montaje de prototipo para ello, lo que significa que en el momento que el robot toca o se engancha con algún objeto se provocan averías y cortocircuitos. El Robot SR1 se ha diseñado con la idea de que pueda desenvolverse de forma autónoma y segura en cualquier tipo de entorno interior, siendo capaz de eludir y superar los obstáculos y trampas que encuentra a su paso gracias a su gran cantidad de sensores. El robot cuenta con distintos tipos:

- Dos Sensores de contacto.
- Un Sensor de inclinación.
- Dos Sensores de luz.
- Un Sensor de infrarrojos modulados.
- Un Sensor de distancia pos ultrasonido + un sensor de luz central.
- Un Sensor de temperatura digital.
- Un Sensor brújula digital.

Existen sensores adicionales que se conectan igualmente en el circuito mediante cables, pero que no van montados directamente sobre el circuito, sino que se colocan en otras partes del chasis. El más popular de ellos es el sensor de líneas, que permite al robot seguir el trazado de una línea pintada en el suelo.

En cuanto al chasis esta hecho de material PVC de 5mm de color amarillo ensamblado mediante tornillos autoroscantes y sin pegamentos, por lo que pueden desmontase cuando es necesario. Plataforma superior para placa de prototipos, carga útil, etc. que puede reemplazarse por otra con alojamiento para un servo motor, o bien por una plataforma que cubre todo el circuito y proporciona una gran base para otros montajes. Paragolpes y ruedas integradas en la carrocería que evitan que se enganche con objetos, esquinas, etc. El paragolpes es prácticamente indestructible y cubre íntegramente todo el frontal del robot, por lo que es efectivo incluso con patas de sillas y pequeños objetos.

La alimentación es dual incluye dos reguladores de tensión independientes para los motores y para la electrónica. Se puede seleccionar entre 4 posibles combinaciones de alimentación con una o dos baterías independientes. El modo estándar incluye alimentación mediante 6 pilas AA de 1,5V.

El SR1 se gobierna con el BasicX-24P, es el microcontrolador programable en Basic más potente del mercado. El BasicX-24 reúne en un circuito del tamaño de un chip de 24 patas a un microcontrolador Atmel, una memoria EEPROM de 32Kbytes, un regulador de tensión y un puerto RS232 de forma que basta con escribir los programas en Basic en el PC y luego volcarlos al módulo, para que este los ejecute de forma totalmente autónoma. Puede alimentarse desde cualquier tensión entre 5 y 24 V usando el regulador interno, o bien una fuente regulada entre 3 y 5.5V.

2.3.2. Bioloid

Bioloid (Figura 2.13) es una plataforma robótica modular de la compañía ROBOTIS, se podría decir que es una de las plataformas más completas de robótica modular. Con bioloid podemos llegar a crear un gran número de robots distintos, Robotis nos ofrece de forma guiada 26 maneras de hacerlo, desde el uso de 4 servomotores, hasta el nivel más alto con 18 servomotores, podemos crear desde un humanoide, hasta un dinosaurio, y gracias a su modulo de sensores podremos interactuar con el de manera sencilla.

En nuestro caso nos hicimos con un Bioloid Comprehensive Kit. Es una plataforma ideal y mucho más avanzada para el desarrollo de nuestros robots. Dispone de un total de 18 servos, sensores de proximidad y luminosidad hacia delante y hacia los lados, un micrófono y un pequeño altavoz.

Bioloid comprehensive kit nos permite de manera guiada crear hasta 26 robots diferentes y probarlos con programas de control de ejemplo.

Contenido del kit:

- 1 x CM-5 (módulo controlador basado en el Atmel ATMega128 a 16 MHz)

- 18 x AX-12 (Servomotores Dynamixel controlados en serie).
- 1 x AX-S1 (módulo sensor del robot).
- Mas de 100 piezas mecánicas, ruedas, neumáticos para el ensamblaje con los servos (Comprehensive Frame Set).
- 1 x Puertos de conexión serie Batería recargable (9,6V, 2,3Ah, NiMH).
- Alimentador de potencia.
- Cable serie RS-232 de 9 pines.
- CD-ROM con Software de programación, vídeos, manuales, etc.
- Tornillería, tuercas, espaciadores.
- Manuales impresos en inglés con instrucciones de montaje y guía de usuario.

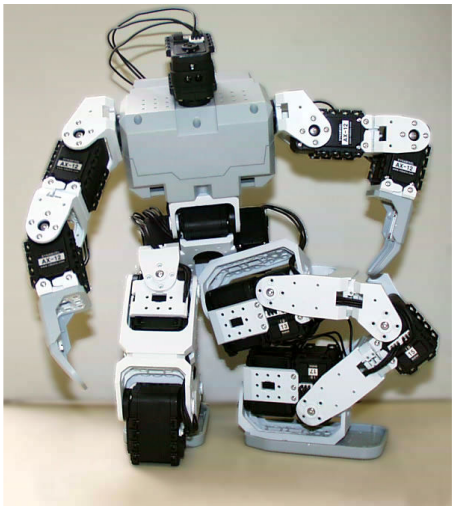


Figura 2.13: Bioloid

Nos intereso este robot móvil, además de por su versatilidad a la hora de poder hacer distintos montajes, por su modo de comunicación inalámbrica bien con el pc bien con otros bioloid, ya que el Bioloid es una de las plataformas que permite este tipo de comunicación,piedra angular de la robótica cooperativa.

3

ARQUITECTURA DEL SISTEMA ELEGIDO

En este tema se mostrará tanto la estructura física, o arquitectura hardware, que se propone para el diseño y construcción de nuestro robot autómata, UCLMin, como la arquitectura del programa utilizado con el mismo, o arquitectura software.

El capítulo anterior nos ha servido para estudiar y evaluar arquitecturas Hardware en las que basarnos a la hora de diseñar nuestro microrobot. En función del Hardware que elijamos se procederá al diseño de una arquitectura Software acorde a sus características

3.1. Arquitectura Hardware

Tenemos varias opciones estudiadas sobre las que podemos basar nuestro nuevo hardware. Todos los robots estudiados tienen prestaciones similares a las que queremos conseguir.

Como ya se ha mencionado, nuestro objetivo es realizar un microrobot que además de funcional y sencillo de manejar, sea barato. De esta manera el Bioloid

(Figura 2.13) queda descartado ya que, aunque es un robot con comunicación inalámbrica integrada, es costoso, además su placa es difícilmente reproducible en nuestra máquina de prototipado de PCB, por lo que todos los componentes deberían comprarse al proveedor, y el coste total de un grupo de robots sería elevado.

El SR1 (Figura 2.12) es más sencillo que el Bioloid y más barato. Lo que abre sus posibilidades a ser objeto de desarrollo. Pero si se estudian sus componentes detenidamente, se puede observar que este robot tiene gran cantidad de sensores que para nuestro robot son totalmente prescindibles. Debido a ésto el micro que el robot utiliza es muy potente (como podrá verse en el Anexo III Comparativa de Microcontroladores). Un micro potente implica mayor complicación en la programación y que su precio sea más elevado y se aleja de nuestros intereses.

Por otro lado, las experiencias previas de robótica móvil desarrolladas en la UCLM no se ajustan a los requisitos del sistema que deseamos diseñar, por lo que no serán de gran utilidad.

El Skybot es el que cumple gran parte de nuestras expectativas: posee una estructura[1] bastante sencilla a la vez que robusta y trabajada, que permite el movimiento del robot por superficies sin provocar en ésta ningún tipo de daño, ya que utiliza ruedas como medio de locomoción. Además posee una cantidad suficiente de sensores, como se vio en la sección dedicada al mismo en el capítulo anterior lo que facilita el reconocimiento del entorno en el que se mueve.

En conclusión, de todos los robots presentados en la sección anterior, el robot que se ajusta al perfil diseñar es el Skybot. Por ello se va a tratar su arquitectura hardware antes de pasar a la del UCLMin, ésta nos ayudará a comprender mejor la de éste último.

3.1.1. Arquitectura hardware Skybot

En nuestro caso la arquitectura considerada (Figura 3.1) inicialmente es la que lleva el Skybot (Figura 2.8). Este robot es de manera muy básica nuestro objetivo. Por eso nos hemos familiarizado con él en una fase inicial para más tarde hacer un diseño propio basándonos en lo aprendido sobre éste.

Este pequeño robot está provisto de un microcontrolador que controla el robot. A través de unos buses de datos se conecta a la placa Sky293 (Figura 4.12) por la que puede recibir información sobre los distintos tipos de sensores que lleva y dirigir en la dirección en la que deseemos los motores de corriente continua. Los motores de corriente continua que el robot posee, gobiernan las ruedas que representan el movimiento. Además de los motores de corriente continua la Skypic puede controlar 8 servos, la alimentación de los mismos puede ser independiente a la de la placa.

[1] Podemos apreciar en las Figuras 2.8 y 2.12 que las estructuras mecánicas son muy similares por no decir idénticas con la salvedad de la rueda loca de distinto diseño

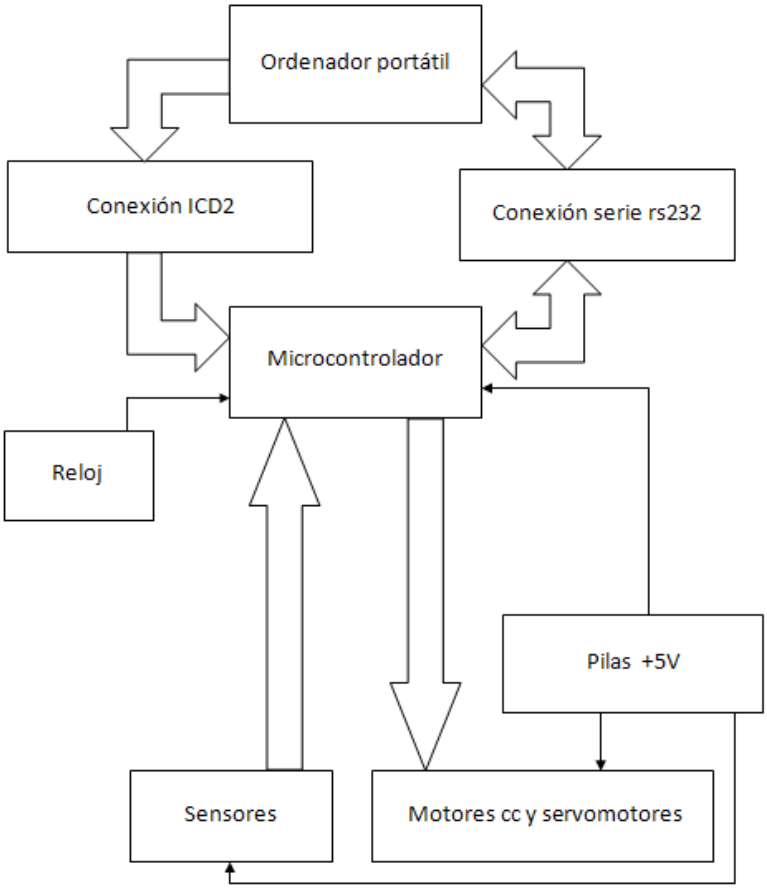


Figura 3.1: Esquema hardware del Skybot

El Skybot cuenta con un reloj que conecta con el micro para proporcionar una señal externa de sincronización. Dicha señal se utiliza en la coordinación de los distintos elementos conectados en los puertos del microcontrolador.

El robot consta de dos clavijas (RJ11) para conexión con el PC. Una de ellas para su programación y la otra para su control en tiempo real.

3.1.2. Arquitectura hardware UCLMin

Una vez familiarizado con esta arquitectura, pasamos a desarrollar la del UCLMin. Vamos a ver el esquema de la arquitectura diseñada:

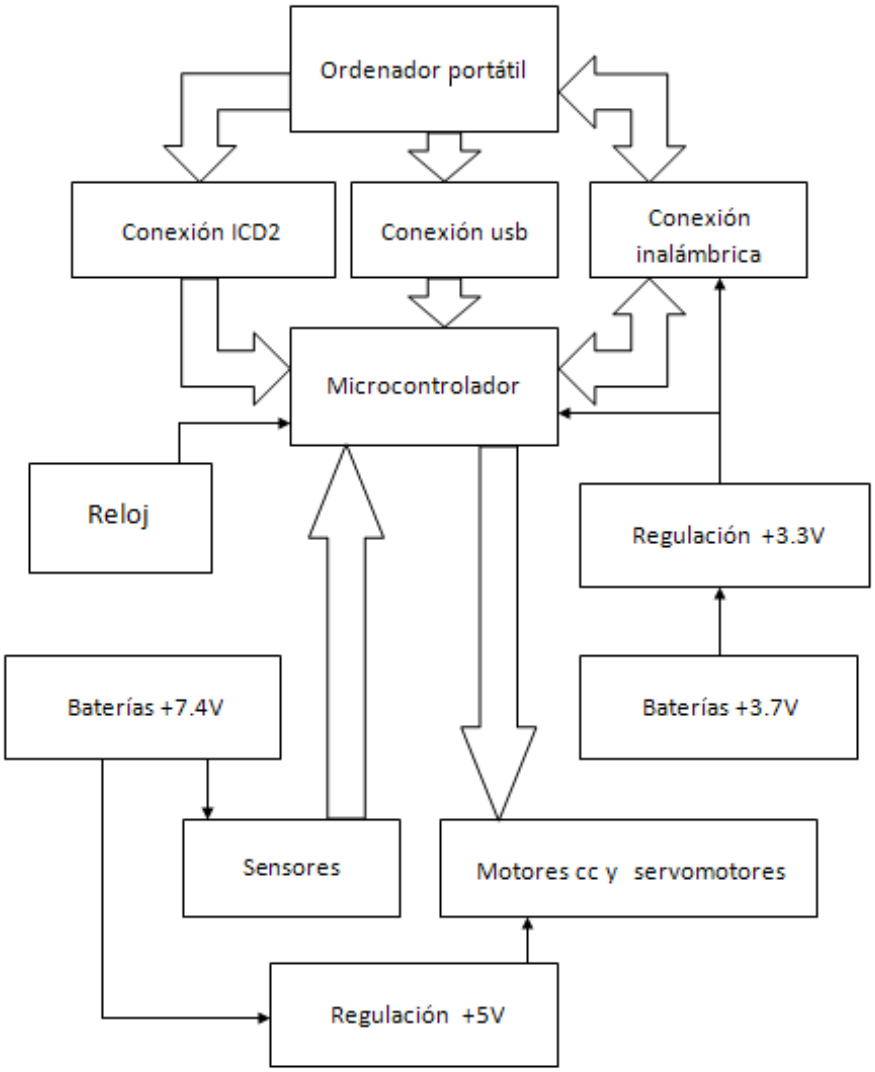


Figura 3.2: Esquema hardware del UCLMin

Aunque a simple vista la arquitectura del UCLMin es similar a la del Skybot, las diferencias son evidentes. Vemos que el micro puede conectarse con el PC a través de un bus USB (que sustituye al obsoleto RS-232) y por el módulo de comunicación inalámbrica. La inclusión de dicho módulo ha obligado a reducir el número de servomotores controlables de ocho a cuatro. Pero para cualquier aplicación sencilla a la que nuestro robot se va a enfrentar son más que suficientes. En cuanto al tema de la alimentación se ha considerado necesario diseñar dos circuitos de alimentación separados, uno a 3.7 que alimenta la electrónica y otro a 7.4 para la parte del robot que demanda más potencia (motores, sensores y servos.) de esa manera se evita

que los ruidos generados por los motores afecte a la electrónica pudiéndose producir un reseteo del microcontrolador o interferencias con el módulo de Radiofrecuencia que son las partes que trabajan a 3.7, el tema de la alimentación se abordará en detalle en la Sección 4.2.

El micro controla el bloque de los motores de corriente continua a través de la Sky293, éste bloque es el que se encarga de la locomoción del microrobot y no ha sido alterado con respecto al Skybot. Consta de dos motores independientes que pueden moverse hacia delante y hacia detrás y hacer girar el robot.

Desde el bloque del microcontrolador también gobernamos el bloque de los sensores. Éste consta de dos sensores infrarrojos (Figura 2.9) y de dos sensores de contacto o bumpers (Figura 2.10). Estos sensores al igual que los motores van conectados a la Sky293.

El bloque del microcontrolador ha sido mejorado y ampliado para poder soportar la comunicación inalámbrica y poder almacenar la información requerida por el protocolo de comunicaciones. Hemos pasado del PIC16F876A con el que trabaja el Skybot a uno de mayor potencia, puesto que permite conexión por usb con el PC y con suficiente memoria como para almacenar la pila de comunicaciones de un protocolo de una comunicación inalámbrica por Radiofrecuencia. Por ello se ha seleccionado el PIC18F2550 (Para más información se puede ver el Anexo C)

3.2. Arquitectura Software UCLMin

Desde el punto de vista del software a desarrollar en el proyecto nos hemos centrado principalmente en la programación del microcontrolador que gestiona:

- Las operaciones a bajo nivel con los actuadores del robot.
- Las operaciones con el módulo inalámbrico que permiten las comunicaciones con la unidad central de coordinación.

Para la programación del microcontrolador se ha empleado el lenguaje C y el compilador de C para PIC, MCC18 Compiler. Se han realizado programas específicos para el hardware montado.

Para explicar las interrelaciones entre los distintos componentes software que intervienen en la programación del UCLMin se va a utilizar el siguiente diagrama (Figura 3.3)

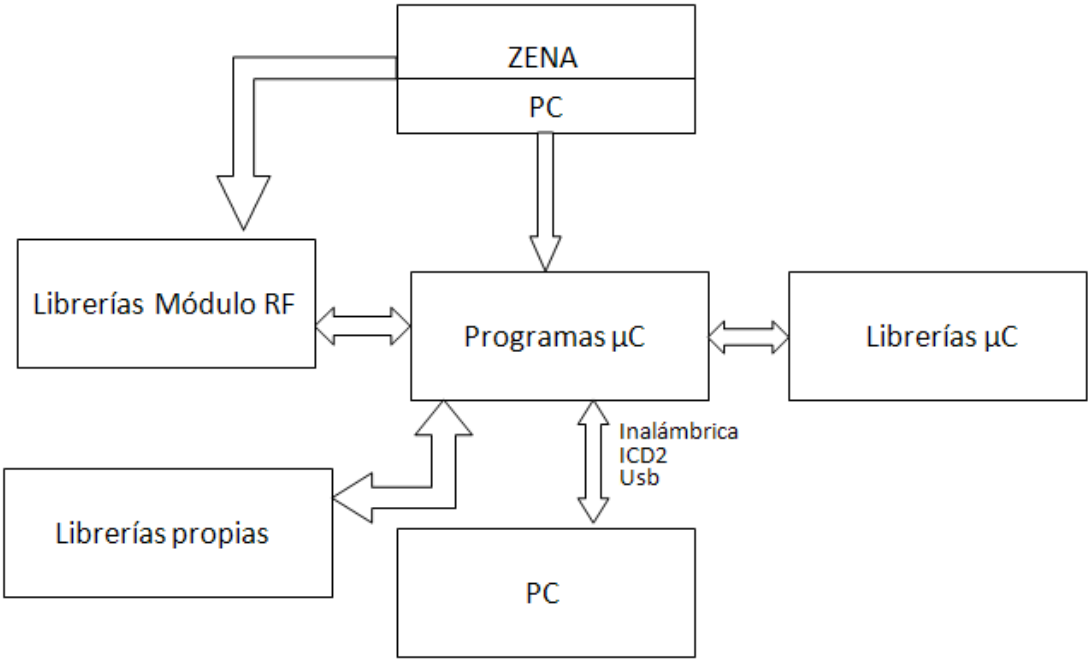


Figura 3.3: Arquitectura software del UCLMin

El diagrama está compuesto por 6 bloques o partes diferenciadas. Cuatro de ellas trabajan desde el Microcontrolador y las otras dos desde el PC. La disposición sería la siguiente:

- Microcontrolador
  - Módulo de Radiofrecuencia
  - Librerías propias (del UCLMin)
  - Librerías del Módulo
  - Programa μC
- PC
  - PC
  - ZENA

Vamos a comenzar por la parte del PC. Éste se comunica con el microcontrolador de tres maneras distintas por USB, por ICD2 y por comunicación inalámbrica. Cada uno de ellos tienen funciones distintas.

1. La comunicación inalámbrica se utiliza para recibir los datos que los sensores toman del entorno sacando por el Hyperterminal de Windows la información necesaria, para ello se utiliza la unidad de control coordinadora (placa PICDEM Z, Sección 4.5.3) que está conectada al PC por puerto serie RS232.
2. La comunicación ICD2 es la necesaria para la programación del microcontrolador. Es la comunicación que utiliza el programador de Microchip para grabar los programas en el microcontrolador.
3. Por último la comunicación USB, esta comunicación es muy común para enviar y recibir información por el puerto y consecuentemente para la programación del microcontrolador[2]

Aparte de las comunicaciones que hay entre el PC y el Microcontrolador, hay otro bloque que trabaja desde el PC, el ZENA. Éste se utiliza para las definiciones necesarias en la comunicación por Radiofrecuencia(Para más información sobre el programa ver Sección B.6). Estas definiciones están relacionadas con el tamaño de los mensajes a enviar, la colocación de los pulsadores, los led de la placa(si es que los ahí), los pines en los que se coloca la interrupción del micro, etc. La influencia de este bloque es exclusivamente sobre el módulo MRF24J40MA.

Si pasamos a la parte del microcontrolador debemos empezar por las librerías del microcontrolador. En estas librerías están albergadas a bajo nivel todas las directrices de registros instrucciones, bancos de memoria, etc. del microcontrolador. Esta librería se utiliza en cualquier programa que se haga para el UCLMin.

Se han programado para el robot una serie de librerías. Éstas albergan definiciones y funciones que relacionan el software con el hardware del robot. Al igual que antes se utilizan en todos los programas que se han hecho. Trabajar con ellas facilita mucho la programación, hace que ésta sea más ordenada y estructurada, ese es el objetivo que se perseguía creando estas librerías.

Otro bloque albergado en el microcontrolador es el del módulo de Radiofrecuencia. El módulo no puede almacenar información, no tiene soporte físico para ello, por lo que toda la programación que se necesite irá en el microcontrolador(librerías MiWi, MiWiDefs, etc. para más información E). La comunicación entre el módulo y el micro es bidireccional, aunque en nuestro caso sólo se ha utilizado el envío de datos desde el robot hasta el PC.

Todas las librerías hasta ahora mencionadas van almacenadas en el programa que se carga en el Microcontrolador.

[2]Para poder programar el microcontrolador a través del USB se necesita haber metido previamente el bootloader, para más información Sección 4.1.

4

REDISEÑO DEL SISTEMA

En este capítulo se abordan en detalle los elementos introducidos en la placa para darle una mayor autonomía, mayor facilidad de conexión con el PC tanto de manera inalámbrica como por cable y una breve descripción del sistema mecánico donde se va a probar.

4.1. Sustitución de la interfaz con el ordenador

La conexión con un PC de los robots de este tipo es el apartado menos desarrollado en las plataformas estudiadas. En general el sistema de comunicación más

utilizado es el puerto COM (puerto serie RS-232) debido a que su protocolo de comunicaciones está muy trabajado y es muy sencillo implementarlo sin necesidad de muchos conocimientos. El problema es que el puerto serie es un puerto que en la actualidad está obsoleto y prácticamente ha desaparecido de los nuevos ordenadores, tanto portátiles como de sobremesa.

Eso complica la conexión del robot con el PC ya que no es sencillo encontrar un puerto serie. Debido a esto decidimos sustituir la interfaz de conexión eliminando el puerto serie al que conecta el robot con el ordenador y el RJ11 desde el que conecta en la placa.

Un puerto serie o puerto serial es una interfaz de comunicaciones de datos digitales, frecuentemente utilizado por ordenadores y periféricos, en donde la información es transmitida bit a bit enviando un solo bit a la vez, en contraste con el puerto paralelo que envía varios bits simultáneamente.

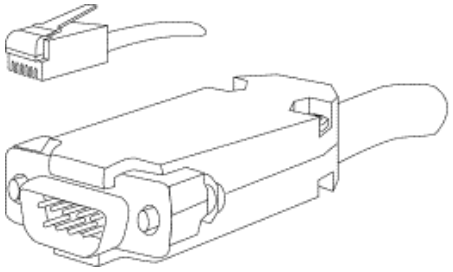


Figura 4.1: Conexión RJ11 a puerto serie (RS-232) del PC

El puerto serie es fácilmente conectable con otro puerto serie o con un RJ11. Esto es debido a que el puerto RJ11 es el utilizado para la conexión del programador del microcontrolador, precisamente esta es la razón por la que se sigue usando. Para poder sustituir la interfaz necesitamos que el microcontrolador previamente tenga cargado un programa llamado bootloader[1] en su memoria, esto nos permite volcar programas desde el ordenador a través de un usb convencional. El tema de la programación se abordará en detalle en el Capítulo 5.

A la hora de buscar interfaces más actuales para una comunicación serie, comprobamos que el estándar más extendido, que viene incluido en todos los ordenadores personales actuales es el Universal Serial Bus (bus universal en serie) o Conductor Universal en Serie (CUS), abreviado comúnmente USB, es un puerto que sirve para conectar periféricos a una computadora. Fue creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.

El diseño del USB tenía en mente eliminar la necesidad de adquirir tarjetas

[1] es equivalente al bootstrapping utilizado para arranque o proceso de inicio de cualquier ordenador, es el programa que generalmente arranca un sistema operativo como GRUB o Lilo.

separadas para poner en los puertos bus ISA o PCI, y mejorar las capacidades plug-and-play permitiendo a esos dispositivos ser conectados o desconectados al sistema sin necesidad de reiniciar.

Las señales del USB se transmiten en un cable de par trenzado con impedancia característica de $90\Omega \pm 15\,\%$, cuyos hilos se denominan D+ y D-.4 Estos, colectivamente, utilizan señalización diferencial en full dúplex[2] para combatir los efectos del ruido electromagnético en enlaces largos. D+ y D- suelen operar en conjunto y no son conexiones simples. Los niveles de transmisión de la señal varían de 0 a 0'3 V para bajos (ceros) y de 2'8 a 3'6 V para altos (unos) en las versiones 1.0 y 1.1, y en $\pm$ 400 mV en alta velocidad (2.0).

Figura 4.2: Trenzado de los cables D+ y D-

En las primeras versiones, los alambres de los cables no están conectados a masa, pero en el modo de alta velocidad se tiene una terminación de 45Ω a tierra o un diferencial de 90Ω para acoplar la impedancia del cable. Este puerto sólo admite la conexión de dispositivos de bajo consumo, es decir, que tengan un consumo máximo de 100 mA por cada puerto; sin embargo, en caso de que estuviese conectado un dispositivo que permite 4 puertos por cada salida USB (extensiones de máximo 4 puertos), entonces la energía del USB se asignará en unidades de 100 mA hasta un máximo de 500 mA por puerto.

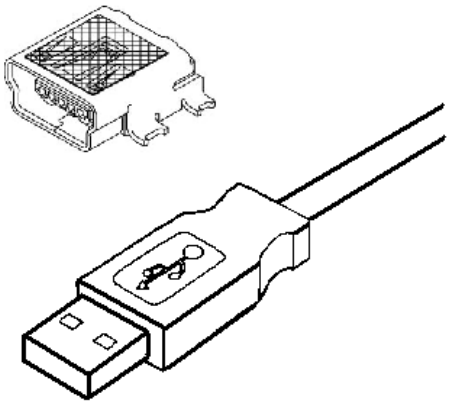


Figura 4.3: Conector de Usb convencional a mini usb

[2]La mayoría de los sistemas y redes de comunicaciones modernos funcionan en modo full dúplex permitiendo canales de envío y recepción simultáneos. Un ejemplo claro son los walkie talkies. A diferencia del puerto serie RS-232 que es half duplex

En nuestro caso hemos utilizado un microusb On The Go (sobre la marcha). Nos permite actuar como servidor o como dispositivo[3]. Incluso después de que el cable esté conectado y las unidades se estén comunicando, las 2 unidades pueden "cambiar de papel" bajo el control de un programa. Es un conector de 5 pines. Hay distintos tipos, el Micro-A, el Micro-B y el Micro-AB. La diferencia a parte de por la forma de sus conectores es por el pin 4. Este pin representa el ID, es decir si el dispositivo trabaja como maestro o como esclavo. En la Tabla 4.1 podemos ver donde se conecta este pin en función del tipo de conector.

Pin	Nombre	Color	Descripción
1	VCC	Rojo	+5V
2	D-	Blanco	Data -
3	D+	Verde	Data +
4	ID	Ninguno	Permite la distinción de Micro-A y Micro-B. ▪ Tipo A: conectado a tierra ▪ Tipo B: no conectado
5	GND	Negro	Señal de tierra

Tabla 4.1: Descripción de los pines del conector usb

La ventaja que nos proporciona el usb con respecto a otro conector en la actualidad es que cualquier portátil u ordenador de sobremesa viene equipado con puertos serie, por que sería más sencillo poder hacer pruebas con el robot en cualquier lugar sin necesidad de un programador[4].

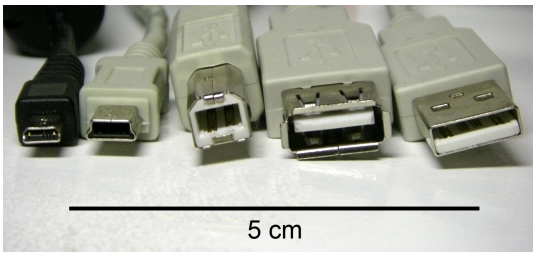


Figura 4.4: Distintos conectores USB

En nuestro caso se ha utilizado un USB miniAB. La diferencia con los tipos descritos en la Tabla 4.1 es que tenemos las dos opciones. Si el pin 4 se conecta a tierra trabaja como tipo A y si no se conecta trabaja como tipo B. En este caso no se ha conectado por lo que trabajamos como un tipo B.

[3]Trabajará como dispositivo cuando se conecte a un ordenador y como servidor cuando se le conecte un teclado o un ratón. En nuestro caso sólo actuará como dispositivo

[4]si bien es cierto que para poder cargar los programas en el microcontrolador a través del usb sería necesario el programador una primera vez para cargar en memoria el bootloader, como ya se ha mencionado antes

4.2. Cambio en el sistema de alimentación

La alimentación de un robot es la que nos limita su autonomía. En la actualidad la gran mayoría de los robots comerciales a esta escala trabajan con pilas eléctricas AA.

Una pila eléctrica es un dispositivo que convierte energía química en energía eléctrica por un proceso químico transitorio, tras de lo cual cesa su actividad y han de renovarse sus elementos constituyentes, puesto que sus características resultan alteradas durante el mismo. Se trata de un generador primario. Esta energía resulta accesible mediante dos terminales que tiene la pila, llamados polos, electrodos o bornes. Uno de ellos es el polo negativo o ánodo y el otro es el polo positivo o cátodo.

Figura 4.5: Descripción de una pila

Las pilas tienen dos características fundamentales:

- **Voltaje** El voltaje, tensión o diferencia de potencial que produce un elemento electroquímico viene determinado completamente por la naturaleza de las sustancias de los electrodos y del electrolito, así como por su concentración. La conexión de elementos en serie permite multiplicar esta tensión básica cuanto se quiera.

  Una vez fijada la tensión, la ley de Ohm determina la corriente que circulará por la carga y consecuentemente el trabajo que podrá realizarse.

- **Capacidad total** La capacidad total de una pila se mide en amperios x hora (A·h); es el número máximo de amperios que el elemento puede suministrar en una hora. Es un valor que no suele conocerse, depende de la intensidad solicitada y la temperatura. Cuando se extrae una gran corriente de manera continuada, la pila entrega menos potencia total que si la carga es más suave. También en esto las pilas alcalinas son mejores.

El problema es que para hacer funcionar de manera simultánea dos motores de corriente continua, los sensores, la electrónica incluyendo el módulo inalámbrico y

los servos se demanda mucha corriente. La demanda de corriente implica un mayor número de pilas y un mayor número de pilas aumenta el tamaño del robot, por lo que resulta conveniente buscar una solución que sea más compacta permitiendo el mismo suministro de potencia.

La solución de partida son las baterías, son más caras que las pilas normales pero el ciclo de vida es mucho mayor. Permiten una vez se han descargado, volver a cargarse usando procedimientos electroquímicos. Este ciclo puede repetirse un determinado número de veces. Es un generador secundario ya que para suministrar energía previamente se le ha suministrado a él en el proceso de carga.

De entre todos los tipos de baterías que existen en el mercado actual, las baterías de Polímero de Litio (LiPo) son las de mayores prestaciones en relación a su peso y tamaño ya que el proceso químico en el que se basa es mejor. Estas baterías tienen una densidad de energía de entre 5 y 12 veces las de Ni-Cd ó Ni-MH, a igualdad de peso. A igualdad de capacidad son, típicamente, cuatro veces más ligeras que las de Ni-Cd.

Cada batería o elemento LiPo tiene un voltaje nominal, cargado, de 3.7V. Estos elementos tienen unas condiciones de carga y descarga bastante rigurosas:

- Una batería LiPo[5] nunca debe ser descargado por debajo de los 3 V por elemento
- Una batería LiPo nunca debe ser cargada por encima de los 4.3 V por elemento.

Las baterías de Li-Po, además de especificar su capacidad (en mAh) y su número de elementos (voltaje= nx3.7, donde n es el número de elementos), indican la corriente máxima que son capaces de suministrar sin sufrir daños, en múltiplos de C (capacidad). Así, una batería de 800 mAh y 15 C es capaz de suministrar una corriente máxima de 15x0.8A= 12A. lógicamente, una batería dimensionada para proporcionar mayor corriente tiene también mayor peso y volumen.

Debido a su reducido tamaño y peso y altas prestaciones en comparación con otras baterías y las pilas convencionales nos decidimos a usarlas.

Una vez tomada la decisión debemos decidir qué cantidad de elementos nos van a hacer falta, debemos tener en cuenta los consumos y las tensiones requeridas por la electrónica y por la parte de potencia[6]. Toda la electrónica debería alimentarse a más de 3.3 V, puesto que las celdas son de 3.7 con una sería más que suficiente. Para la parte de potencia se necesitan al menos 5 V, por lo que automáticamente hay que pasar a 2 celdas. Después de las prácticas realizadas se ha llegado a los valores de corriente necesarias. Las dos alimentaciones son:

[5]Batería LiPo: representa la unión de 1 o más elementos LiPo consiguiendo mayores tensiones si se colocan en serie o mayores capacidades si se colocan en paralelo.

[6]entiéndase como potencia en un microrobot la parte de motores, sensores y servomotores.

- ZIPPY 850mAh 20C single cell. Esta batería es de una sola celda, es decir que trabaja a 3,7 V. Sus características principales son:
  - Carga: 850 mAh
  - Celdas, Voltaje: 1, 3.7 V
  - Descarga máxima: 20C[7]
  - Peso: 20.2 g
  - Dimensiones: 60x31x7 mm (ver Figura 4.6)

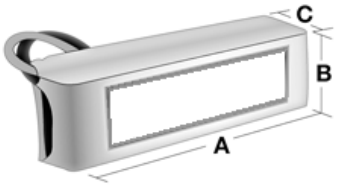


Figura 4.6: Dimensiones de las baterías

Ésta alimenta a toda la electrónica del UCLMin (microcontrolador y módulo inalámbrico). La electrónica trabaja a 3.3 V por lo que ha sido necesario colocar un regulador de tensión. El elegido ha sido el TPS79533DCQ de Texas Instruments cuya hoja de características puede consultarse en el Anexo F.

- ZIPPY Flightmax 1600mAh 2S1P 20C
  - Carga: 1600 mAh
  - Celdas, Voltaje: 2, 7.4 V
  - Descarga máxima: 20C constante, 30C de pico
  - Peso: 99 g
  - Dimensiones: 100x34x14mm (ver Figura 4.6)

  Esta batería alimenta toda la parte de la potencia. Los servos alimentados directamente a 7.4 V. Y utilizando un regulador de tensión a 5 V para motores y sensores, en concreto el MAX603 de MAXIM cuya hoja de características puede consultarse en el Anexo F.

[7]Para calcular la descarga máxima conociendo la carga y su capacidad (C de la batería, si una batería tiene 20 C podrá descargar 20 veces su capacidad) simplemente bastaría con multiplicar por lo que esta batería alcanzaría una descarga máxima de: 850x20=17000 mA = 17 A

4.3. Módulo inalámbrico

4.3.1. Elección del protocolo

La inclusión del módulo inalámbrico ha sido la decisión más complicada que hemos tomado. No sólo hay que elegir qué módulo es el que vamos a utilizar, además debemos tener en cuenta los protocolos que cada uno de estos módulos soportan y una vez vistas las posibilidades decidir.

Hay gran cantidad de tipos de comunicación inalámbrica. Nosotros nos vamos a centrar en cuatro principalmente:

- ZigBee
- WiFi
- Bluetooth
- GPRS

De entre los cuatro tipos de comunicación estudiados, nos decidimos por utilizar los protocolos ZigBee. Éstos están definidos para su uso en aplicaciones embebidas con requerimientos muy bajos de transmisión de datos y consumo energético. Se pretende su uso en aplicaciones de propósito general con características autoorganizativas y bajo costo (redes en malla, en concreto). Puede utilizarse para realizar control industrial, albergar sensores empotrados, recolectar datos médicos, ejercer labores de detección de humo o intrusos o domótica. La red en su conjunto utilizará una cantidad muy pequeña de energía de forma que cada dispositivo individual pueda tener una autonomía de hasta 5 años antes de necesitar un recambio en su sistema de alimentación.(Para más información, véase Anexo D).

Hay tres protocolos ZigBee. El ZigBee propiamente dicho, y dos más desarrollados a partir de la pila de éste que son el MiWi y el MiWi P2P. Estos dos protocolos han sido desarrollados por Microchip, son de uso libre y lo único que se aconseja es trabajar con módulos de la propia marca, debido a que estos protocolos adaptan el ZigBee a las características de sus módulos.

Una vez estudiados estos protocolos, nos decantamos por desechar el MiWi P2P porque si bien para la aplicación que en el presente proyecto se iba a desarrollar era más que suficiente, debíamos tener en cuenta el posible desarrollo futuro. Este protocolo nos permite la comunicación punto a punto[8] que para comunicar el robot con el ordenador y viceversa era la opción mejor y más rápida. Pero como ya se ha comentado debíamos pensar en el futuro y este protocolo no nos es útil para

[8]P2P, o peer to peer, es la arquitectura de redes en la que algunos programas basan su estructura en la que los participantes comparten recursos para comunicarse directamente entre ellos.

el desarrollo de una red de más de dos dispositivos (que es al fin y al cabo una comunicación P2P).

Debíamos elegir el protocolo entre las dos opciones que nos quedaban. Estudiando las diferencias vimos que eran mínimas y que para nuestros intereses eran el mismo protocolo. Las diferencias entre ellos son dos fundamentalmente:

- Su capacidad de infraestructura es menor ya que ZigBee nos permite tener hasta 64000 nodos mientras que el MiWi lo tiene limitado a 1024.
- Estamos obligados a utilizar componentes de Microchip para el desarrollo de aplicaciones con el MiWi.

Puesto que cumplimos los dos requisitos, decidimos elegir el de menor consumo de recursos. Por lo que nos decantamos por el MiWi.

4.3.2. MiWi

MiWi es un protocolo inalámbrico, para redes de área personal basado en el estándar IEEE 802.15.4 que engloba las redes de área personal con tasas bajas de transmisión de datos (low-rate wireless personal area network, LR-WPAN).

Está dirigido a dispositivos y redes de bajo coste, que no necesitan una alta transferencia de datos (250Kbit/s), a corta distancia (100 metros sin obstáculos), y con un consumo energético mínimo.

Está dentro del espectro de los 2,4Ghz a través de su transceptor MRF4J4 y es compatible con todos los dispositivos IEEE 802.15.4, como el ZigBee.

Es capaz de formar redes punto a punto (MiWi P2P), de estrella o de malla, según IEEE 802.15.4. Puede tener 8 coordinadores como máximo por red, y cada uno de éstos 127 hijos, haciendo un total de 1024 de nodos por red.

El robot debe ser lo más general posible y eso nos obliga en cierto modo a seleccionar el módulo de Microchip (Figura 4.7). Este módulo se programa de manera relativamente sencilla con una serie de comandos de la plataforma. Pero estos comandos pueden usarse en cualquier programador de C. Sin embargo, el módulo XBee necesita de un programa específico suministrado por la misma empresa de los módulos. Que el módulo implique tener que aprender un programa nuevo supone que cualquier persona con unos conocimientos mínimos en robótica y programación no podría trabajar con él. Como el objetivo es todo lo contrario nos decantamos por el MRF24J40MA.

J2

Microcontroller	Signal	Pin		Pin	Signal	Microcontroller
RB3		12		11		RB2
RB1		10		9	SCK	RC3
RC4	MISO	8		7	MOSI	RC5
RC0	$\overline{CS}$	6		5	INT	RB0
RC1	WAKE	4		3	$\overline{RESET}$	RC2
	GND	2		1	+3.3V	

(Top view on Motherboard)

Figura 4.7: Módulo de radiofrecuencia MRF24J40ma y conexión de sus pines

A continuación se va a hacer una breve descripción de los pines más importantes del módulo:

1. VCC: El módulo se alimenta a 3.3V.
2. GND: Conectado a tierra en la placa
3. RESET: Es una entrada digital al módulo. Es bajo activa y representa el Reset del Hardware. Esta entrada se utiliza al comienzo del programa, cuando el dispositivo entra en una red.
4. WAKE: Es una entrada digital al módulo. Es un disparador. Su existencia permite que en reposo el módulo consuma 3 μA. Cuando se activa el módulo está a la espera de un paquete que podrá ser recibido o enviado en función de la dirección de la comunicación.
5. INT: Es una salida digital que actúa sobre la interrupción (Para más información sobre interrupciones acudir a la Sección 4.4) principal del micro, cuando esta se activa el micro activa la comunicación SPI (Para más información acudir a la sección 4.4.1) que tiene con el módulo.

6. CS: Es una entrada digital bajo activa, se utiliza para permitir la conexión del SPI. Es la primera señal que se activa, cuando lo hace el módulo empieza a tomar pulsos de reloj por el SCK y a enviar (por SDO) o recibir (por SDI) información.
7. MOSI/SDI: Es una entrada digital de datos del SPI al módulo MRF24J40. Recibe el paquete de datos que el micro quiere enviar a la unidad central de coordinación en nuestro caso.
8. MISO/SDO: Es una salida digital de datos del SPI, su función es similar a la del SDI pero en sentido opuesto, recibe los datos de otro dispositivo y manda la información al micro.
9. SCK: Es una entrada digital, representa el pulso de reloj para la comunicación SPI
10. NC: No conectado, no tiene utilidad. Se conecta a tierra
11. NC: No conectado, no tiene utilidad. Se conecta a tierra
12. NC: No conectado, no tiene utilidad. Se conecta a tierra

4.4. Microcontrolador

Un microcontrolador es un circuito integrado o chip que incluye en su interior las tres unidades funcionales de un PC:

- unidad central de procesamiento
- memoria
- unidades de E/S (entrada/salida).

Son diseñados para reducir el costo económico y el consumo de energía de un sistema en particular. Por eso el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

En nuestro caso el micro que debíamos utilizar tenía que cumplir unos requisitos mínimos. Éstos requisitos no son más que las prestaciones de nuestro robot. Haciendo una comparativa entre ellos (Para más información véase Anexo C) se ha llegado a la conclusión de que el microcontrolador más adecuado de entre los estudiados es el PIC18F2550, es un microcontrolador de Microchip, de gama alta dentro de los microcontroladores de 8 bits. con las siguientes características:

Como se puede observar en la Tabla 4.2 nos proporciona los recursos necesarios para nuestro robot. Nos permite la comunicación con el PC tanto por USB, como

por Radiofrecuencia utilizando el MRF24J40MA, ya que tiene memoria suficiente como para poder hacer correr la pila de comunicaciones[9]

Tipo de memoria programable	Flash
Tamaño de memoria programable(KB)	32
Velocidad CPU	12 MIPS(Millones de inst. por segundo)
Bytes de RAM	2,048
Datos EEPROM (bytes)	256
Comunicación digital con periféricos	1-A/E/USART, 1-MSSP(SPI/I2C)
Captura/Comparación/PWM Periféricos	2 CCP
Temporizadores	1 x 8-bit, 3 x 16-bit
Conversor analógico-digital	10 ch, 10-bit
Comparadores	2
USB (canales, velocidad, versión)	1, Full Speed, USB 2.0
Rango de voltaje de operaciones (V)	2 to 5.5
Número de pines	28
Frecuencia de operación (MHz)	48
Número de entradas y salidas	16

Tabla 4.2: Características del PIC18F2550

4.4.1. Comunicación SPI

Una de las características más importantes del micro elegido es que posee un módulo de comunicación por SPI.

El Bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interface de periféricos serie o bus SPI es un estándar para controlar casi cualquier electrónica digital que acepte un flujo de bits serie regulado por un reloj.

Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Muchos sistemas digitales tienen periféricos que necesitan existir pero no ser rápidos. La ventaja de un bus serie es que minimiza el número de conductores, pines y el tamaño del circuito integrado. Esto reduce el coste de fabricar montar y probar la electrónica.

Un bus de periféricos serie es la opción más flexible cuando muchos tipos diferentes de periféricos serie están presentes. Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales. Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube otros

[9]Ésta fue la razón por la que no se pudo utilizar el PIC16F876A, que era el que utilizaba el Skybot.

cuando el reloj baja. Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj. Algunos dispositivos tienen dos relojes. Uno para capturar o mostrar los datos y el otro para el dispositivo interno.

En nuestro caso la conexión es entre un master (el micro) y un esclavo (el MRF24J40MA) únicamente. El diagrama de conexión entre ellos es el de la Figura 4.8

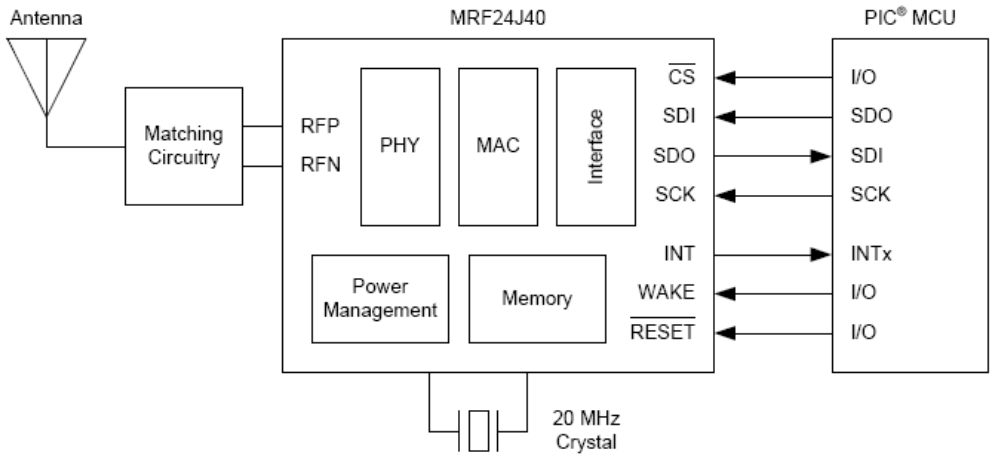


Figura 4.8: Diagrama de conexión del SPI

Por último, las interrupciones constituyen quizá el mecanismo más importante para la conexión del microcontrolador con el mundo exterior, sincronizando la ejecución de programas con acontecimientos externos.

El funcionamiento de las interrupciones es similar al de las subrutinas de las cuales se diferencian principalmente en los procedimientos que las ponen en marcha. Así como las subrutinas se ejecutan cada vez que en el programa aparece una instrucción de llamada, las interrupciones se ponen en marcha al aparecer en cualquier instante un evento externo al programa, es decir por un mecanismo hardware.

Hay distintos tipos de interrupciones, nosotros utilizamos dos:

- La INT0 que se utiliza por desbordamiento del temporizador TMR0. Para activar la interrupción del TMR0, deben estar activadas las interrupciones en el micro y para ello el bit correspondiente en el debe de estar a 1; bajo estas condiciones cuando el temporizador TMR0 se desborda al pasar de FFh a 00h, se activa el flag TOIF del registro INTCON.

Si no se carga de nuevo TMR0 cuando se desborda, éste sigue contando desde 00h hasta FFh. Cuando se carga el registro TMR0 con un valor XXh, éste cuenta FFh-XXh impulsos.

- En nuestro caso la activación del pin RB2/INT2, en este pin es donde está colocada la interrupción del módulo inalámbrico.

4.5. Descripción de la placa final

4.5.1. Placa UCLMin

La placa UCLMin (Figura 4.9, o en el Anexo III, donde se puede ver el esquemático completo de la placa) es el cerebro de nuestro robot.

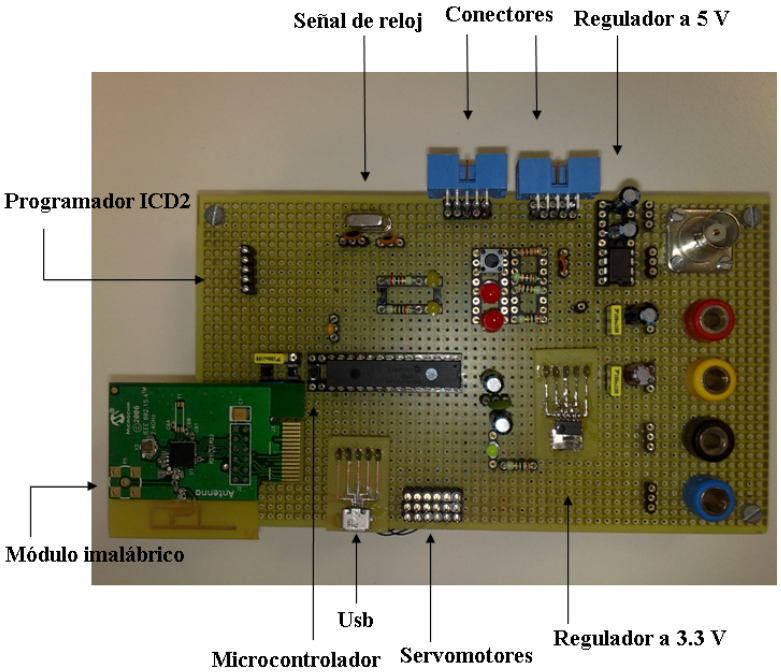


Figura 4.9: Placa de pruebas UCLMin v 1.0

En ella se encuentra todo lo necesario para la comunicación del robot con el PC. Esta comunicación puede llevarse a cabo de tres maneras distintas:

- Conector de 5 pines compatible con ICD2 de Microchip.
- Conector mini-USB.

- Comunicación inalámbrica, a través del módulo de Radiofrecuencia MRF24J40MA (Figura 4.7), de Microchip.

Además de las conexiones con el PC, la placa UCLMin tiene otros dos tipos de conectores:

- Dos conectores de 10 pines(Figura 4.10) acodados para la conexión con la Sky293, para la conexión del micro con sensores y motores. La disposición de los pines de estos puertos es la que se puede ver en la Tabla 4.3:

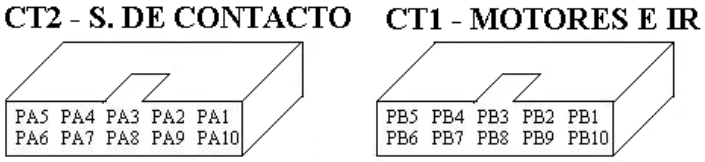


Figura 4.10: Puertos de expansión

Puerto Sensores de contacto	**Puerto Motores E IR**
PA1: Entrada del bumper izquierdo	**PB1:** Sentido de giro Motor 1
PA2: NC	**PB2:** Sentido de giro Motor 2
PA3: NC	**PB3:** Estado sensor 2
PA4: NC	**PB4:** NC
PA5: Vcc	**PB5:** Vcc
PA6: GND	**PB6:** GND
PA7: NC	**PB7:** NC
PA8: NC	**PB8:** Motor 2, ON/OFF
PA9: Entrada del bumper derecho	**PB9:** Motor 1, ON/OFF
PA10: NC	**PB10:** Estado sensor 1

Tabla 4.3: Puertos de expansión de UCLMin

- Cuatro conexiones directas de hasta cuatro servos tipo Futaba, Hitec, etc.

A cada uno de los sensores que la UCLMin trata se le ha colocado un led. De esta manera se puede comprobar cuando un sensor se activa o se desactiva o si tiene algún problema de manera más visual.

La alimentación de la placa se ha llevado a cabo por dos vías distintas. De esta manera se aíslan posibles ruidos que se generasen en los motores y sobre todo en los servomotores que pudieran afectar a la electrónica de control. Para estas alimentaciones como ya se ha explicado en este mismo capítulo (Sección 4.2) se han utilizado baterías LiPo.

A la versión final de la placa Figura (4.11), se le ha dado un tamaño superior al óptimo, por la sencilla razón de poder acoplarla a la Sky293 (figura 4.12) y de esta manera también poder aprovechar la estructura mecánica sobre la que ésta va montada.

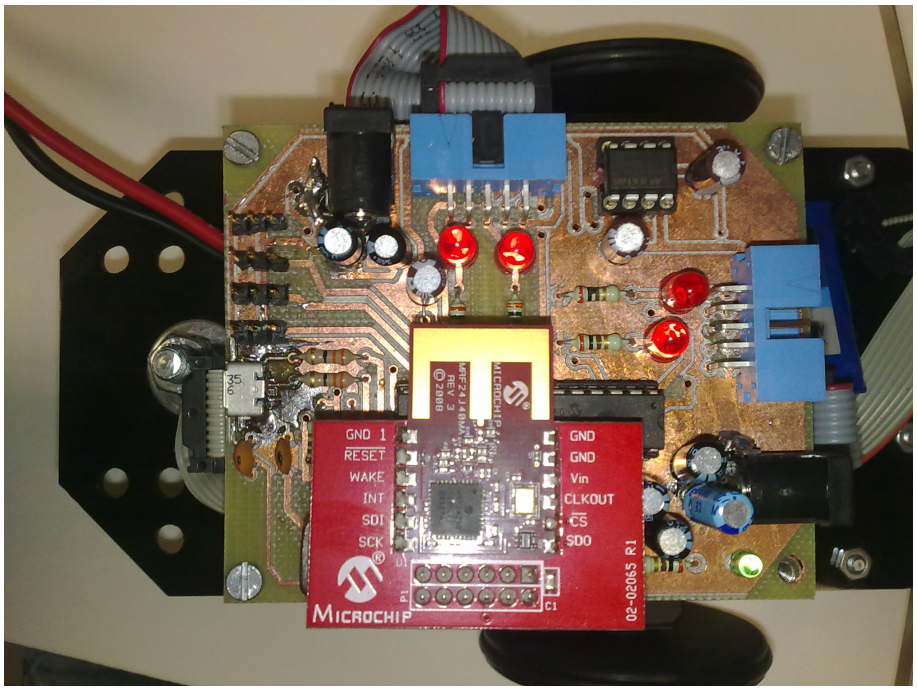


Figura 4.11: Placa UCLMin versión final ya montada en la estructura del Skybot

4.5.2. Placa auxiliar, Sky293

La placa Sky293 proporciona la posibilidad de controlar motores, leer sensores y activar/desactivar un relé. Está diseñada para adaptarse perfectamente a la tarjeta SkyPic (Figura 2.8), aunque se puede usar con cualquier otro microcontrolador.

Sus características principales son las siguientes:

- Posibilidad de control de dos motores de corriente continua o uno paso a paso. Se usa el chip L293B
- Capacidad para leer cuatro sensores de infrarrojos (CNY70), pudiendo ser optoacopladores
- Activar/Desactivar un relé [10]. La Sky293 viene preparada para usarlo, pero no

[10] es un dispositivo electromecánico, que funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

está en uso.

- Disponibilidad de 6 entradas digitales de propósito general con la posibilidad de usarlas con resistencias de pull-up que se pueden activar con sus respectivos jumpers
- Disponibilidad de seis entradas analógicas compartidas con las entradas digitales y el relé
- Entrada de reloj externo para el Timer0
- Conexión de los elementos externos a través de clemas
- Alimentación independiente para los motores, o bien usando la misma que la electrónica
- Conexión directa a una tarjeta microcontroladora (véase Skypic, UCLMin, etc.), a través del puerto A y B

En la Figura 4.12 podemos ver la disposición de los conectores en la placa.

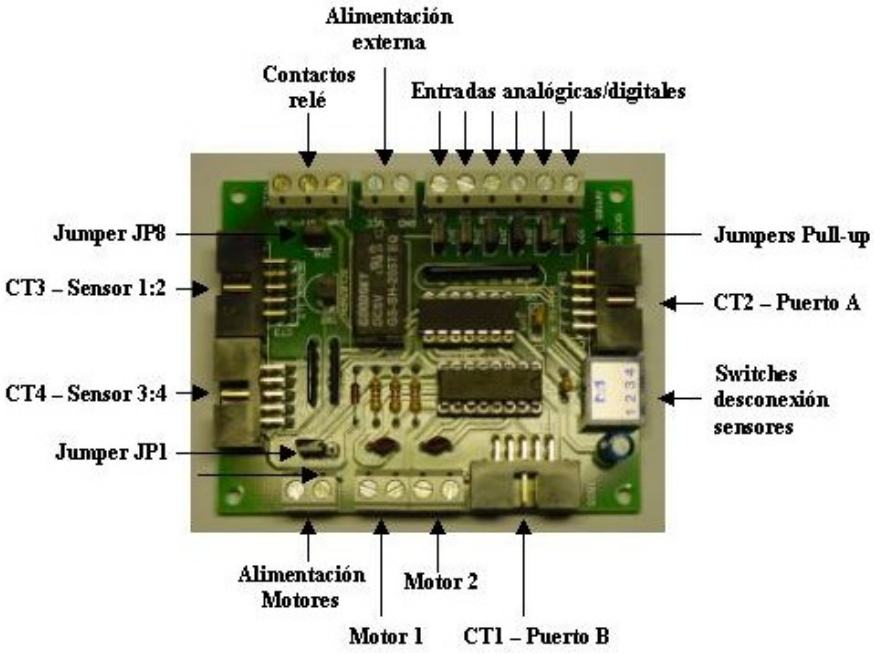


Figura 4.12: Placa Sky293

Además la Sky293 tiene dos tipos de puertos:

- Puertos de expansión, denominados A y B, son los puertos que vamos a conectar a nuestra UCLMin(El conector tiene la misma disposición que en la Figura 4.10).

Puerto A	**Puerto B**
PA1: Entrada analógica/digital 1	**PB1:** Sentido de giro Motor 1
PA2: Entrada analógica/digital 3	**PB2:** Sentido de giro Motor 2
PA3: Entrada analógica/digital 4 ó Relé	**PB3:** Estado sensor 2
PA4: NC	**PB4:** Estado sensor 4
PA5: Vcc	**PB5:** Vcc
PA6: GND	**PB6:** GND
PA7: NC	**PB7:** Estado sensor 3
PA8: Input/Output ó TOCK	**PB8:** Motor 2, ON/OFF
PA9: Entrada analógica/digital 2	**PB9:** Motor 1, ON/OFF
PA10: Entrada analógica/digital 0	**PB10:** Estado sensor 1

Tabla 4.4: Puertos de expansión

- Puertos de sensores, denominados Sensor 1:2 y Sensor 3:4, donde se conectan los sensores infrarrojos. En nuestro caso únicamente utilizaremos el puerto Sensor 1:2 (el conector tiene la misma disposición que en la Figura 4.10).

Sensor 1:2	**Sensor 3:4**
P1: Pin E (Sensor 1)	**P1:** Pin E (Sensor 3)
P2: Vcc - Pin C,A (Sensor 1)	**P2:** Vcc - Pin C,A (Sensor 3)
P3: Pin K (Sensor 1)	**P3:** Pin K (Sensor 3)
P4: Pin E (Sensor 2)	**P4:** Pin E (Sensor 4)
P5: Vcc - Pin C,A (Sensor 2)	**P5:** Vcc - Pin C,A (Sensor 2)
P6: Pin K (Sensor 2)	**P6:** Pin K (Sensor 4)
P7: Vcc - Pin C,A (Sensor 2)	**P7:** Vcc - Pin C,A (Sensor 4)
P8: Pin E (Sensor 2)	**P8:** Pin E (Sensor 4)
P9: Pin K (Sensor 1)	**P9:** Pin K (Sensor 3)
P10: Vcc - Pin C,A (Sensor 1)	**P10:** Vcc - Pin C,A (Sensor 3)

Tabla 4.5: Puertos de sensores

La Sky293 se alimenta con 5v, a través de los puertos de expansión (puerto A o Puerto B) o mediante la clema. La alimentación de los motores se puede hacer de forma independiente a través de una clema. En ese caso hay que poner el jumper JP1 en la posición 1-2.

Podemos ver el esquemático de la placa en el Anexo A

4.5.3. Placa auxiliar, PICDEM Z

La placa PICDEM Z es la que actuará como coordinador en la red inalámbrica MiWi. Esta placa es propiedad de Microchip y se puede encontrar toda la documentación en su página web `http://www.microchip.com`. Esta placa pertenece a un kit de desarrollo para comunicación inalámbrica. Para este cometido cuenta con la posibilidad de trabajar con tres protocolos distintos:

- ZiGBee
- MiWi
- MiWi P2P

El kit de desarrollo está compuesto de dos placas iguales a esta, con dos módulos de radiofrecuencia (MRF24J40MA) y un analizador de frecuencias (ZENA).

4.6. Sistema mecánico

La estructura(Figura 4.13) mecánica que se va a utilizara para las pruebas iniciales del robot es bastante sencilla pero consistente y muy funcional. La estructura mecánica está compuesta por 4 piezas de metacrilato, dos servos Futaba 3003 modificados para funcionar como motores de corriente continua y una rueda loca. Las piezas de la estructura se unen entre ellas con pegamento y los motores se sujetan mediante tornillos normales de métrica 4.

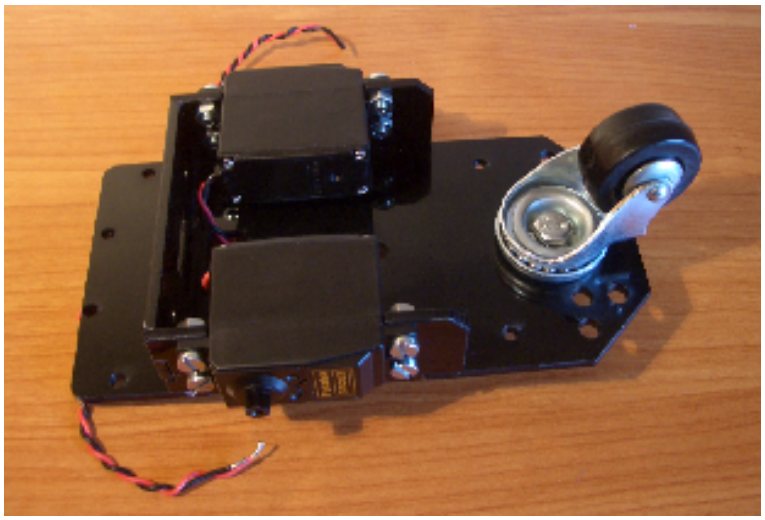


Figura 4.13: Estructura mecánica para UCLMin

5

PROGRAMACIÓN

En este capítulo se tratarán los programas realizados para las pruebas de las distintas partes del robot y por último, el programa de comunicación inalámbrica que se ha llevado a cabo.

Se intentará partir de los programas más sencillos, para ir avanzando de manera progresiva, hasta llegar a poder controlar el robot de manera autónoma. Después se explicará en detalle el funcionamiento del programa que nos permite comunicar la información de los sensores al robot por comunicación inalámbrica.

5.1. Configuración del PIC18F2550

Un aspecto importante a tratar antes de pasar a la programación son las conexiones realizadas al pic en sus distintos puertos, ya que son parte fundamental de la configuración del microcontrolador. Por otro lado resulta muy conveniente realizar

esta puntualización antes de pasar a comentar el código para una mejor compresión de los programas.

Dichas configuraciones se realizarán mediante tres puertos de 8 bits (A, B y C). Cada uno de los bits corresponde con una entrada/salida digital de nuestro microcontrolador. Cada bit puede ser configurado a 1 o a 0 dependiendo de la función que tenga dicho pin. El convenio elegido es el siguiente: si se configura a 1 el bit se trata como una entrada y si se pone a 0 se trata como una salida. Dicho esto se procede a la explicación de los bits de los distintos puertos:

Bit	I/O	Función en el robot
A0	O	Dirección del motor 1
A1	O	Dirección del motor 2
A2	I	Sensor CNY
A3	O	Enable motor 2
A4	O	Enable motor 1
A5	I	Sensor CNY
A6	I	Oscilador
A7	I	Oscilador

Tabla 5.1: Bits de configuración del puerto A

Con estos valores podemos llegar a la definición del puerto, ya que sería 11100100 (El primero de ellos es el MSB o most significant bit, es decir, el bit más significativo y representa el pin A7. El último es el LSB o less significant bit, es decir, el bit más significativo que corresponde al pin A0). Es común definir el puerto de manera hexadecimal, por lo que el puerto A queda definido como 0xE4 (Para más información de conversión binario-hexadecimal acudir a la Sección G.2).

Bit	I/O	Función en el robot
B0	I	SDI
B1	O	SCK
B2	I	INTx
B3	O	CS
B4	O	RESET
B5	O	WAKE
B6	I	Sensor de contacto
B7	I	Sensor de contacto

Tabla 5.2: Bits de configuración del puerto B

Por lo que el puerto B quedaría 11000101 que en hexadecimal 0xC5.

Bit	I/O	Función en el robot
C0	O	Servo
C1	O	Servo
C2	O	Servo
C3	No existe	Vusb
C4	O	D- (datos del USB)
C5	O	D+ (datos del USB)
C6	O	Servo
C7	O	SDO

Tabla 5.3: Bits de configuración del puerto C

El puerto C quedará configurado como 00000000 que en hexadecimal utilizando la tabla de conversión sería 0x00. El pin C3 no se puede utilizar como entrada o salida, ya que está configurado para ser una salida regulada de tensión a 3.3 V.

5.2. Librerías de uso común

Antes de pasar a describir los programas desarrollados, es necesario describir una serie de librerías que serán utilizadas por contener definiciones propias del micro o funciones básicas creadas para facilitar el uso del mismo. Estos ficheros incluyen:

- Constantes y funciones básicas de configuración para el microcontrolador que se va a utilizar, como se ha mencionado en el capítulo anterior (Sección 4.4).
- Librerías de usuario conteniendo funciones básicas utilizadas en el desarrollo de los programas posteriores.
- Ficheros de cabecera con definiciones de constantes que faciliten la configuración del robot.

libreria_UCLMin.c

```
1
2   /*****************************************************************************/
3   /* libreria_UCLMin.c      Octubre-2009                                       */
4   /*---------------------------------------------------------------------------*/
5   /* Librería de funciones auxiliares para el manejo del UCLMin                 */
6   /*---------------------------------------------------------------------------*/
7   /* Este fichero contiene el código de las funciones de movimiento del UCLMin  */
8   /* es indispensable incluir en los ficheros de código creados por el usuario  */
9   /* el fichero con los headers correspondiente a estas funciones               */
10  /* (libreria_UCLMin.h)                                                        */
11  /*---------------------------------------------------------------------------*/
12  /* Dependencias:                                                              */
13  /*                                                                            */
14  /* -> delay0            - Librería de acceso al temporizador del pic 18f2550  */
15  /* -> p18f2550          - Librería de acceso a los registros del pic 18f2550  */
16  /* -> libreria_UCLMin.h - Fichero de cabecera que contiene las firmas y       */
17  /*                        constantes necesarias para esta librería            */
18  /*---------------------------------------------------------------------------*/
19  /*  Autor: Enrique Holgado de Frutos <enrique.holgado.de.frutos@gmail.com>    */
20  /*  Supervisado por: Francisco Ramos de la Flor                               */
21  /*---------------------------------------------------------------------------*/
22  /*  LICENCIA GPL                                                              */
23  /*****************************************************************************/
24
25  #include "libreria_UCLMin.h"
26
27
28  /*************************************************************/
29  /*                                                           */
30  /* ConfigurarUCLMin                                          */
31  /* ----------------                                          */
32  /*                                                           */
33  /* Funcion encargada de configurar el puerto B del UCLMin    */
34  /* para que sepa leer la informacion de los 2 sensores       */
35  /* infrarrojos, y para que sea capaz de manejar ambos        */
36  /* motores.                                                  */
37  /* Tambien inicializa el temporizador a 0.                   */
38  /*                                                           */
39  /* Parametros de la funcion: Ninguno                         */
40  /* Retorno de la funcion:    Ninguno                         */
41  /*                                                           */
42  /*************************************************************/
43  void ConfigurarUCLMin(void)
44  {
45     //-- Configurar los puertos del UCLMin
46     //-- A7, A6, A5 y A2 como entradas y A4, B2, B1 y B0 como salidas
47     //-- B7, B6, B2 y B0 como entradas y B5, B4, B3 y B1 como salidas
48     //-- RC5 y  RC4 como entradas ya que son el usb
49     //--y C7, C6,  C2, C1 y C0 como salidas
50
51     TRISA = 0xE4;     //0b11100100
52     TRISB = 0xC5;     //0b11000101
53     TRISC = 0x18;     //0b00011000
54
55
56     //-- Configurar el puerto A para que sea digital
57     ADCON1 = 0x0F;
58
59     //-- Inicializar temporizador a 0 utilizando la librería delay0
60     timer0_configurar();
61  }
```

libreria_UCLMin.h

```
1
2   /*****************************************************************************/
3   /* libreria_UCLMin.h      Octubre-2009                                       */
4   /*---------------------------------------------------------------------------*/
5   /* Libreria de firmas de las funciones auxiliares para el manejo del UCLMin  */
6   /*---------------------------------------------------------------------------*/
7   /* Este fichero esta pensado para ser incluido en el programa principal.     */
8   /* Antes de utilizar las funciones de movimiento del UCLMin,es indispensable */
9   /* llamar a la funcion ConfigurarUCLMin().                                   */
10  /*---------------------------------------------------------------------------*/
11  /* Dependencias:                                                             */
12  /*                                                                           */
13  /* -> delay0            - Libreria de acceso al temporizador del pic 18f2550 */
14  /* -> pic18f2550        - Libreria de acceso a los registros del pic 18f2550 */
15  /* -> libreria_UCLMin.c - Fichero que contiene el codigo de las funciones de */
16  /*                        acceso al UCLMin                                   */
17  /*---------------------------------------------------------------------------*/
18  /*  Autor: Enrique Holgado de Frutos <enrique.holgado.de.frutos@gmail.com>   */
19  /*  Supervisado por: Francisco Ramos de la Flor                              */
20  /*---------------------------------------------------------------------------*/
21  /*  LICENCIA GPL                                                             */
22  /*****************************************************************************/
23
24  //Incluimos el pic a utilizar teniendo en cuenta que para su programación se
25  //utilizará el compilador de Microchip MCC18.
26  #include <p18f2550.h>
27
28  //Incluimos las librerias necesarias
29  #include "delay0.h"
30
31  //-- Definiciones para los motores. Estos valores define
32  //-- los movimientos del robot
33  #define   AVANZA     0x1A
34  #define   ATRAS      0x19
35  #define   IZQUIERDA  0x1B
36  #define   DERECHA    0x18
37  #define   STOP       0x00
38
39  //Definiciones de los led colocados de prueba. Teniendo en cuenta que estos led
40  //comparten pin del pic con los sensores infrarrojos y para utilizarlos como
41  //salida habrá que reconfigurarlos para ello.
42
43  #define LED_1 0x04
44  #define LED_2 0x20
45
46  //Definimos los sensores de contacto
47
48  #define Sensor_cont_izq 0x80
49  #define Sensor_cont_der 0x40
50
51  //Definimos los sensores IR
52
53  #define Sensor_IR_izq 0x20
54  #define Sensor_IR_der 0x40
55
56  /*************************************************************
57   *                                                           *
58   * ConfigurarUCLMin                                          *
59   * ----------------                                          *
60   *                                                           *
61   * Funcion encargada de configurar los puertos del UCLMin    *
62   * para que sepa leer la informacion de los 4 sensores       *
63   * infrarrojos, y para que sea capaz de manejar ambos        *
64   * motores.                                                  *
65   * Tambien inicializa el temporizador a 0.                   *
66   *                                                           *
67   * Parametros de la funcion: Ninguno                         *
68   * Retorno de la funcion:    Ninguno                         *
69   *                                                           *
70   *************************************************************/
71  void ConfigurarUCLMin(void);
```

También debemos definir las librerías utilizadas para el temporizador del robot, ya que serán las que nos permitan generar pausas. Estas pausas se utilizarán en los sucesivos programas.

delay0.h

```
1
2   /*****************************************************************************/
3   /* delay0.h      Octubre-2009                                                */
4   /*---------------------------------------------------------------------------*/
5   /* Libreria de temporizacion para el UCLMin                                  */
6   /*---------------------------------------------------------------------------*/
7   /* Este fichero esta pensado para ser incluido en el programa principal.     */
8   /* Antes de utilizar la funcion delay0() es necesario llamar a               */
9   /* timer0_configurar() para configurar correctamente el timer 0              */
10  /*---------------------------------------------------------------------------*/
11  /* Utilizacion del temporizador 0 para generar una pausa de 10ms y           */
12  /* obtener una rutina de delay en unidades de 10ms                           */
13  /* No se utilizan interrupciones                                             */
14  /* El temporizador 0 es de 8 bits                                            */
15  /*---------------------------------------------------------------------------*/
16  /*---------------------------------------------------------------------------*/
17  /*  LICENCIA GPL                                                             */
18  /*****************************************************************************/
19
20  #include "delay0.h"
21
22
23  /*************************************/
24  /* Configurar el temporizador 0     */
25  /*************************************/
26  void timer0_configurar()
27  {
28    //-- Usarlo en modo temporizador, prescaler = 256
29    T0CON=0xC7;
30  }
31
32  /************************************************/
33  /* Hacer una pausa en unidades de 10ms          */
34  /* ENTRADA:                                     */
35  /*   -pausa: Valor de la pausa en decenas de ms */
36  /************************************************/
37  void delay0(unsigned char pausa)
38  {
39
40    //-- Esperar hasta que trancurran pausa ticks de reloj
41    while(pausa>0)
42    {
43      TMR0L=TICKS10;   // Inicializar contador
44      INTCONbits.TMR0IF=0;      // Quitar flag overflow
45      //-- Esperar hasta que bit T0IF se ponga a '1'. Esto
46      //-- indica que han transcurrido 10ms
47      while (!INTCONbits.TMR0IF);
48
49      //-- Ha transcurrido un tick
50      pausa--;
51    }
52  }
```

delay0.c

```
/*****************************************************************************/
/* delay0.h      Septiembre-2009                                             */
/*---------------------------------------------------------------------------*/
/* Libreria de temporizacion para el UCLMin                                  */
/*---------------------------------------------------------------------------*/
/* Este fichero esta pensado para ser incluido en el programa principal.     */
/* Antes de utilizar la funcion delay0() es necesario llamar a               */
/* timer0_configurar() para configurar correctamente el timer 0              */
/*---------------------------------------------------------------------------*/
/* Utilizacion del temporizador 0 para generar una pausa de 10ms y           */
/* obtener una rutina de delay en unidades de 10ms                           */
/* No se utilizan interrupciones                                             */
/* El temporizador 0 es de 8 bits                                            */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/*  LICENCIA GPL                                                             */
/*****************************************************************************/

#ifndef _DELAYH #define _DELAYH

#include <p18f2550.h>

//-- Definiciones
#define TICKS10 0x3D  // Valor con el que inicializar contador para
                      // conseguir TICKs de 10ms

/*************************************/
/* Configurar el temporizador 0     */
/*************************************/
void timer0_configurar(void);

/************************************************/
/* Hacer una pausa en unidades de 10ms          */
/* ENTRADA:                                     */
/*   -pausa: Valor de la pausa en decenas de ms */
/************************************************/
void delay0(unsigned char pausa);

#endif
```

5.3. Programas básicos

En esta sección se van a explicar los programas más sencillos probados en el robot. Para facilitar su comprensión se han utilizado un diagrama de flujo de datos.

Los diagramas de flujo son una manera de representar visualmente el flujo de datos a través de sistemas de tratamiento de información. Los diagramas de flujo describen que operaciones y en que secuencia se requieren para solucionar un problema dado.

5.3.1. Encender un led

Es el programa "Hola Mundo"[1] que se desarrolla en cualquier plataforma. Es el primer programa que se desarrolla en cualquier robot cuando se empieza con él. Lo que se intenta con este programa es familiarizarse con el microcontrolador, con sus puertos y activar una salida de esos puertos.

main.c

```
1
2   /*************************************************************/
3   /*Programa realizado por:                                    */
4   /*                                                           */
5   /*         - Enrique Holgado de Frutos                       */
6   /*Supervisado por:                                           */
7   /*         - Francisco Ramos de la Flor                      */
8   /*                                                           */
9   /*************************************************************/
10  /*                                                           */
11  /*   Este programa tiene como objetivo encender un led       */
12  /*colocado en las salida RA2.                                */
13  /*                                                           */
14  /*************************************************************/
15  /*                                                           */
16  /*   Fecha: 10-Octubre-2009                                  */
17  /*   Versión del programa: 1.0                               */
18  /*                                                           */
19  /*************************************************************/
20
21  // Especificar la librería a utilizar
22  #include <p18F2550.h>
23  #include "libreria_UCLMin.h"
24
25
26  /***********************************************/
27  /*                                             */
28  /*              CUERPO DEL PROGRAMA            */
29  /*                                             */
30  /***********************************************/
31
32  void main (void)
33  {
34      TRISA=0xE0;         //Definimos el LED_1 como salida.
35      RA2=0
36      PORTA=LED_1;        //Igualamos las salidas del puerto al valor del LED
37                          //de esta manera enciende el led.
38     //-- Fin
39  }
```

[1]En informática, un programa Hola mundo (o Hello World, en inglés) es el que imprime el texto «¡Hola, mundo!» en un dispositivo de visualización (generalmente una pantalla de monitor). Se suele usar como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico.

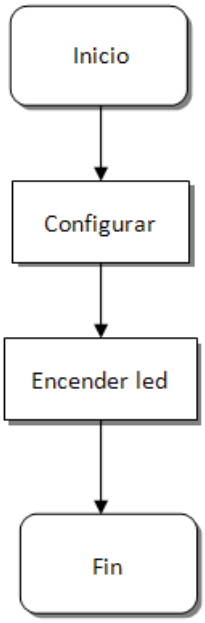


Figura 5.1: diagrama de flujo de datos

5.3.2. Encender un led realimentando información de los sensores de contacto

Una vez que hemos sido capaces de encender un led, el siguiente paso es encender un led cuando pulsamos un botón, de esta manera si el botón no se pulsa el led permanecerá apagado.

main.c

```c

/*************************************************************/
/*Programa realizado por:                                    */
/*         - Enrique Holgado de Frutos                       */
/*Supervisado por:                                           */
/*         - Francisco Ramos de la Flor                      */
/*                                                           */
/*************************************************************/
/*                                                           */
/*  Este programa tiene como objetivo encender un led        */
/*colocado en las salidas RA5 y RA2 al pulsar los sensores   */
/*de contacto colocados en RB6 y RB7, lo harán respectivamen-*/
/*te. RA5 con RB6 y RA2 con RB7.                             */
/*                                                           */
/*************************************************************/
/*                                                           */
/*  Fecha: 20-Octubre-2009                                   */
/*  Versión del programa: 1.0                                */
/*                                                           */
/*************************************************************/


// Especificamos las librerias necesarias
#include "libreria_UCLMin.h"



/****************************/
/*                          */
/*   Comienzo del programa  */
/*                          */
/****************************/


void main(void)
{
   TRISA=0x00;

   while(1)                 //bucle infinito.
   {
      if(PORTB=SENSOR_CONT_IZQ) //Se activa el sensor de contacto derecho
      {
         PORTA = LED_1;      // Enciende el led situado en la salida RA2
      }
      else
      {
         PORTA = 0x00;      //No enciende nada
      }

      if(PORTB=SENSOR_CONT_DER)  //Se activa el sensor de contacto izquierdo
      {
         PORTA = LED_2;      // Enciende el led situado en la salida RA5
      }
      else
      {
         PORTA = 0x00;      //No enciende nada
      }
   }
    //-- Fin
}
```

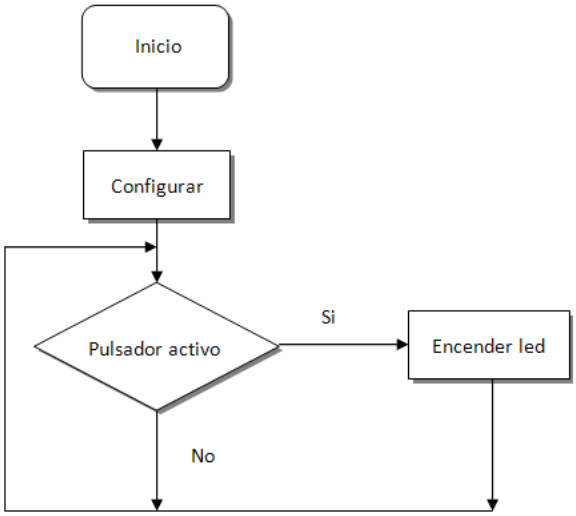


Figura 5.2: diagrama de flujo de datos del programa

5.3.3. Temporizadores

Una vez que se ha conseguido comprender el funcionamiento y la activación y desactivación de los puertos de entrada y de salida, se pasa a trabajar con temporizadores. Para este cometido se crea una librería que hace uso de la señal externa de reloj mencionada. Esta librería es la llamada delay0.

El ejemplo que se ha hecho para trabajar con temporizadores es bastante sencillo, se va a encender y a apagar un led y se hará durante un período de tiempo, el que nosotros definamos.

main.c

```c

/*************************************************************/
/*Programa realizado por:                                    */
/*                                                           */
/*         - Enrique Holgado de Frutos                       */
/*Supervisado por:                                           */
/*         - Francisco Ramos de la Flor                      */
/*                                                           */
/*************************************************************/
/*                                                           */
/*   Este programa tiene como objetivo encender y apagar     */
/* un led colocado en la salida del pin RA2.                 */
/*                                                           */
/*                                                           */
/*************************************************************/
/*                                                           */
/*   Fecha: 15-Octubre-2009                                  */
/*   Versión del programa: 1.0                               */
/*                                                           */
/*************************************************************/

// Incluimos la librería del UCLMin
#include "libreria_UCLMin.h"

/***********************************************/
/*                                             */
/*              CUERPO DEL PROGRAMA            */
/*                                             */
/***********************************************/


void main(void)
{
    ConfigurarUCLMin();       //Configuración de entradas, salidas y temporizador
    TRISA=0x00;               //Configuramos el puerto A como salidas


    while(1)
    {
        PORTA= LED_1;       // Cambia el valor del bit RA2 del puerto
        delay0(100);        // Cuenta hasta un segundo
        PORTA= 0x00;        // Se pone a cero
        delay0(100);        // Cuenta hasta un segundo

    }
   //-- Fin
}
```

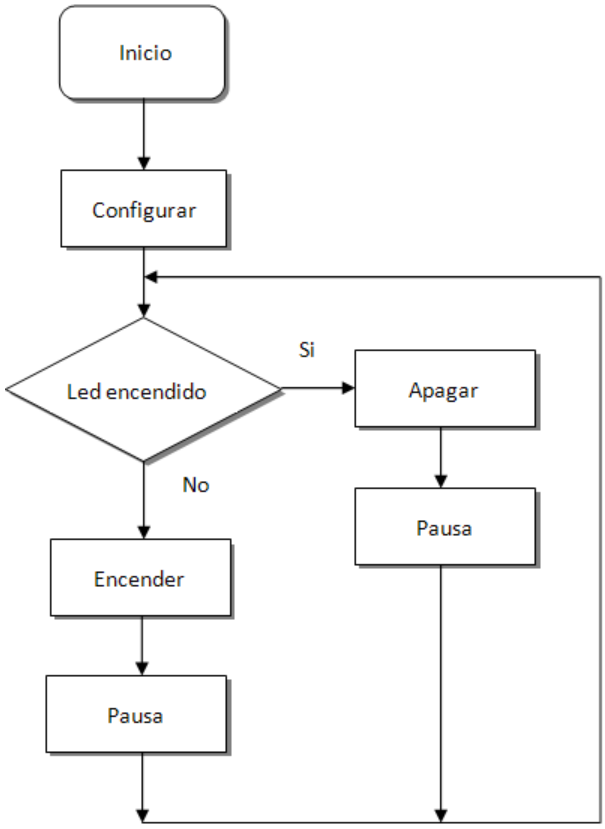


Figura 5.3: diagrama de flujo de datos del programa con temporizadores

5.3.4. Cambio de dirección tras chocar con un obstáculo

En este último programa de esta Sección 5.3. En este programa se unen todos los anteriormente citados. EL robot se va a mover hacia delante hasta que cualquiera de sus dos sensores de contacto se active. Cuando esto ocurra el robot retrocederá y, en función del sensor que se haya activado, girará en un sentido o en otro.

main.c

```c

/*************************************************************/
/*Programa realizado por:                                    */
/*         - Enrique Holgado de Frutos                       */
/*Supervisado por:                                           */
/*         - Francisco Ramos de la Flor                      */
/*                                                           */
/*                                                           */
/*************************************************************/
/*                                                           */
/*   Este programa tiene como objetivo mover el robot        */
/*hacia delante, siempre y cuando no encuentre en su camino  */
/*un obstaculo que active un sensor de contacto. En ese caso */
/*el robot marchará para atrás y girará hacia el lado opuesto*/
/*del obstáculo para poder esquivarlo siguiendo su marcha    */
/*hacia delante.                                             */
/*                                                           */
/*************************************************************/
/*                                                           */
/*   Fecha: 19-Octubre-2009                                  */
/*   Versión del programa: 1.0                               */
/*************************************************************/

// Incluimos la libreria de nuestro robot para hacer las
// definiciones del PIC, etc.
#include "libreria_UCLMin.h"

/****************************/
/*                          */
/*   Comienzo del programa  */
/*                          */
/****************************/


void main(void) {
   ConfigurarUCLMin();       //Configuración de entradas, salidas y temporizador

    while(1)                  //Bucle infinito
   {
       PORTA=AVANZA;         //El robot avanza hasta que encuentre un obstaculo

       if(PORTB=SENSOR_CONT_DER)  //Se activa el sensor de contacto derecho
       {
           PORTA=ATRAS;      //El robot retrocede durante (200x10mil) 2 segundos
           delay0(200);
           PORTA=IZQUIERDA;  //El robot gira hacia la izquierda durante  2 segundos
           delay0(200);
       }
       if(PORTB=SENSOR_CONT_IZQ)   //Se activa el sensor de contacto izquierdo
       {

           PORTA=ATRAS;     //El robot retrocede durante  2 segundos
           delay0(200);
           PORTA=DERECHA;   //El robot gira hacia la derecha durante 2 segundos
           delay0(200);
      }

      PORTA=AVANZA;         //El robot avanza hasta que encuentre un obstaculo

     }
        //-- Fin
}
```

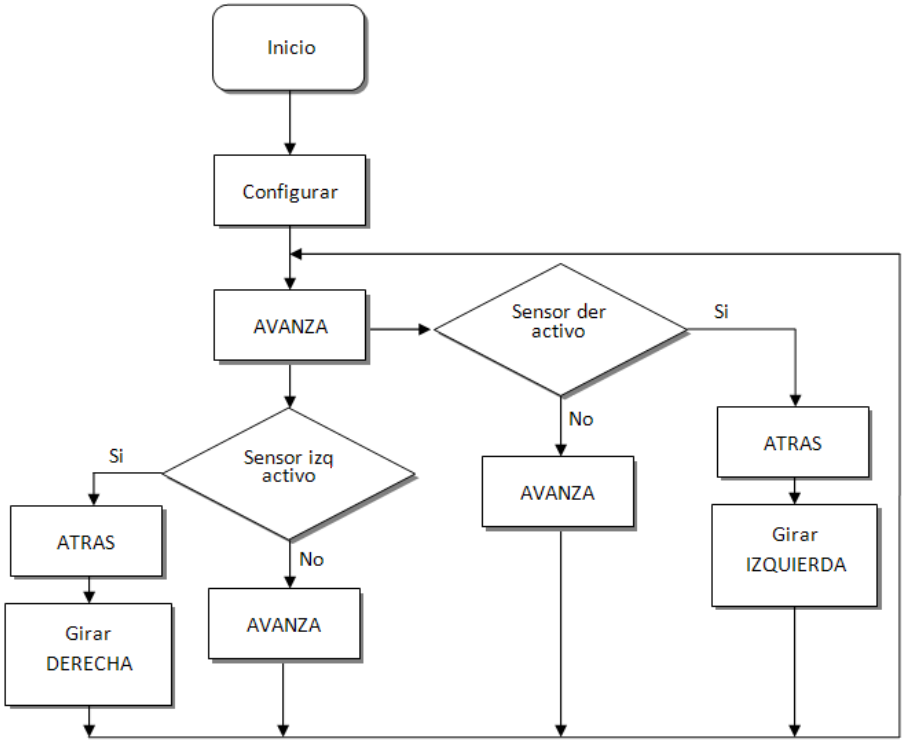


Figura 5.4: diagrama de flujo de datos del programa

5.4. Seguidor de líneas

Como ya se ha dicho en el Capítulo 4 el robot va dotado con dos sensores IR. El programa consiste en seguir un circuito dibujado en negro sobre un fondo blanco.

```
1
2   /*************************************************************/
3   /*Programa realizado por:                                    */
4   /*                                                           */
5   /*         - Enrique Holgado de Frutos                       */
6   /*         - Francisco Ramos de la Flor                      */
7   /*                                                           */
8   /*************************************************************/
9   /*                                                           */
10  /*   Este programa tiene como objetivo seguir la línea       */
11  /*  de un circuito                                           */
12  /*                                                           */
13  /*************************************************************/
14
15  #include "libreria_UCLMin.h"
16
17  //Hacemos las definiciones específicas para este programa
18  unsigned char sensores;
19  #define IZQUIERDO SENSOR_IR_IZQ
20  #define DERECHO   SENSOR_IR_DER
21
22
23  /****************************/
24  /*                          */
25  /*   Comienzo del programa  */
26  /*                          */
27  /****************************/
28
29
30  void main(void)
31  {
32      ConfigurarUCLMin();     //Configuración de entradas,
33                              //salidas y temporizador
34
35
36  PORTB=STOP;
37
38     for (;;)
39       {
40        sensores=PORTA& (IZQUIERDO | DERECHO); // Leer sensores
41        switch (sensores)
42            {
43           case (IZQUIERDO | DERECHO):  //-- Caso negro-negro
44              PORTA=AVANZA;
45              break;
46           case DERECHO:  //-- Caso Blanco-negro
47              PORTA=IZQUIERDA;
48              break;
49           case IZQUIERDO:  //-- Caso Negro-Blanco
50              PORTA=DERECHA;
51              break;
52           default:
53              PORTA=STOP;
54        }
55     }
56  }
57
```

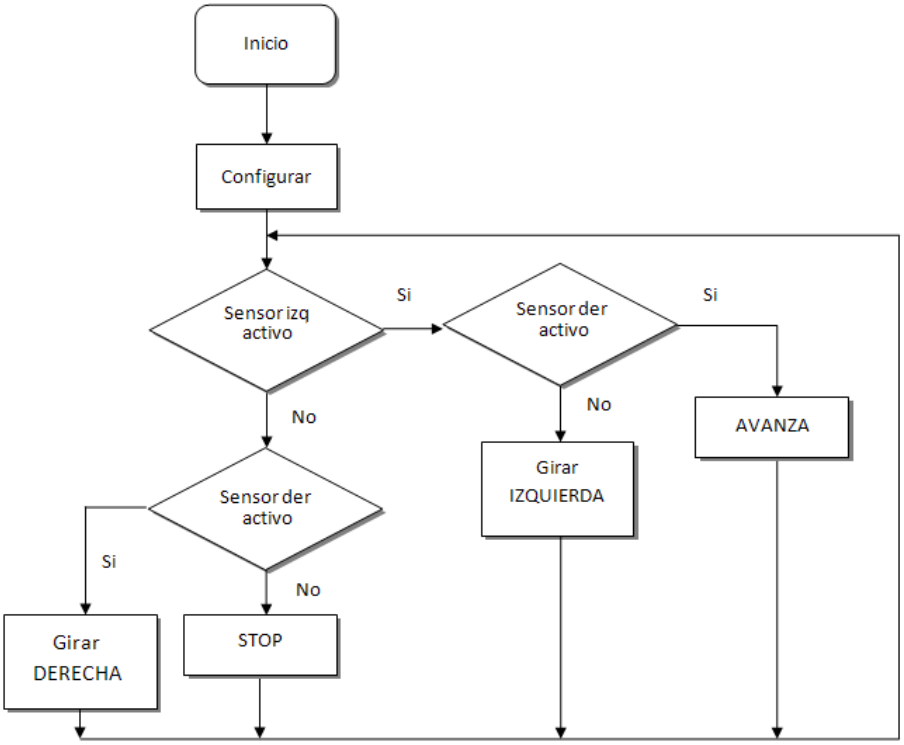


Figura 5.5: Diagrama de flujo de datos para el programa de sensores IR

5.5. Programa de comunicación via MiWi

Una vez que se domina con relativa facilidad las funciones más básicas del PIC, se desarrolla este último programa, en el cual se pueden monitorizar los cuatro sensores con los que va dotado el UCMLin.

Para que la comunicación sea posible necesitamos hacer unas definiciones previas sobre nuestra placa (para hacer estas definiciones se ha utilizado el ZENA, véase Sección B.6). Estas definiciones se almacenan en un archivo llamado MiWiDefs. Son definiciones relativas a la comunicación entre el módulo y el microcontrolador, la posición de los sensores en la placa,etc. El Diagrama de flujo de datos de este programa está en la Sección E.2

main.c

```
1
2   ....
3
4               #ifdef __PIC32MX__
5
6               if((PUSH_BUTTON_2 == 0) || Pushbutton_2_Wakeup_Pressed)
7
8               #else
9
10              if(PUSH_BUTTON_2 == 0)
11
12              #endif
13              {
14                  #ifdef __PIC32MX__
15
16                  if((PUSH_BUTTON_2_pressed == FALSE) || Pushbutton_2_Wakeup_Pressed)
17
18                  #else
19
20                  if(PUSH_BUTTON_2_pressed == FALSE)
21
22                  #endif
23                  {
24                      static BYTE transmitMode = 0;
25
26                      PUSH_BUTTON_2_pressed = TRUE;
27
28                      #ifdef __PIC32MX__
29
30                      Pushbutton_2_Wakeup_Pressed = 0;
31
32                      #endif
33
34
35                      TxPayLoad();
36                      WriteData(USER_REPORT_TYPE);
37                      WriteData(LIGHT_REPORT);
38                      WriteData(LIGHT_TOGGLE);
39
40
41
42                      if( myFriend != 0xFF )  // socket has already been established
43                      {
44                          SendReportByHandle(myFriend, FALSE);
45                          ConsolePutROMString((ROM char*)
46                          "Send Report by Handle(Socket)\r\n");
47                      }
48                      else
49                      {
50                          // if no socket, send report by long or short address
51                          if( (transmitMode++ % 2) == 0 )
52                          {
53                              tempLongAddress[0] = 0x93;
54                              tempLongAddress[1] = 0x78;
55                              tempLongAddress[2] = 0x56;
```

```
56                              tempLongAddress[3] = 0x34;
57                              tempLongAddress[4] = 0x12;
58                              tempLongAddress[5] = 0xA3;
59                              tempLongAddress[6] = 0x04;
60                              tempLongAddress[7] = 0x00;
61
62                              SendReportByLongAddress(tempLongAddress);
63                              ConsolePutROMString((ROM char*)
64                              "Send Report by Long Address\r\n");
65                          }
66                          else
67                          {
68                              tempShortAddress.Val = 0x0000;
69                              SendReportByShortAddress(myPANID,
70                                                       tempShortAddress, FALSE);
71                              ConsolePutROMString((ROM char*)
72                              "Send Report by Short Address\r\n");
73                          }
74                      }
75                  }
76                  PUSH_BUTTON_2_press_time = TickGet();
77              }
78              else
79              {
80                  TICK t = TickGet();
81
82                  tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_2_press_time);
83
84                  if(tickDifference.Val > DEBOUNCE_TIME)
85                  {
86                      PUSH_BUTTON_2_pressed = FALSE;
87                  }
88              }
89
90
91          #ifdef __PIC32MX__
92
93              if((PUSH_BUTTON_1 == 0) || Pushbutton_2_Wakeup_Pressed)
94
95              #else
96
97              if(PUSH_BUTTON_1 == 0)
98
99              #endif
100             {
101                 #ifdef __PIC32MX__
102
103                 if((PUSH_BUTTON_1_pressed == FALSE) || Pushbutton_2_Wakeup_Pressed)
104
105                 #else
106
107                 if(PUSH_BUTTON_1_pressed == FALSE)
108
109                 #endif
110                 {
111                     static BYTE transmitMode = 0;
112
113                     PUSH_BUTTON_1_pressed = TRUE;
114
115                     #ifdef __PIC32MX__
116
117                     Pushbutton_1_Wakeup_Pressed = 0;
118
119                     #endif
120
121
122
123                     TxPayLoad();
124                     WriteData(USER_REPORT_TYPE_1);
125                     WriteData(LIGHT_REPORT);
126                     WriteData(LIGHT_TOGGLE);
127
128
129
130                     if( myFriend != 0xFF )  // socket has already been established
131                     {
132                         SendReportByHandle(myFriend, FALSE);
133                         ConsolePutROMString((ROM char*)
134                         "Send Report by Handle(Socket)\r\n");
```

```
135                     }
136                     else
137                     {
138                         // if no socket, send report by long or short address
139                         if( (transmitMode++ % 2) == 0 )
140                         {
141                             tempLongAddress[0] = 0x93;
142                             tempLongAddress[1] = 0x78;
143                             tempLongAddress[2] = 0x56;
144                             tempLongAddress[3] = 0x34;
145                             tempLongAddress[4] = 0x12;
146                             tempLongAddress[5] = 0xA3;
147                             tempLongAddress[6] = 0x04;
148                             tempLongAddress[7] = 0x00;
149
150                             SendReportByLongAddress(tempLongAddress);
151                             ConsolePutROMString((ROM char*)
152                             "Send Report by Long Address\r\n");
153                         }
154                         else
155                         {
156                             tempShortAddress.Val = 0x0000;
157                             SendReportByShortAddress(myPANID,
158                                                      tempShortAddress, FALSE);
159                             ConsolePutROMString((ROM char*)
160                             "Send Report by Short Address\r\n");
161                         }
162                     }
163                 }
164                 PUSH_BUTTON_1_press_time = TickGet();
165             }
166             else
167             {
168                 TICK t = TickGet();
169
170                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_1_press_time);
171
172                 if(tickDifference.Val > DEBOUNCE_TIME)
173                 {
174                     PUSH_BUTTON_1_pressed = FALSE;
175                 }
176             }
177
178
179
180
181
182 #ifdef __PIC32MX__
183
184             if((PUSH_BUTTON_3 == 0) || Pushbutton_2_Wakeup_Pressed)
185
186             #else
187
188             if(PUSH_BUTTON_3 == 0)
189
190             #endif
191             {
192                 #ifdef __PIC32MX__
193
194                 if((PUSH_BUTTON_3_pressed == FALSE) || Pushbutton_3_Wakeup_Pressed)
195
196                 #else
197
198                 if(PUSH_BUTTON_3_pressed == FALSE)
199
200                 #endif
201                 {
202                     static BYTE transmitMode = 0;
203
204                     PUSH_BUTTON_3_pressed = TRUE;
205
206                     #ifdef __PIC32MX__
207
208                     Pushbutton_3_Wakeup_Pressed = 0;
209
210                     #endif
211
212                     TxPayLoad();
213                     WriteData(USER_REPORT_TYPE_2);
```

```
214                     WriteData(LIGHT_REPORT);
215                     WriteData(LIGHT_TOGGLE);
216
217                     if( myFriend != 0xFF )  // socket has already been established
218                     {
219                         SendReportByHandle(myFriend, FALSE);
220                         ConsolePutROMString((ROM char*)
221                         "Send Report by Handle(Socket)\r\n");
222                     }
223                     else
224                     {
225                         // if no socket, send report by long or short address
226                         if( (transmitMode++ % 2) == 0 )
227                         {
228                             tempLongAddress[0] = 0x93;
229                             tempLongAddress[1] = 0x78;
230                             tempLongAddress[2] = 0x56;
231                             tempLongAddress[3] = 0x34;
232                             tempLongAddress[4] = 0x12;
233                             tempLongAddress[5] = 0xA3;
234                             tempLongAddress[6] = 0x04;
235                             tempLongAddress[7] = 0x00;
236
237                             SendReportByLongAddress(tempLongAddress);
238                             ConsolePutROMString((ROM char*)
239                             "Send Report by Long Address\r\n");
240                         }
241                         else
242                         {
243                             tempShortAddress.Val = 0x0000;
244                             SendReportByShortAddress(myPANID,
245                                                      tempShortAddress, FALSE);
246                             ConsolePutROMString((ROM char*)
247                             "Send Report by Short Address\r\n");
248                         }
249                     }
250                 }
251                 PUSH_BUTTON_3_press_time = TickGet();
252             }
253             else
254             {
255                 TICK t = TickGet();
256
257                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_3_press_time);
258
259                 if(tickDifference.Val > DEBOUNCE_TIME)
260                 {
261                     PUSH_BUTTON_3_pressed = FALSE;
262                 }
263             }
264
265
266         #ifdef __PIC32MX__
267
268             if((PUSH_BUTTON_4 == 0) || Pushbutton_4_Wakeup_Pressed)
269
270             #else
271
272             if(PUSH_BUTTON_4 == 0)
273
274             #endif
275             {
276                 #ifdef __PIC32MX__
277
278                 if((PUSH_BUTTON_4_pressed == FALSE) || Pushbutton_4_Wakeup_Pressed)
279
280                 #else
281
282                 if(PUSH_BUTTON_4_pressed == FALSE)
283
284                 #endif
285                 {
286                     static BYTE transmitMode = 0;
287
288                     PUSH_BUTTON_4_pressed = TRUE;
289
290                     #ifdef __PIC32MX__
291
292                     Pushbutton_4_Wakeup_Pressed = 0;
```

```
293
294                     #endif
295
296                     TxPayLoad();
297                     WriteData(USER_REPORT_TYPE_3);
298                     WriteData(LIGHT_REPORT);
299                     WriteData(LIGHT_TOGGLE);
300
301                     if( myFriend != 0xFF )  // socket has already been established
302                     {
303                         SendReportByHandle(myFriend, FALSE);
304                         ConsolePutROMString((ROM char*)
305                         "Send Report by Handle(Socket)\r\n");
306                     }
307                     else
308                     {
309                         // if no socket, send report by long or short address
310                         if( (transmitMode++ % 2) == 0 )
311                         {
312                             tempLongAddress[0] = 0x93;
313                             tempLongAddress[1] = 0x78;
314                             tempLongAddress[2] = 0x56;
315                             tempLongAddress[3] = 0x34;
316                             tempLongAddress[4] = 0x12;
317                             tempLongAddress[5] = 0xA3;
318                             tempLongAddress[6] = 0x04;
319                             tempLongAddress[7] = 0x00;
320
321                             SendReportByLongAddress(tempLongAddress);
322                             ConsolePutROMString((ROM char*)
323                             "Send Report by Long Address\r\n");
324                         }
325                         else
326                         {
327                             tempShortAddress.Val = 0x0000;
328                             SendReportByShortAddress(myPANID,
329                                                      tempShortAddress, FALSE);
330                             ConsolePutROMString((ROM char*)
331                             "Send Report by Short Address\r\n");
332                         }
333                     }
334                 }
335                 PUSH_BUTTON_4_press_time = TickGet();
336             }
337             else
338             {
339                 TICK t = TickGet();
340
341                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_4_press_time);
342
343                 if(tickDifference.Val > DEBOUNCE_TIME)
344                 {
345                     PUSH_BUTTON_4_pressed = FALSE;
346                 }
347             }
348
349 ....
350
```

6

PRESUPUESTO

En este capítulo se intentará a modo de resumen, ver el coste real del proyecto, principalmente se estudiará el precio de las placas que se necesitan para ello.

6.1. Robot

6.1.1. Placa UCLMin

A continuación se detallarán los componentes utilizados para la placa y su precio, para así poder obtener un precio orientativo del proyecto.

Microcontrolador PIC18F2550	4,81 €
Módulo de RF MRF24J40MA	8,40 €
Regulador de tensión a 3.3 V TPS79533DCQ	2.78 €
Regulador de tensión a 5 V MAX603	4,10 €
Resistencia(x7, varias)	2,00 €
Condensador(x13, varios)	6,90 €
Led(x5)	1,50 €
Conector (x8,varios)	4,50 €
Total	**39,49 €**

Tabla 6.1: Presupuesto de UCLMin

6.1.2. Placa Sky293

Esta placa aunque no ha sido diseñada y construida en la escuela también es necesaria para el funcionamiento del robot.

Puente H, L293	2,94 €
Conector(x11,varios)	8,40 €
Resistencia(x13,varias)	3,10 €
Conversor Analógico-Digital 40106D	3,20 €
Relé	2,83 €
Total	**20,47 €**

Tabla 6.2: Presupuesto de Sky293

6.1.3. Baterías LiPo

Se han comprado dos baterías distintas para las dos alimentaciones (Para más información ver Sección 4.2). Sus precios son los siguientes:

ZIPPY 850mAh 20C single cell	2,90 €
ZIPPY Flightmax 1600mAh 2S1P 20C	9,80 €
Total	**12,70 €**

El problema de estas baterías es que no son fáciles de encontrar. Se fabrican en EEUU(Estados Unidos) y hay que pedirlas por internet. Ésto provoca un aumento en el precio final de las baterías ya que los portes desde allí son altos.

6.1.4. Parte mecánica

La parte mecánica comprende los motores y las piezas de metacrilato. El precio del metacrilato es de 9 € por una pieza de 35x35. Con cada una de esas piezas podría construirse una unidad estructural. En cuanto a los motores, no son más que servos modificados. El precio de un servo es de 9 €. Por lo que el precio de la estructura mecánica sería de:

Pieza de metacrilato de 35x35cm	9 €
2 Servos	18 €
Total	**27 €**

La parte que posteriormente se modificará y mejorará es la parte del robot. Esta parte tiene un precio de 99,66 €(representa la suma del precio de la UCLMin (39,49 €), la Sky293 (20,47 €), las baterías (12,70 €) y la estructura mecánica (27 €)).

6.2. Material adicional

Este material no es necesario para la construcción del prototipo pero es bastante recomendable para desarrollar aplicaciones por Radiofrecuencia. El cargador es muy importante porque las baterías LiPo hay que cargarlas de manera equilibrada entre sus elementos, para que no haya luego problemas al descargarlas.

6.2.1. Placa PICDEM Z

La placa PICDEM Z ha sido comprada directamente en Microchip. En nuestro caso para hacernos con una placa de éstas decidimos comprar el kit PICDEM Z MRF24J40 2.4 GHz Dev Kit (Figura 6.1).

Figura 6.1: kit PICDEM Z MRF24J40 2.4 GHz Dev Kit

kit PICDEM Z MRF24J40 2.4 GHz Dev Kit.	240,96 €
Total	**240,96 €**

6.2.2. Cargador y variador para baterías LiPo

Las baterías LiPo al ser recargables necesitan un cargador. Son baterías bastante delicadas y necesitan uno especial. Se ha comprado para este fin el cargador de la Figura 6.2.

Figura 6.2: Cargador/Balanceador de baterías LiPo

Turnigy Accucel-6 50W 5A Balancer/Charger	28,50 €
Total	**28,50 €**

7

CONCLUSIONES

7.1. Conclusiones

El diseño y construcción de una base o plataforma de robótica móvil no es una tarea tan sencilla como a priori, a la vista de los diseños existentes, pueda parecer. Si se pretende construir algo más que un mero juguete, es necesario disponer de una serie de conocimientos relativos a múltiples disciplinas, que inicialmente, a la hora de comenzar el estudio del proyecto no se tienen en cuenta.

Debe realizarse un estudio los más exhaustivo posible porque cualquier cosa no considerada previamente va a derivar en un potencial problema y en improvisación. Ha de tenerse en cuenta el carácter multidisciplinar de los proyectos de robótica. No sólo implica el diseño de la electrónica, también se debe pensar en la mecánica del mismo y en no cerrar el diseño de manera excesiva para permitir avances, ya que un robot nunca termina, siempre puede perfeccionarse.

En la realización de este proyecto se han aplicado conocimientos de informática en el desarrollo de las aplicaciones para el robot. La utilización de lenguajes de alto nivel, C, y de compilador, permite lograr una mayor abstracción respecto al hardware utilizado, es decir, del tipo de microcontrolador.

También se han aplicado conocimientos de electrónica. Debemos saber corrientes soportadas por los componentes, tensiones, regímenes de funcionamiento,

etc.

A lo largo del desarrollo del proyecto hemos ido cumpliendo una serie de etapas:

- Se hizo un estudio del estado del arte de la robótica móvil.
- Una vez que nos decidimos por un modelo RMR diseñamos nuestra arquitectura Hardware y se montó en una placa de pruebas. Este es un paso obligatorio en todos los proyectos en los que se diseña un dispositivo, ya que se tiene que comprobar el correcto funcionamiento de cada uno de los elementos de hardware.
- Cuando se comprobó que el software y el hardware funcionaban correctamente, pudimos pasar a diseñar el PCB de las placas para corroborar el correcto funcionamiento tanto del software como del hardware sobre los dispositivos finales.
- Como se puede ver el tamaño de la placa UCLMin no está optimizado, podría hacerse bastante más pequeña pero por cuestiones de comodidad para montarla sobre el robot se hizo del mismo tamaño que la Sky293.

Los objetivos del proyecto eran múltiples:

- Diseñar y construir un nodo de comunicaciones capaz de comunicarse con diferentes dispositivos utilizando un protocolo de comunicación inalámbrica.
- Diseñar y construir una placa lo más genérica posible que permitiera controlar el robot de manera autónoma, intentando además que fuera lo más actual posible para facilitar el trabajo con él.
- Crear una aplicación que aúne la comunicación inalámbrica con el movimiento autónomo del mismo.

Todos los objetivos han sido cumplidos como se ha visto en el desarrollo de la memoria. La placa construida nos permite controlar el robot para que pueda moverse por un entorno desconocido.

7.2. Propuestas de mejora

Respecto al trabajo futuro que parta de este proyecto, seguramente continuará en la línea de una o varias de las siguientes direcciones:

- Optimización de la placa UCLMin y mejora y reducción de los componentes de la Sky293, para poder hacer una única placa de menor tamaño e igual funcionalidad. Para ello lo mejor será utilizar componentes SMD.

- Sustitución o inclusión de elementos que nos permitan monitorizar el estado de las baterías ya que son un punto delicado del robot. Enviar una señal por Radiofrecuencia al ordenador cuando el nivel de tensión de la batería llegue cerca del punto crítico.
- Inclusión de nuevos sensores como podrían ser los ultrasonidos. Éstos nos permiten medir la distancia a los obstáculos sin necesidad de colisionar.
- Ampliación del programa de comunicación inalámbrica para poder comunicarlo con otros robots y emprender así labores cooperativas.

  Si se consigue comunicar dos robots de manera inalámbrica podría plantearse montar una competición real entre robots a modo de Robocode[1]
- Realizar pruebas de la placa haciendo uso de todas sus funcionalidades. Para ello se propone el diseño de un mecanismo que posea:
  - Ruedas como medio de locomoción
  - Necesite una aplicación para los cuatro servos que el robot puede controlar.
  - Permitir la toma de datos del entorno a través de los sensores y posteriormente enviarlos a la unidad central de coordinación y/o a otros robots móviles, para hacer uso de la comunicación por Radiofrecuencia.

  Todo eso podemos aunarlo en un vehículo que ya existe: una carretilla elevadora (Figura 7.1).

  Es un mecanismo muy utilizado en la industria. Es un aparato autónomo apto para llevar cargas en voladizo, posee dos ejes (como un automóvil) y en la parte delantera del vehículo tiene una plataforma compuesta por dos barras planas paralelas planas, llamadas horquillas, están unidas a un mástil de elevación para la manipulación de los pallets.

Figura 7.1: Carretilla elevadora

[1]Robocode es un programa basado en la plataforma Java que consiste en programar tanques para que se muevan por un entorno de manera autónoma y dispare a los contrincantes que encuentre en su camino de la manera en que se programe.

Bibliografía

[*Avances ZigBee,2008*] Página centrada en los avances de la comunicación ZigBee
http://www.zigbee.es/wp/

[*Berman, Schechtman y Edan, 2009*] Sigal Berman, Edna Schechtman and Yael Edan (2009).
"**Evaluation of automatic guided vehicle systems**"
Robotics and Computer-Integrated Manufacturing, Vol. 25, No. 3, pages "522-528".

[*Bayindir y Sahin, 2005*] Levent Bayindir y Sahin (2005).
"**A review of studies in swarm robotics**"
Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 10, No. 2, pages "49-53".

[*Datasheet 18f2550 microchip,2008*] Datasheet del microcontrolador utilizado
http://www.microchip.com

[*Datasheet MRF24J40MA microchip, 2006*] Datasheet del módulo MRF24J40MA
http://www.microchip.com

[*Datasheet MRF24J40 microchip, 2006*] Datasheet del chip de RF MRF24J40
http://www.microchip.com

[*Fernández y Feliu, 2008*] Raúl Fernández Rodríguez (2008).
"**Diseño y construcción de un robot trepador para la inspección de depósits de combustible**"
Memoria de período docente e investigador de estudios de tercer ciclo. Programa de doctorado: Mecatrónica

[*Malave Mauriem, 2006*] Mariam Malave Mauriem (2006).
"**Sistemas de comunicación inalámbrica, transmisión de voz**".
http://neutron.ing.ucv.ve/revista-e/No2/mauriem.html.

[*Mariscal García, 2005*] Efraín Mariscal García (2005).
"**Planeación y seguimiento de trayectorias de robots móviles en una simulación de un ambiente real**".
Ra Ximhai, enero-abril,año/vol 1, No. 1.

[*Monografía, ZigBee*] Monografías, Artículo relacionado con la comunicación ZigBee en relación con la domótica
http://www.monografias.com

[*pfc Nº 08-09-200182, 2008*] Miguel Domingo García Jiménez (2008).
"**Desarrollo de un sistema basado en microcontrolador para uso genérico y su aplicación a un AGV**"
pfc Nº 08-09-200182

[*wikipedia*] Wikipedia, artículos relacionados con los distintos tipos de comunicacíon inalámbrica
http://es.wikipedia.org/wiki

[*ZigBee*] Artículo relacionado con el sistema de comunicación ZigBee
http://artiana.gotdns.com/dynamic/Articulos/ZigBee.html

[*Zheng,Li Guo y Yang, 2008*] Taixiong Zheng, Rui Li, Wenhao Guo and Liangyi Yang (2008).
"**Multi-robot Cooperative Task Processing in Great Environment**"
Robotics, Automation and Mechatronics, IEEE Conference on pages "1113-1117".

Anexos

[**A**] Esquemático de las placas

[**B**] Descripción del software utilizado

[**C**] Comparativa de Microcontroladores

[**D**] Comparativa de comunicaciones inalámbricas

[**E**] Códigos de programación

[**F**] Características técnicas de los elementos de la placa

[**G**] Conversión de Binario a Hexadecimal

A

ESQUEMÁTICO DE LAS PLACAS

En este primer anexo se van a incluir los esquemáticos de las placas que van sobre el robot. Los esquemáticos se han hecho en EAGLE (Sección B.1.1). Las placas que van montadas en el robot son las siguientes:

- Sky293.
- UCLMin.

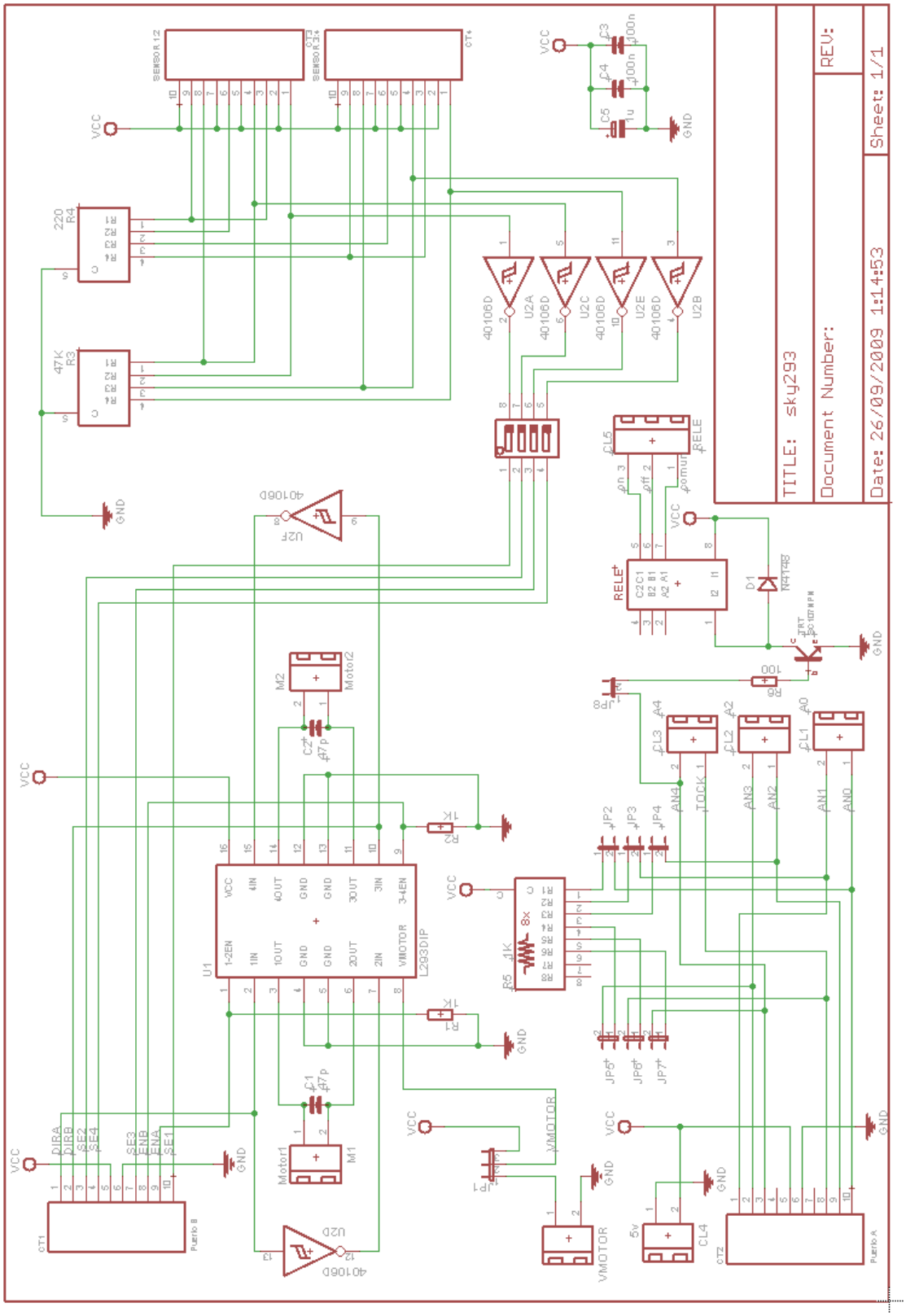
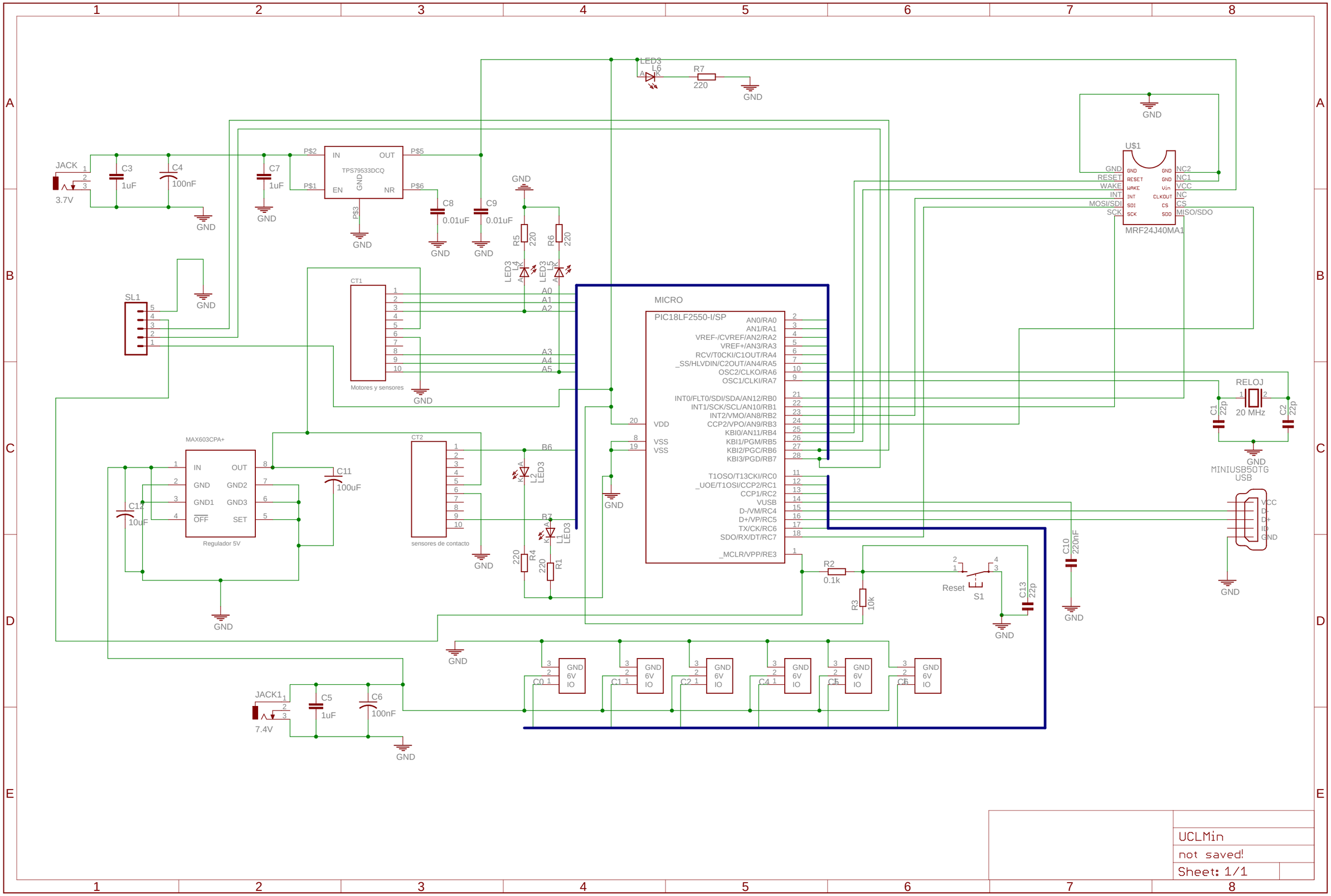


Figura A.1: Esquemático de la Sky293



UCLMin

not saved!

Sheet: 1/1

B

DESCRIPCIÓN DEL SOFTWARE UTILIZADO

B.1. Software utilizado

Para el desarrollo del presente proyecto se han necesitado una serie de programas informáticos que presentaremos en este anexo que es a lo que está destinado este apartado de la memoria. Los programas utilizados principalmente han sido tres, el primero para el diseño de las placas, el segundo para programación y el tercero para las definiciones del protocolo de comunicaciones:

1. El programa usado en el diseño de los circuitos electrónicos ha sido el EAGLE (*Easily Applicable Graphical Layout Editor*) de la marca CadSoft Computer, Inc, concretamente la versión 5.0.0 para Windows, un programa que no necesita licencia y fácil de manejar.

2. Para la programación del microcontrolador necesitamos haber grabado en el primeramente el bootloader del mismo. Este nos permite cargar programas que previamente hayamos escrito en C y convertido a lenguaje ensamblador. Para la programación de los pic hemos utilizado la herramienta gratuita que proporciona Microchip, el MPLAB en su versión actual.

3. También se ha utilizado el Analizador de redes ZENA, se utiliza para generar archivos de las definiciones necesarios para el desarrollo de las aplicaciones, en función de distintos factores que se verán en el apartado correspodiente.

B.1.1. Diseño de Printed Circuit Boards(PCB)

Como ya hemos mencionado con anterioridad el software elegido es EAGLE, a continuación se detallan unas nociones básicas para el uso del programa.

Arrancamos el programa y encontramos la siguiente ventana inicial:

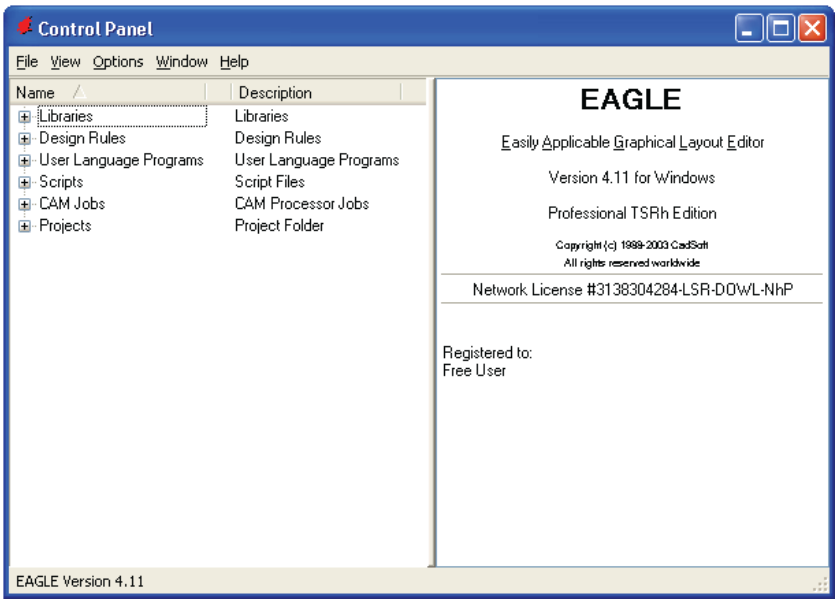


Figura B.1: Ventana de arranque de EAGLE

Vemos que en esta página inicial tenemos varias carpetas.

1. *Libraries* contiene los diseños de los componentes que se usan en el diseño de los PCB. Si no se encuentra el componente específico, se puede buscar la librería en Internet o crearla. Normalmente se buscará, el mismo programador del software tiene en su página web gran cantidad de librerías no incluidas inicialmente en el programa `http://www.cadsoftusa.com/download.htm`.

2. *Design Rules* donde se encuentran todas las normas de diseño que deben cumplirse para el diseño de la placa y su posible fabricación posterior.

3. *User Language Programs* en la que se engloban las sentencias que pueden utilizarse en el diseño de la placa a través de la línea de comandos que el programa posee.Normalmente no se utiliza, ya que es más sencillo hacerlo de manera gráfica con las barras de menús que el programa posee.

4. *Scripts* al igual que los *User Language Programs* es una poderosa herramienta que nos permite dar color a las distintas placas, importar datos o configuraciones de EAGLE con ayuda de estos archivos.

5. *CAM Jobs* nos proporciona los distintos archivos de salida necesarios para la fabricación física de la placa.

6. *Projects* contiene todos los proyectos guardados o guarda los proyectos creados. En vez de crear proyectos completos, se puede empezar por el esquema, y a partir de ahí hacer la placa si interesa o no. La función es la misma por cualquiera de estos dos métodos.

Para crear un proyecto, se ha de hacer *File/New/Project* sobre la carpeta en la que se desee crearlo. Después se selecciona el nombre que se le quiere poner y se confirma. Para crear los archivos que va a contener, se selecciona la carpeta, y una vez dentro de ella creamos el Schematic, ya que para el diseño de una placa, hay que empezar por el esquema y, una vez colocados todos los componentes, diseñar el archivo Board. Para crear el archivo en el nuevo proyecto se puede hacer con el botón derecho sobre la carpeta o bien desde *File* y en ambas, seleccionar *New/Schematic*.

Figura B.2: barra de menús de EAGLE

Al crear un esquema, la primera ventana muestra en la parte superior la barra de comandos y sobre ella las barras de menús y de acciones.

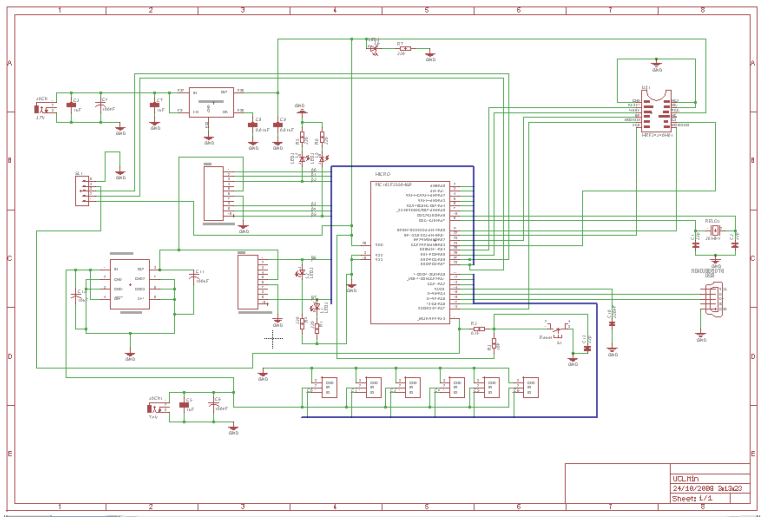


Figura B.3: Ventana del esquemático del programa EAGLE

Los botones que vamos a utilizar fundamentalmente de la barra de menús prin-

cipalmente serán los siguientes:

-  Lo primero es cambiar la rejilla para adecuarla al trabajo. Debemos tener cuidado ya que las medidas que usa por defecto son pulgadas. Se aconseja trabajar a unos *50 mil*, que son milipulgadas, ya que las medidas de los componentes están en pulgadas[1]. Ya que si se selecciona un número entero de *mm*, puede que no encajen las conexiones en los pines.

-  Este icono sirve para insertar un elemento de la librería (un componente). Se abre una lista en la cual buscamos y seleccionamos el dispositivo.

-  Se utiliza para borrar. Para ello se selecciona el objeto a borrar, teniendo en cuenta que si es un componente debemos marcarlo sobre la cruz[2].

-  Para unir los pines por una línea que en la placa sea una pista. Si al intentar unirlos no se acierta sobre el pin en cuestión, se puede deber a que se use una rejilla demasiado grande. Si dos cables se cruzan, no tienen por qué estar unidos. Para que estén unidos debe aparecer un punto en la intersección.

-  Para cambiar el nombre o el valor de un elemento, por ejemplo asignar el valor a una resistencia o ponerle un nombre a una conexión, seleccionando el que sea necesario y pulsando sobre el elemento en cuestión.

-  Una vez terminado el esquemático de la placa, lo pasamos al Board para poder colocar los componentes sobre la placa que se fabricará posteriormente

Una vez terminado el esquemático de la placa y generamos el Board aparecerá una ventana como la que se muestra en la Figura B.4 donde inicialmente tenemos todos los componentes alineados en un lateral del rectángulo unidos por líneas amarillas que indican las conexiones a realizar.

Como en el esquemático debemos tener en cuenta la rejilla, utilizar el mismo valor durante todo el trabajo ya que si no puede darnos problemas en el ruteado de las pistas. A continuación se colocarán los componentes en su lugar, procediéndose después al ruteado de pistas. Hay que tener cuidado a la hora de mover los

[1] $1 pulgada = 2{,}54 cm$

[2] Esta cruz sólo aparecerá si tenemos activas las capas tOrigins y bOrigins

componentes, ya que si se utiliza un ratón con ruleta, al pulsar ésta, el componente cambia de cara. Si esto ocurre y es por causa accidental puede suponer un problema si es un circuito integrado, ya que cambiaría la disposición de los pines. Si es una resistencia, no importaría, pues al tener sólo 2 pines, daría igual.

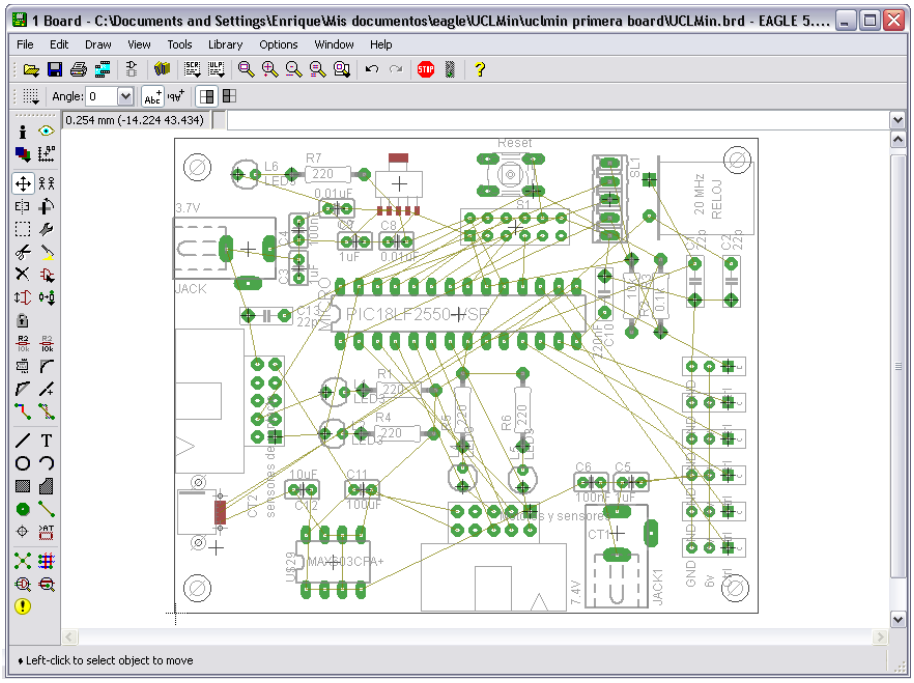


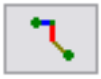
Figura B.4: Ventana del board del programa EAGLE

Al igual que al crear el esquemático, vamos a estudiar los botones más significativos en la board.

-  Se utiliza para mover componentes y situarlos en el lugar que queramos en la placa.

-  Para seleccionar varios elementos a la vez se rodean con el cuadro que aparece tras pulsar el icono. Para moverlos se utilizar el botón antes mencionado. Y para que se mueva el grupo presionamos el botón derecho del ratón y elegimos la acción *move group* en el menú que aparece.

-  Tras tener los elementos colocados como se desee, se puede utilizar esta herramienta. Ésta nos genera las líneas amarillas de las conexiones de nuevo marcando los componentes más cercanos.

-  Se usa para rutear las pistas una vez que tenemos los elementos colocados en la placa.Esta herramienta se llama ratsnest. Para ello elegimos el origen de la pista y nos indica donde debemos unirla. Al utilizar esta herramienta nos aparece la barra de ruteado que se divide en dos partes:
  -  La herramienta seleccionada es la que se encuentra hundida. La forma de la pista dependerá de la opción que tengamos seleccionada. Para cambiar de herramienta mientras se está colocando una pista puede hacerse seleccionándola o pulsando el botón derecho, modo en el cual se irán recorriendo todas las herramientas hasta seleccionar la deseada.
  - 

En esta parte de la barra se muestra información sobre el tamaño de la pista , la forma del agujero del cambio de cara y su tamaño. Para cambiar de capa se hace pulsando el botón derecho mientras se está haciendo una ruta y seleccionando la capa deseada. Las capas normales de ruteado son TOP (roja) y BOTTOM (azul). El resto se podrían usar en sistemas de creado multicapa.

-  Una vez elaborado todo el trabajo y obtenido el diseño definitivo, se pasará el DRC (Check Design Rules) que nos dirá si hay errores en el trabajo realizado. Después debemos cargar la norma a utilizar. En este proyecto vamos a utilizar la norma 2ci-clase3

-  Una vez pasada la norma sin errores, se hará el plano de masas. Para ello, en nuestro caso se seleccionaba la norma 32milsep2. Para ello se selecciona esta herramienta creando un polígono alrededor de las dimensiones de la placa. Antes de cerrar este polígono, escribir en la línea de comando el nombre de la señal utilizada como masa en el esquemático (por ejemplo GND o CGND) y cerrar el polígono. Ahora se ha de hacer lo mismo con la otra capa si es que se han desarrollado pistas por las dos capas. Si es una monocapa, sólo hará falta hacer el plano de masas por una de sus caras. Tras esto, aparecerán dos rectángulos de los colores de las capas utilizadas (en este caso TOP y BOTTOM).

Si se selecciona ahora la herramienta ya utilizada ratsnest muestra el relleno que hace el plano de masas.

Una vez terminado este trabajo, es la hora de mandar a que hagan las placas físicamente. Para ello, es necesario mandar los archivos GERBER. Estos archivos

los hace de forma automática, pero es necesario ejecutar unos comandos. Para ello, en la barra de comandos de la placa, se ejecuta *run drillcfg*, con lo que se configuran los agujeros, guardándolo con un archivo de extensión *.drl*. Una vez configurados los agujeros se ejecuta el subprograma CAM donde se ejecuta el trabajo *excellon.cam* y *gerb274.cam*, creándose los archivos necesarios para crear la placa física.

B.1.2. Software para la programación

El software empleado para la programación es el MPLAB IDE. Esta herramienta necesita un hardware para su comunicación con el microcontrolador para su programación. Este hardware en el caso que nos aborda es el MPLAB ICD 2. En este apartado se describirán ambos componentes.

El MPLAB IDE es una herramienta gratuita para el desarrollo de aplicaciones que usan microcontroladores PICs de Microchip. Funciona en el entorno de Windows, es fácil de manejar e incluye gran cantidad de componentes software gratuitos para un rápido desarrollo de aplicación y depurado. También sirve como una interfaz gráfica para software externos compatibles y herramientas hardware de desarrollo. El movimiento entre herramientas se hace de forma rápida y es fácil de actualizar.

Como aquí se ha comentado, en el caso que nos ocupa, se ha tenido que instalar un software auxiliar para la programación, ya que el MPLAB IDE es sólo la interfaz del hardware programador, pero no entiende el lenguaje en que se escribe. Así, se ha tenido que instalar un compilador en C (que ha sido el lenguaje de programación escogido por su sencillez y universalidad). Este compilador ha sido el MCC18 suministrado por Microchip de manera gratuita[3]. El compilador es un programa que permite traducir el código fuente de un programa, normalmente en lenguaje de alto nivel, a otro lenguaje de nivel inferior, normalmente lenguaje máquina. Esta traducción de un lenguaje a otro se denomina compilación.

Una vez instalados ambos programas, es necesario instalar una interfaz que los comunica, para que al escribir en el editor que presenta el MPLAB, éste sea capaz de identificar los comandos.

La conexión entre el MPLAB IDE y el programador MPLAB ICD 2 se hace por medio de un cable USB que hace que se alimente el programador.

[3]Microchip lo proporciona de manera gratuita durante 60 días.

Figura B.5: Programador MPLAB ICD 2

La salida del programador hacia el microcontrolador se produce por una clavija telefónica que, en el lado del microcontrolador, dependerá de la clavija de conexión a utilizar. Nosotros en la placa hemos colocado un conector rápido para la programación.

Para la elaboración de un programa, el funcionamiento es sencillo. Se ejecuta el programa del MPLAB IDE, en este caso la versión 8.36 de este programa. Una vez iniciado, se le da a *Project/Project Wizard*, que es como un programa ayuda para proporcionarle la información necesaria inicial del programa. A continuación se nos pedirá seleccionar el *device*, que es el microcontrolador que utiliza la placa a programar. Una vez dada esa información, nos pedirá que le demos la ubicación donde creará el programa, después si se desea incluir algún fichero base para modificar sobre él. Si se selecciona algún fichero para introducirlo como base, es conveniente activar la casilla correspondiente para copiar el archivo, ya que si no, sobrescribiremos la información en el fichero base, modificándolo. Cuando se termina de crear el proyecto, nos aparece una ventana con el nombre de nuestro archivo.mcw, que es la ventana de información de nuestro proyecto. Ahí, conforme se vaya avanzando en el proyecto, aparecerá nueva información. Si damos doble click sobre el archivo fuente que hemos creado, nos abrirá una ventana de un editor de texto para poder comenzar nuestro programa.

Para ello, lo primero es cargar el archivo de librería del microcontrolador en cuestión, y a continuación se añadirá también el archivo que contiene las órdenes básicas. Si se necesita otro archivo de órdenes específicas, se cargará a continuación de éste. Por último, vendrán una serie de definiciones y usos, que tendrán que ponerse en función de la placa a programar.

A continuación vendrá lo que es el programa en sí, que dependerá del usuario. Una vez terminado el programa, es hora de compilarlo y, si no tiene errores, pro-

gramarlo o depurarlo, según se prefiera. Programarlo implica que el programa se introduce en el microcontrolador y funciona automáticamente; depurarlo implica que el programa se carga en el microcontrolador, pero el usuario maneja el programa desde la ventana del MPLAB IDE.

B.1.3. Analizador de redes ZENA

Este apartado presenta el analizador de redes inalámbricas[4] ZENA, y describe brevemente sus capacidades. El analizador ZENA ofrece tres herramientas principales para desarrollar soluciones de forma rápida y eficiente con las pilas ZigBee y las propias de Microchip MiWi y MiWi P2P basados en la IEEE 802.15.4.

El analizador ZENA permite a los desarrolladores modificar rápidamente los programas de ejemplo y modificar las pilas para adaptarse a los requisitos de la aplicación.

Utilizando el ZENA podemos visualizar el envío y recepción de los paquetes por el protocolo elegido.

También podemos ver la topología de la red que se forma entre los distintos nodos de la red. Estas herramientas, combinadas, forman una poderosa herramienta en el desarrollo inalámbrico para el protocolo IEEE 802.15.4.

Tras instalar ZENA abrimos la pantalla del programa y sale la ventana que se observa en la Figura B.6

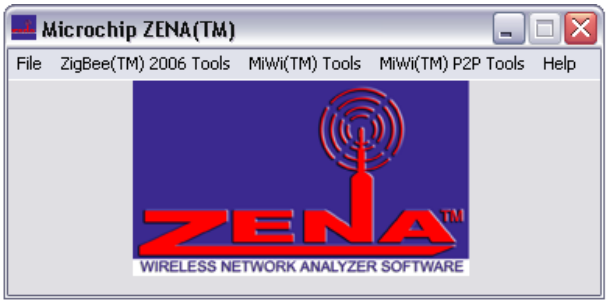


Figura B.6: Pantalla de inicio del analizador ZENA

Una vez que nos encontramos dentro del programa, vemos que es una interfaz bastante sencilla de utilizar. Hay 5 opciones:

- File
- ZigBee(TM) 2006 Tools

[4] de hardware y software

- MiWi(TM) Tools
- MiWi(TM) P2P Tools
- Help

B.1.3.1. File

Esta opción lo único que nos permite es salir del programa, cerrarlo.

B.1.3.2. ZigBee(TM) 2006 Tools

Vamos a desarrollar esta opción, ya que es la más compleja y por tanto la más completa, para los otros dos protocolos el desarrollo es el mismo, incluso más sencillo.

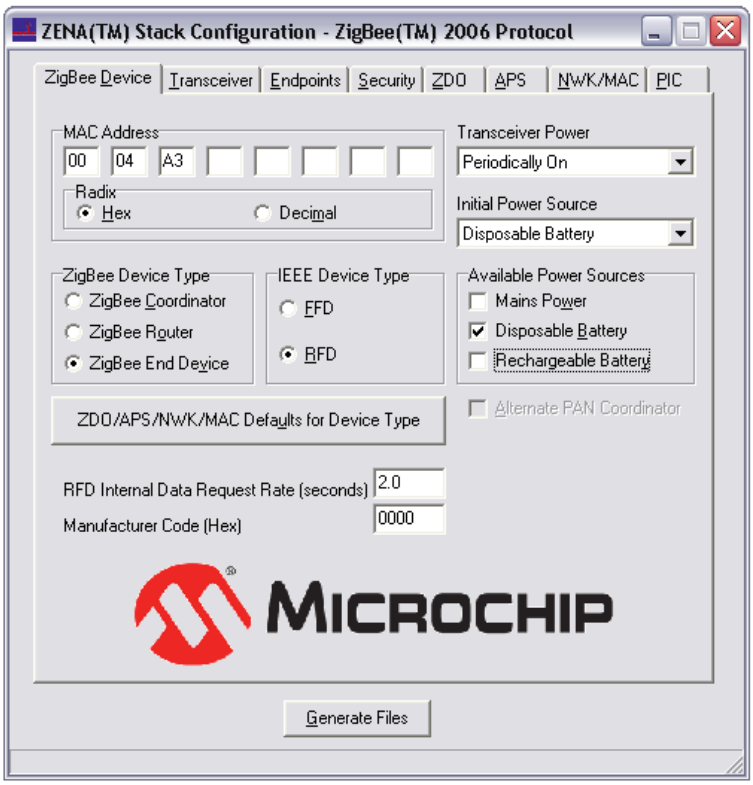


Figura B.7: Pantalla inicial del protocolo ZigBee

Dentro de la opción de ZigBee(TM) 2006 Tools se nos presentan dos opciones:

- Stack Configuration: En este apartado se nos permite modificar la configuración para la aplicación.

  En la primera pestaña (Figura B.7) podemos variar del nodo todo lo que nos interese:

  - La MAC Address del dispositivo y si la queremos escribir en decimal o hexadecimal.
  - El tipo de nodo que es el dispositivo. Tenemos tres opciones:
    - ZigBee Coordinator o Coordinador ZigBee
    - ZigBee Router o Router ZigBee
    - ZigBee End Device o Terminal ZigBee
  - Tipo de dispositivo del IEEE, ya que algunos dispositivos del protocolo ZigBee tienen la opción de seleccionarlo, en función de la aplicación que nosotros vayamos a desarrollar.
  - ZDO/APS/NWK/MAC[5] que se pondrán de manera automática al cambiar el tipo de dispositivo.
  - Alimentación del Transceptor que podrá ser:
    - Always On, es decir, continuamente.
    - Periodically On, es decir, de manera periódica.
    - On when stimulated, es decir, cuando sea estimulado.
  - Recurso de alimentación inicial de nuestra aplicación:
    - Disposable Battery, o pilas desechables.
    - Rechargeable Battery, o baterías recargable.
    - Mains Power, o alimentación de la red.
  - También nos permite describir los tipos de recursos de alimentación que la aplicación tiene.

  En la segunda pestaña (Figura B.8) se tienen todos los datos del transceptor:

[5] Características del dispositivo

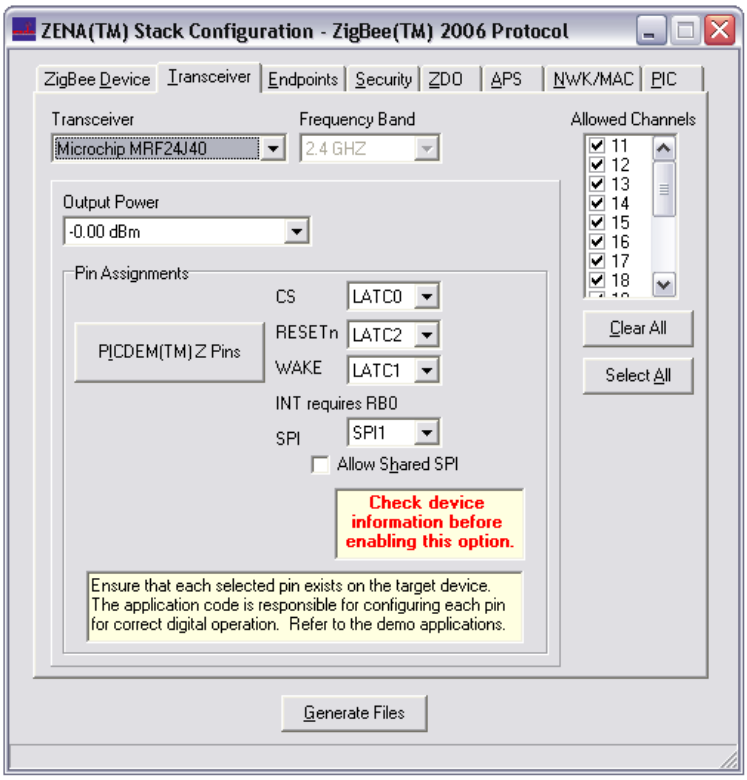


Figura B.8: Pestaña del transceiver en el ZENA

- El modelo de transceptor en nuestro caso el MRF24J40MA. La banda de frecuencia en la que trabaja que son 2.4 GHZ por defecto.
- El ruido de salida, en la que pondremos 0.00 dBm
- Los canales permitidos de conexión.
- La asignación de pines para las distintas conexiones del módulo de RF. Con la opción de poner por defecto los pines del PICDEM Z.

En la tercera pestaña se nos permite agregar archivos de cabeceras para los terminales.

La cuarta pestaña es la que se encarga de la seguridad que en nuestro caso no es necesaria, por lo que se obviará.

Al igual que la quinta, sexta y séptima pestaña pues son propias del protocolo ZigBee y nosotros utilizaremos MiWi.

Por último la octava pestaña (Figura B.9) que nos permite decidir cual será nuestro dispositivo, la velocidad del reloj y de la conexión UART, el programador, el tamaño de la memoria del dispositivo, y el tamaño de la memoria para el almacenamiento del programa.

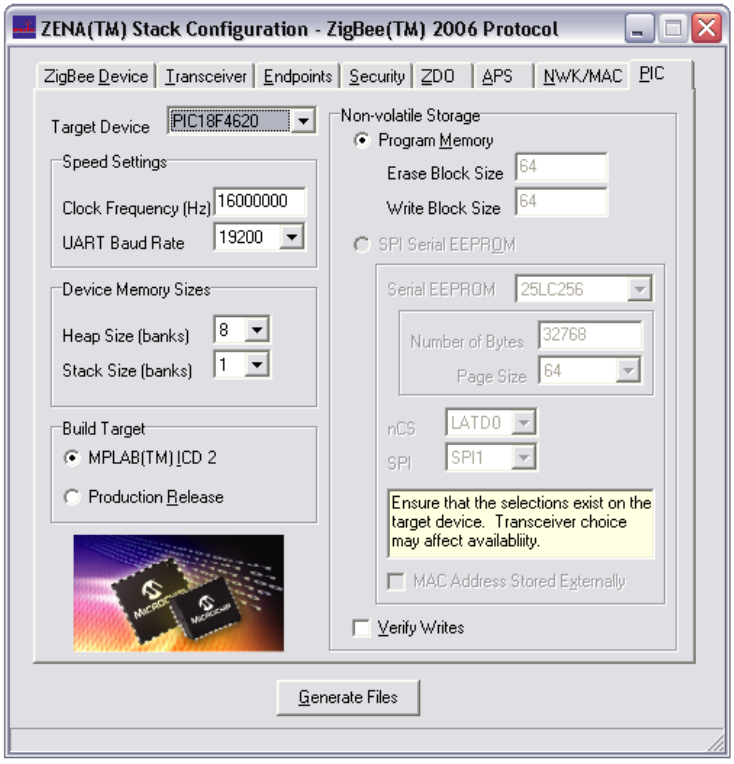


Figura B.9: Pestaña del PIC en el ZENA

- Network Traffic Monitor. La monitorización de envío de paquetes queda para líneas de investigación futuras ya que para el control de la comunicación de un dispositivo[6] con el pc es sencilla y no se necesita para ver el tipo de red ni el envío de paquete de datos. Cuando se comuniquen varios robots para trabajos cooperativos, la monitorización será fundamental.

B.1.3.3. MiWi(TM) Tools

Los pasos a seguir son los mismos, únicamente que este apartado está dedicado al MiWi y el anterior al ZiGBee, pero trabajar con este protocolo es similar al anterior.

Vamos, para que sirva de ejemplo, a crear el archivo MiWiDefs de nuestro robot. Se introducen los datos del nodo como ya se ha explicado, MAC, tipo de dispositivo (RFD), el transceptor a utilizar (MRF24J40MA), que estamos trabajando en una custom board y los pines en los que tenemos colocados CS, RESET y WAKE en la placa, el modelo de PIC a utilizar no necesitamos introducir más datos, la velocidad del reloj y la conexión con el PC en el caso de que existiera.

[6] Robot UCLMin

Con estos valores introducidos generamos el archivo.

B.1.3.4. MiWi(TM) P2P Tools

Los pasos a seguir son los mismos, únicamente que este apartado está dedicado al MiWi P2P y el anterior al ZiGBee, pero trabajar con este protocolo es similar al anterior.

B.1.3.5. Help

En esta opción podemos ver la licencia del ZENA y las características de la versión, el copyright, etc.

C

COMPARATIVA DE MICROCONTROLADORES

C.1. Microcontrolador

C.1.1. Definición

Un microcontrolador es un circuito integrado que contiene un pequeño ordenador: CPU, memoria y buses de instrucciones y datos. Además, dispone de una serie de recursos internos, muchos de ellos mapeados directamente en memoria, que son los que lo hacen útil para la realización y control de determinadas tareas. Los recursos que más nos interesan en este proyecto son de lo más habitual en la mayor ya de los microcontroladores de uso general:

- **Puertos de E/S**. Un microcontrolador dispone de una serie patillas organizadas en puertos de varios bits. Cada patilla de un puerto representa un bit. Por ejemplo, un puerto de ocho bits dispone de ocho patillas. Cada patilla puede ser configurada como entrada o como salida. Si está configurada como entrada, entonces en función del valor de tensión que se le esté aplicando, el microcontrolador almacena un "0" o un "1" en la zona de memoria correspondiente a la patilla del puerto en cuestión. Análogamente, si se configura como salida, el microcontrolador establecerá en la patilla correspondiente del puerto un valor de tensión alto o bajo en función de si en la zona de memoria correspondiente hay almacenado un "0" o un "1".

  Este sistema permite controlar diversos dispositivos: desde memorias a un simple led. La corriente proporcionada por el microcontrolador no es muy

grande, de modo que muchas veces hay que amplificarla si se quiere actuar directamente sobre dispositivos de mucho consumo.

- **Conversores A/D**. Los conversores analógico-digital permiten al microcontrolador muestrear o generar señales. Las utilidades son muchas: leer el valor de tensión a la salida de un potenciómetro, medir la carga de las baterías, etc.

- **Contadores y temporizadores**. Gracias a ellos, el microcontrolador puede contar los eventos (cambios de estado en una patilla) producidos en la patilla asociada al contador o bien medir el tiempo transcurrido entre dos o más eventos. Generalmente, los contadores tienen una capacidad de entre 8 y 16 bits, de forma que el número de eventos o el lapso de tiempo que pueden considerar es limitado. Cuando la capacidad del contador se excede, se produce una interrupción que le indica al microcontrolador lo que ha ocurrido. El programador decide si ignorar o no dicha interrupción.

- **Interrupciones externas**. En algunos microcontroladores es posible detener la ejecución del programa actual para ejecutar una cierta tarea en el momento de recibirse un determinado evento externo. Una vez ejecutada esa rutina, el microcontrolador sigue con la ejecución de la tarea interrumpida.

Otros recursos muy importantes para este proyecto, pero menos presentes en los microcontroladores son los siguientes:

- **Módulos de comunicaciones**. Para poder comunicarse con otros dispositivos, además de los puertos de E/S los microcontroladores disponen de una serie de módulos de comunicaciones que le permiten conectarse a distintos buses o canales de comunicaciones estándar: RS-232, USB, I2C, etc. Algunos de estos estándares, por ejemplo RS-232, pueden implementarse por software, usando un puerto de I/O, si el microcontrolador no dispone del hardware necesario.

- **Módulos PWM**. Permiten la generación de señales cuadradas de frecuencia y ciclo de trabajo variable. Este tipo de señales es ampliamente usado para el control de la velocidad de los motores DC.

- **Memoria EEPROM**[1]. Algunos microcontroladores tienen integrada una memoria EEPROM, muy útil para el almacenamiento de variables de configuración, ya que estas memorias no se borran aunque haya un corte de tensión. Además, pueden ser reprogramadas en cualquier momento para poder actualizar los valores almacenados.

Es muy frecuente que existan diversas versiones del mismo modelo de microcontrolador. Internamente son prácticamente iguales, pero varían el número de patillas (y por tanto el número de puertos y recursos disponibles) y la cantidad de memoria

[1] *Electrically Erasable Programmable ROM*, o ROM regrabable y programable electricamente.

o contadores disponibles. En función de los requerimientos de la aplicación, se opta por una versión u otra.

La limitación en la aplicación de los microcontroladores a un desarrollo de ingeniería tiene su límite en la imaginación del desarrollador.

C.1.2. Basic X24P,SR1

Este controlador de la marca Atmel está considerado como el más potente del mercado. El Basic X24 reúne en un circuito del tamaño de un chip de 24 patas a un microcontrolador Atmel, una memoria EEPROM de 32Kbytes, un regulador de tensión y un puerto RS232(serie).

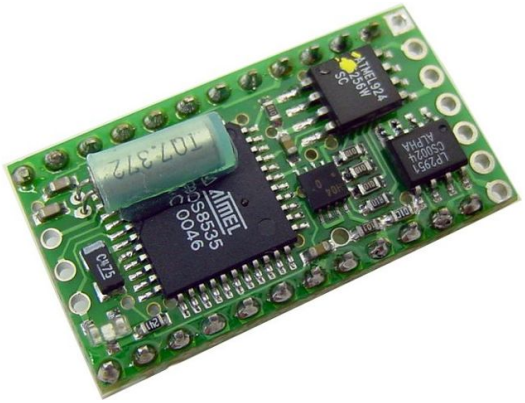


Figura C.1: Microcontrolador del robot SR1

La ventaja de este microcontrolador es que tiene un lenguaje de programación de alto nivel, ya que los programas que utiliza se escriben en Basic y más tarde se vuelcan en el micro haciendo que el robot sea totalmente autónomo Otra ventaja que tiene es que puede alimentarse con una tensión de entre 5 y 24 V utilizando el propio regular que lleva o una fuente regulada entre 3 y 5.5 V.

Este microcontrolador es el utilizado por el SR1 (Figura 2.12).

C.1.3. Atmel ATMega 128,Bioloid

Es un microcontrolador de 8 bits CMOS basado en el AVR basado en la arquitectura RISC. El AVR es una CPU de arquitectura Harvard. Tiene 32 registros de 8 bits. Algunas instrucciones sólo operan en un subconjunto de estos registros. La concatenación de los 32 registros, los registros de entrada/salida y la memoria de datos conforman un espacio de direcciones unificado, al cual se accede a través de operaciones de carga/almacenamiento.Estos 32 registros están conectados a la

Unidad Aritmética Lógica (ALU) ,permitiendo a dos etapas (carga y ejecución) independientes tener acceso de manera simultánea en un único ciclo de reloj. Esto lo hace ser mucho mas rápido. A diferencia de los microcontroladores PIC, el stack se ubica en este espacio de memoria unificado, y no está limitado a un tamaño fijo.

Un ejemplo de robot que lleva este micro sería el Bioloid (Figura 2.13).

C.1.4. Familia PIC

Los 'PIC' son una familia de microcontroladores tipo RISC[2] fabricados por Microchip Technology Inc. y derivados del PIC1650, originalmente desarrollado por la división de microelectrónica de General Instruments.

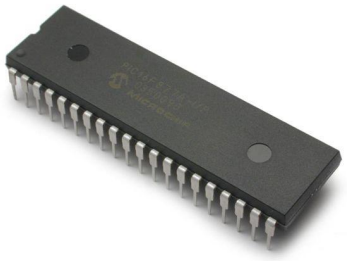


Figura C.2: Microcontrolador PIC

El nombre actual es un acrónimo. En realidad, el nombre completo es PICmicro, aunque generalmente se utiliza como Peripheral Interface Controller (Controlador de Interfaz Periférico).

El PIC usa un juego de instrucciones tipo RISC, cuyo número puede variar desde 35 para PICs de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada sleep.

Para transferir el código de un ordenador al PIC normalmente se usa un dispositivo llamado programador. La mayoría de PICs que Microchip distribuye hoy en día incorporan ICSP (In Circuit Serial Programming, programación serie incorporada) o LVP (Low Voltage Programming, programación a bajo voltaje), lo que permite programar el PIC directamente en el circuito destino. Para la ICSP se usan los pines RB6 y RB7 como reloj y datos y el MCLR para activar el modo programación aplicando un voltaje de 13 voltios.

[2]del inglés Reduced Instruction Set Computer, Computadora con Conjunto de Instrucciones Reducidas.

El tamaño de palabra de los microcontroladores PIC es fuente de muchas confusiones. Todos los PICs (excepto los dsPIC) manejan datos en trozos de 8 bits, con lo que se deberían llamar microcontroladores de 8 bits. Pero a diferencia de la mayoría de UCPs(unidad central de procesos), el PIC usa arquitectura Harvard[3], por lo que el tamaño de las instrucciones puede ser distinto del de la palabra de datos. De hecho, las diferentes familias de PICs usan tamaños de instrucción distintos, lo que hace difícil comparar el tamaño del código del PIC con el de otros microcontroladores.

- PIC16F876A, utilizado en el robot skybot. Gracias a la plataforma desarrollada por el grupo iearobotics tenemos mucha información y muchos programas desarrollados. Además es una plataforma sobre la cual se sigue investigando.

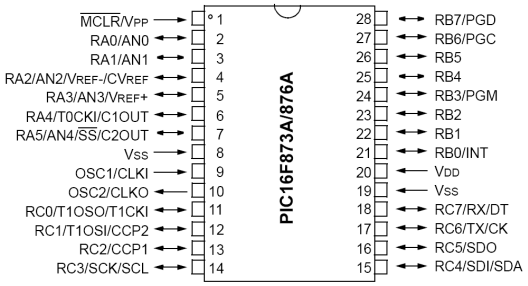


Figura C.3: PIC16F876A

- PIC18F2455, en este mismo proyecto de Miguel Domingo García Jiménez se desarrolló una aplicación. Esta aplicación que se desarrolló consistía en un AGV(Automatic Guided Vehicle). Al ser una aplicación mucho más sencilla y que requiere muchas menos prestaciones se sustituyó el 18F4580 por uno de la misma familia pero más sencillo. Se utilizó el PIC18F2455.

[3]originalmente se refería a las arquitecturas de computadoras que utilizaban dispositivos de almacenamiento físicamente separados para las instrucciones y para los datos (en oposición a la Arquitectura de von Neumann). El término proviene de la computadora Harvard Mark I, que almacenaba las instrucciones en cintas perforadas y los datos en interruptores.

Todas las computadoras constan principalmente de dos partes, la CPU que procesa los datos, y la memoria que guarda los datos. Cuando hablamos de memoria manejamos dos parámetros, los datos en sí, y el lugar donde se encuentran almacenados (o dirección).

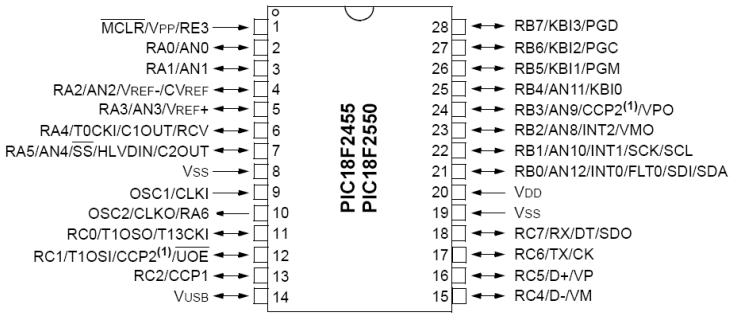


Figura C.4: PIC18F2455

- PIC18F4580, utilizado en las placas del sistema controlador para los nuevos puestos de la asignatura de automatización desarrollados por Miguel Domingo García Jiménez en su proyecto final de carrera. Es un pic mucho más potente que los otros dos ya que es capaz de controlar una placa con un alto número de entradas y salidas tanto analógicas como digitales.

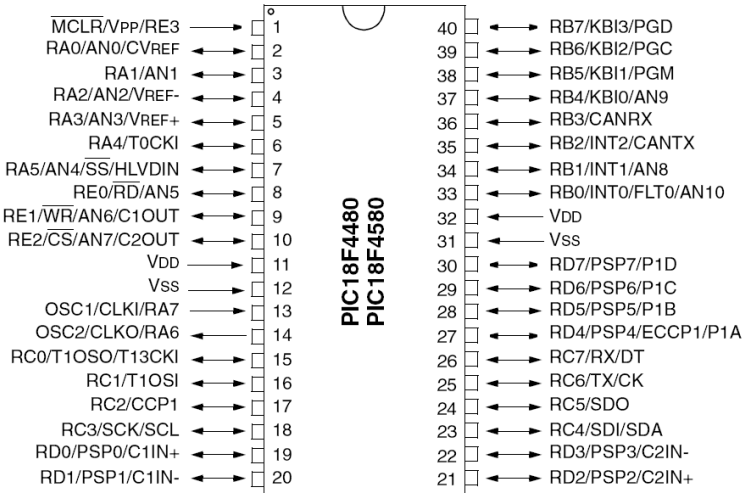


Figura C.5: PIC18F4580

- El último pic que vamos a estudiar es el 18f2550. Este pic lo vamos a estudiar debido a que mantiene la colocación de los pines del 16F876A con varias ventajas:

  - Tiene más memoria que el equivalente al de la familia de los 16F.
  - Incorpora en su nueva versión un conversor USB-Serial, por lo que permite conectar el micro al pc por usb. Esto permite una conexión más

actual que a través del puerto serie, el cual utilizaban todos los anteriores.

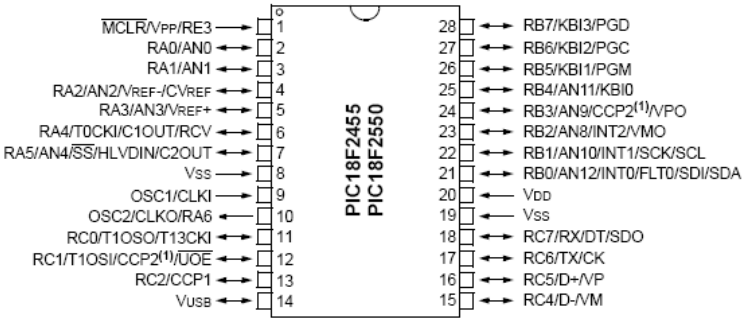


Figura C.6: PIC18F2550

C.2. Comparativa de los microcontroladores

En la Tabla C.1 se ha hecho a modo de resumen una comparativa con los micros estudiados para ver las diferencias entre sus características principales. Debemos estudiar principalmente su velocidad para adquisición de datos, el número de instrucciones que soportan, que el número de entradas y salidas sea el suficiente para poder montar en el mismo todo lo que queremos que nuestro robot lleve, y sobre todo el tamaño de la memoria del mismo. El tamaño es el punto más importante del micro a elegir ya que el módulo inalámbrico necesita gran parte de esta para el envío y recepción de datos, debido a que el envío no tiene por que ser inmediato y la almacenara en un buffer que se encontrará en la memoria del micro. De esta manera queda descartado automáticamente el PIC16F876A (utlizado por el skybot) y el PIC18F2455 ya que aunque nos da el número de salidas y entradas y el de instrucciones suficientes como para hacer un pequeño robot, no es capaz de almacenar la información del módulo de radio frecuencia.

Buscamos que el robot sea barato y sencillo al montaje y la programación para futuras investigaciones por lo que automáticamente quedan descartados el Atmel y el Basic debido a su alto precio y a que hay que programarlo en lenguajes más complicados que C, respectivamente.

Sólo nos quedan los tres pic de la familia de los 18F, tampoco queremos que nos sobre micro ya que con un tamaño de 28 pines tenemos salidas más que suficientes para montar con garantías nuestro pequeño robot sin necesidad de acceder a un micro de 40 pines, por lo que el PIC18F4580 queda descartado.

Por lo que el elegido para el *UCLMin* es el PIC18F2550

	PIC16F876A	PIC18F4580	PIC18F2455	PIC18F2550	Atmel ATMega 128	Basic X24P
Tipo de memoria programable	Flash	Flash	Flash	Flash	Flash	EEPROM
Tamaño de memoria programable(KB)	14	32	24	32	128	
Velocidad CPU (MIPS)	5	10	12	12	+16	83,000
Bytes de RAM	368	1,536	2,048	2,048	4K[a]	400
Datos EEPROM (bytes)	256	256	256	256	4K Bytes	32K bytes
Comunicación digital con periféricos	1-A/E/USART[b], 1-MSSP(SPI/I2C)	1-A/E/USART, 1-MSSP(SPI/I2C)	1-A/E/USART, 1-MSSP(SPI/I2C)	1-A/E/USART, 1-MSSP(SPI/I2C)	2 USART	
Captura/Comparación/PWM[c] Periféricos	2 CCP	1 CCP, 1 ECCP	2 CCP	2 CCP	6CCP	
Temporizadores	2 x 8-bit, 1 x 16-bit	1 x 8-bit, 3 x 16-bit	1 x 8-bit, 3 x 16-bit	1 x 8-bit, 3 x 16-bit		
ADC[d]	5 ch, 10-bit	11 ch, 10-bit	10 ch, 10-bit	10 ch, 10-bit	8 ch, 10 bit	8 ch, 10 bit
Comparadores	2	2	2	2		
USB (canales, velocidad, versión)	-	-	1, Full Speed, USB 2.0	1, Full Speed, USB 2.0	-	-
CAN	-	1 ECAN	-	-		
Rango de temperatura (C)	-40 a 125	-40 a 125	-40 a 85	-40 a 85	-40 a +85 ^{o}C	-40 a +85 ^{o}C
Rango de voltaje de operaciones (V)	2 to 5.5	2 - 5.5	2 - 5.5	2 - 5.5	4.5 - 5.5V	3 y 5.5V
Númmero de pines	28	40	28	28	64	24
Frecuencia de operación (MHz)	20	40	48	48	16	
Número de entradas y salidas	22	16[e]	25	16	19	21[f]

Tabla C.1: Comparativa de los microcontroladores

[a] SRAM
[b] Universal Synchronous Asynchronous Receiver
[c] Power-Managed Modes
[d] conversor analógico-digital
[e] bidireccionales
[f] 16 Normales + 2 P. Serie + 3 Disponible fuera del área de los pines normales

D

COMPARATIVA DE COMUNICACIONES INALÁMBRICAS

D.1. Introducción

La comunicación inalámbrica[1] es un tipo de comunicación,relativamente nuevo, en el cual no se utiliza un medio de propagación físico, sino la modulación de ondas electromagnéticas. Estas ondas se propagan por el espacio sin un medio físico que comunique cada uno de los extremos de la transmisión. En ese sentido, los dispositivos físicos sólo están presentes en los emisores y receptores de la señal. Esta transferencia de información nos permite tener dos grandes ventajas las cuales son la movilidad y flexibilidad del sistema en general. Vamos en este anexo a estudiar distintos módulos de emisión y recepción de datos.Centraremos el estudio en 4 tipos de comunicación principalmente:

- ZigBee
- WiFi
- Bluetooth
- GPRS

[1] en inglés wireless, sin cables

D.1.1. Reseña histórica de la comunicación

Las primeras manifestaciones en la comunicación de la especie humana fue la voz, la invención del fuego y las pinturas rupestres; posteriormente al evolucionar, fue apareciendo la escritura.Éste es el elemento que permitió desarrollar las culturas que hoy en día se conocen. Con el desarrollo de las civilizaciones y de las lenguas escritas surgió también la necesidad de comunicarse a distancia de forma regular, con el fin de facilitar el comercio entre las diferentes naciones e imperios.

Las antiguas civilizaciones utilizaban para comunicarse a mensajeros, palomas, etc. con el comienzo de la utilización de la rueda,todo lo utilizado hasta el momento como sistemas de comunicación casi desapareció. Todo se hizo más sencillo y a una mayor velocidad. Aún así hasta el siglo XVIII las comunicaciones sufrieron un estancamiento.

Gracias a Benjamin Franklin que, en 1752, demostró que los rayos son chispas eléctricas gigantescas, se descubrió la electricidad. Samuel F.B. Morse en 1836 creó lo que hoy conocemos Telégrafo. Thomas Alva Edison, en 1874, desarrolló la telegrafía cuádruple, la cual permitía transmitir dos mensajes simultáneamente en ambos sentidos.

A pesar de este gran avance, no era suficiente lo que se lograba comunicar, y se requería de algún medio para la comunicación de la voz. Ante esto, surge el teléfono, inventado por Alexander Graham Bell, que logra la primera transmisión de la voz en 1876.

Así los primeros sistemas telegráficos y telefónicos utilizaban cable para lograr la transmisión de mensajes. Con los avances en el estudio de la electricidad, el físico alemán Heinrich Hertz descubre, en 1887 las ondas electromagnéticas, estableciendo las bases para la telegrafía sin hilos.

Pero no fue hasta el siglo XX cuando se inventaron los tubos de vacío y aparece la electrónica, fueron fundamentales para los avances posteriores. Gracias a estos avances se consigue por primera vez transmitir por radio en 1906 en Estados Unidos, 19 años después ya existían 600 emisoras de radio en todo el mundo.

Por otro lado, el físico francés Nicéphore Niepce utilizando una plancha metálica recubierta de betún, expuesta durante ocho horas, consiguió la primera fotografía. Perfeccionando este procedimiento, el pintor e inventor francés Louis Jacques Mandé Daguerre descubrió un proceso químico de revelado que permitía tiempos de exposición mucho menores.

En el siglo XIX, se desarrolla este invento hasta llegar al cinetoscopio, presentado por Tomas Edison en 1889 el cual patentó en 1891. Los hermanos Lumière, presentan y patentan el cinematógrafo en el año de 1895. Hasta que en el año de 1920 se le añade el sonido. Creando así, el cine.

La transmisión de imágenes a distancia esta ligada a varios avances e inven-

tos, como el disco perforado explorador, inventado en 1884 por el pionero de la televisión, el alemán Paul Gottlieb Nipkow. Otros de los hechos en el desarrollo de la televisión son el iconoscopio y el cinescopio, para transmitir y recibir, respectivamente, imágenes a distancia, inventados ambos en 1923 por el ingeniero electrónico ruso Vladimir Kosma Zworykin. Logrando con esto una de las más grandes industrias a escala mundial, lo que hoy conocemos como cadenas de televisión.

Con todos estos avances tecnológicos y necesidades, la comunicación o transmisión de datos fue tomando cada vez más auge. Los primeros intentos y realizaciones en la tarea de conjugar ambas disciplinas - comunicaciones y procesamiento de datos - tuvieron lugar en Estados Unidos, donde durante años cuarenta del siglo XX se desarrollo una aplicación para la U.S. Army y posteriormente, en 1953, otra para la gestión y reserva de las plazas en la American Airlines, que constituyeron los dos primeros sistemas de procesamiento de datos a distancia.

En la actualidad hay satélites de comunicaciones, navegación, militares, meteorológicos, de estudio de recursos terrestres y científicos. La mayor parte de ellos son satélites utilizados para la comunicación telefónica y la transmisión de datos digitales e imágenes de televisión.

Todo este desarrollo de las comunicaciones dio lugar a un nuevo concepto; telecomunicación, que significa çonjunto de medios de comunicación a distancia o transmisión de palabras, sonidos, imágenes o datos en forma de impulsos o señales electrónicas o electromagnéticas".

D.1.2. Situación actual

Los humanos por naturaleza necesitamos desenvolvernos en situaciones en las cuales se requiere comunicación. Uno de los medios más discutidos es la comunicación inalámbrica. Ésta, que se da a través de ondas electromagnéticas, facilita la operación en donde el elemento a comunicar no puede permanecer en un solo lugar (almacenes, oficinas, etc.). Por este motivo se están realizando en este campo amplias investigaciones, ya que es el futuro de las comunicaciones. Aún así, cabe también mencionar que las redes cableadas tienen una mayor ventaja en transmisión de datos que las inalámbricas. Mientras que las cableadas proporcionan velocidades de hasta 20Mbps (con posibilidad de crecimiento), las inalámbricas alcanzan hasta 11 Mbps (a 2.4 GHz con el standard 802.11g, con una capacidad de crecimiento menor). Por eso, se busca una mezcla entre las redes inalámbricas y el cable, de manera que el sistema cableado sea la parte principal y la inalámbrica sea la que le proporcione movilidad al equipo y al operador para desplazarse con facilidad.

D.2. ZigBee

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (wireless personal area network, WPAN). Su principal objetivo son las aplicaciones para redes Wireless que requieran comunicaciones seguras y fiables con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

El nombre ZigBee se deriva de los patrones erráticos comunicativos que hacen muchas abejas cuando se encuentran entre las flores, durante la recogida de polen.

ZigBee se ha desarrollado para satisfacer la creciente demanda de capacidad de red inalámbrica entre varios dispositivos de baja potencia. En la industria ZigBee se esta utilizando para la próxima generación de fabricación automatizada, con pequeños transmisores en cada dispositivo, lo que permite la comunicación entre dispositivos a un ordenador central.

Para llevar a cabo este sistema, un grupo de trabajo llamado Alianza ZigBee (en inglés ZigBee Alliance) formado por varias industrias, sin ánimo de lucro, la mayoría de ellas fabricantes de semiconductores, está desarrollando el estándar. La alianza de empresas está trabajando codo con codo con IEEE[2] para asegurar una integración, completa y operativa. Esta alianza en las cuales destacan empresas como Invensys, Mitsubishi, Philips y Motorota trabajan para crear un sistema estándar de comunicaciones, vía radio y bidireccional, para usarlo dentro de dispositivos de automatización domótica, inmótica[3], control industrial, periféricos de PC y sensores médicos. Los miembros de esta alianza justifican el desarrollo de este estándar para cubrir el vacío que se produce por debajo del Bluetooth.

D.3. WiFi

Se define **Wi-Fi** (Wireless Fidelity[4]) como conjunto de estándares para redes inalámbricas basado en las especificaciones IEEE 802.11 (especialmente la 802.11b), creado para redes locales inalámbricas, pero que también se utiliza para acceso a

[2]corresponde a las siglas de The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial cuyo trabajo es promover la creatividad, el desarrollo y la integración, compartir y aplicar los avances en las tecnologías de la información, electrónica y ciencias en general para beneficio de la humanidad y de los mismos profesionales.

[3]Entendemos por inmótica la incorporación al equipamiento de edificios de uso terciario o industrial sistemas de gestión técnica automatizada de las instalaciones. Con el objetivo de reducir el consumo de energía, aumentar el comfort y la seguridad de los mismos

[4]Realmente el término no viene de Wireless Fidelity como equivalente a Hi-Fi, High Fidelity, que se usa en la grabación de sonido. La WECA contrató a una empresa de publicidad para que le diera un nombre a su estándar, de tal manera que fuera fácil de identificar y recordar.

internet.

Nokia y Symbol Technologies crearon en 1999 la asociación conocida como WECA (Wireless Ethernet Compatibility Alliance, Alianza de Compatibilidad Ethernet Inalámbrica). Esta asociación fue la que luego pasó a denominarse Wi-Fi Alliance en 2003. Se formó con un objetivo claro, crear una marca que permitiese fomentar más fácilmente la tecnología inalámbrica y asegurar la compatibilidad de equipos.

De esta forma en abril de 2000 WECA certifica la interoperatibilidad de equipos según la norma IEEE 802.11b bajo la marca Wi-Fi. Esto quiere decir que el usuario tiene la garantía de que todos los equipos que tengan el sello Wi-Fi pueden trabajar juntos sin problemas, independientemente del fabricante de cada uno de ellos.

D.4. Bluetooth

Bluetooth[5], es una especificación industrial para Redes Inalámbricas de Área Personal, posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura y globalmente libre (2,4 GHz.). Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar la comunicación entre equipos tanto móviles como fijos.
- Eliminación de cables y conectores entre ellos
- Se pueden crear pequeñas redes inalámbricas y facilitar su sincronización.

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, con una cobertura baja y basados en transceptores de bajo coste.

Gracias a este protocolo, los dispositivos que lo implementan pueden comunicarse entre ellos cuando se encuentran dentro de su alcance. Se realizan por radiofrecuencia de forma que los dispositivos no tienen que estar alineados ni en contacto. Si la potencia de transmisión lo permite pueden incluso estar en habitaciones separadas. Estos dispositivos se clasifican como çlase 1", çlase 2.º çlase 3", cuanto menor sea su clase mayor será su potencia de transmisión. Aunque sus potencias y alcances sean menores son totalmente compatibles entre ellos. En la siguiente tabla podemos ver una clasificación con las distintas potencias y rango de conexión en función de la clase:

Clase	Potencia máx. permitida (mW)	Potencia máx. permitida (dBm)	Rango (m)
1	100	20	100
2	2,5	4	10
3	1	0	1

[5]Diente azul o Dientazul

En 1994, Ericsson inició un estudio para investigar la viabilidad de una nueva interfaz de bajo costo y consumo para la interconexión vía radio (eliminando así cables) entre dispositivos como teléfonos móviles y otros accesorios. El estudio partía de un largo proyecto que investigaba unos multicomunicadores conectados a una red celular, hasta que se llegó a un enlace de radio de corto alcance, llamado MC link. Conforme este proyecto avanzaba se fue haciendo claro que éste tipo de enlace podía ser utilizado ampliamente en un gran número de aplicaciones, debido a que tenía como principal virtud que se basaba en un chip de radio.

La especificación de Bluetooth define un canal de comunicación de máximo 720 kb/s (1 Mbps de capacidad bruta) con rango óptimo de 10 metros (opcionalmente 100 m con repetidores).

La frecuencia de radio con la que trabaja está en el rango de 2,4 a 2,48 GHz con amplio espectro y saltos de frecuencia con posibilidad de transmitir en Full Duplex[6] con un máximo de 1600 saltos/s. Los saltos de frecuencia se dan entre un total de 79 frecuencias con intervalos de 1Mhz; esto permite dar seguridad y robustez.

La potencia de salida para transmitir a una distancia máxima de 10 metros es de 0 dBm (1 mW), mientras que la versión de largo alcance transmite entre 20 y 30 dBm (entre 100 mW y 1 W).

Para lograr alcanzar el objetivo de bajo consumo y bajo costo, se ideó una solución que se puede implementar en un solo chip utilizando circuitos CMOS. De esta manera, se logró crear una solución de 9x9 mm y que consume aproximadamente 97 % menos energía que un teléfono móvil común.

D.5. GPRS

GPRS (General Packet Radio Service o servicio general de paquetes vía radio) es una extensión del Sistema Global para Comunicaciones Móviles (Global System for Mobile Communications o GSM) para la transmisión de datos no conmutada (o por paquetes).

El acceso al canal utilizado en GPRS se basa en divisiones de frecuencia sobre un dúplex y TDMA[7]. Durante la conexión, a cada usuario se le asigna un par de canales de frecuencia, uno para subida y otro para bajada. Esto se combina con la multiplexación estadística en el dominio del tiempo, permitiendo a varios usuarios compartir el mismo canal de frecuencia. Los paquetes tienen longitud constante, correspondiente a la ranura de tiempo del GSM.

Existen tres clases de dispositivos móviles teniendo en cuenta la posibilidad de

[6] los aparatos que trabajan en full duplex pueden transmitir y recibir al mismo tiempo

[7] es una tecnología inalámbrica de segunda generación, que distribuye las unidades de información en ranuras alternas de tiempo, dando acceso múltiple a un número reducido de frecuencias. TDMA permite dar servicios de alta calidad de voz y datos.

usar servicios GSM y GPRS simultáneamente:

- **Clase A** Estos dispositivos pueden utilizar simultáneamente servicios GPRS y GSM.
- **Clase B** Sólo pueden estar conectados a uno de los dos servicios en cada momento. Mientras se utiliza un servicio GSM (llamadas de voz o SMS), se suspende el servicio GPRS, que se reinicia automáticamente cuando finaliza el servicio GSM. La mayoría de los teléfonos móviles son de este tipo.
- **Clase C** Se conectan alternativamente a uno u otro servicio. El cambio entre GSM y GPRS debe realizarse de forma manual.

D.6. Comparación de tecnologías inalámbricas

	Wi-Fi	**Bluetooth**	**ZigBee**	**GPRS**
Bandas de frecuencia	2.4 GHz	2.4 GHz	2.4 GHz, 868/915 MHz	900/1800MHz
Tamaño de pila de memoria	1 Mb	1 Mb	20 kb	
Tasa de transferencia	11 Mbps	1 Mbps	250 kbps (2.4 GHz), 40 (915 MHz), 20 (868 MHz)	9.6 a 114 kbps
Número de canales	11-14	79	16(2.4 GHz), 10(915 MHz), 1(868 MHz)	Digtal
Tipos de datos	Digital	Digital, Audio	Digital (texto)	1000 m
Rango de transmisión	100 m	10m-100m	10m/100m	1
Número de dispositivos	32	8	255/65535	
Requisitos de alimentación	Media - horas de batería	Media- dias de bateria	Muy baja- años de batería	Alta
Introducción al mercado	Alta	Media	Baja	Puntos de acceso
Arquitecturas	Estrella	Estrella	Estrella, árbol, punto a punto y malla	Red global de voz y datos
Mejores aplicaciones	Edificio	Pc, teléfonos y elementos móviles	Control de bajo coste y monitorización	Alto
Consumo de potencia	400 mA transmitiendo, 20 mA en reposo	40 mA transmitiendo, 0.20 mA en reposo	30 mA transmitiendo, 3 mA en reposo	Alto
Precio	Alto	Normal	Bajo	Alto
Complejidad	Complejo	Complejo	Simple	Media

E

CÓDIGOS DE PROGRAMACIÓN

En este anexo se van a incluir los códigos necesarios para hacer funcionar el **Programa de comunicación via MiWi** (Sección 5.5). Y posteriormente se introducirá todo el código del programa principal o archivo main del Programa de comunicación via MiWi.

E.1. Archivos auxiliares

- MiWiDefs.h
- MiWi.h
- MRF24J40.h
- MSPI.c
- 18f2550.lkr
- 1852550.h

La mayoría del código ha sido desarrollado por Microchip, ya que es el diseñador y desarrollador principal del protocolo MiWi. En el proyecto lo que se ha hecho son modificaciones sobre estos para adaptarlos a las condiciones de nuestras placas.

E.1.1. MiWiDefs.h

Este es el archivo que se crea cuando se utiliza el ZENA(Sección B.6), en él se dan las definiciones básicas de la placa que se va a utilizar.

MiWiDefs.h

```
1   // ****************************************************************************
2   //
3   //  Software License Agreement
4   //
5   //  Copyright (c) 2007-2008 Microchip Technology Inc.
6   //  All rights reserved.
7   //
8   //  Microchip licenses to you the right to use, modify, copy and distribute
9   //  Software only when embedded on a Microchip microcontroller or digital
10  //  signal controller and used with a Microchip radio frequency transceiver,
11  //  which are integrated into your product or third party product (pursuant
12  //  to the sublicense terms in the accompanying license agreement).
13  //
14  //  You should refer to the license agreement accompanying this Software for
15  //  additional information regarding your rights and obligations.
16  //
17  //  SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
18  //  KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
19  //  WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A
20  //  PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE
21  //  LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,
22  //  CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY
23  //  DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY
24  //  INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST
25  //  PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,
26  //  TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT
27  //  LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
28  //
29  //  08/20/08
30  //
31  // ****************************************************************************
32
33  // Created by ZENA(TM) Version 3.0.0.0, 10/24/2009, 2:11:27
34
35  #ifndef _MIWIDEFS_H_
36  #define _MIWIDEFS_H_
37
38
39  // Build Configuration
40  #define ENABLE_CONSOLE
41  #define CUSTOM_PIC18
42
43  // MAC Address
44  #define EUI_7 0x55
45  #define EUI_6 0x01
46  #define EUI_5 0x02
47  #define EUI_4 0x03
48  #define EUI_3 0x04
49  #define EUI_2 0x05
50  #define EUI_1 0x06
51  #define EUI_0 0x07
52
53  // PIC Information
54  #define CLOCK_FREQ 20000000
55  #define BAUD_RATE 19200
56
57
58
59  // Device Information
60  #define I_AM_RFD
61  #define DEVICE_TYPE 0       // RFD
62  #define ALTERNATE_PAN_COORDINATOR 0
63  #define RX_ON_WHEN_IDLE 0
64  #define POWER_SOURCE 0      // Not Mains
65  #define ALLOCATE_ADDRESS 1
66
67  // Security Configuration
68  #define SECURITY_CAPABLE 0
69  #define CAP_INFO ((((BYTE)ALLOCATE_ADDRESS)<<7) |
70  (((BYTE)SECURITY_CAPABLE)<<6) |
```

```
71  (((BYTE)RX_ON_WHEN_IDLE)<<3) | (((BYTE)POWER_SOURCE)<<2) |
72  (((BYTE)DEVICE_TYPE)<<1) | ALTERNATE_PAN_COORDINATOR ) // 0x80
73
74  // Transceiver Configuration
75  #define TMRL TMR0L
76  #define RFIF INTCON3bits.INT2IF
77  #define RFIE INTCON3bits.INT2IE
78  #define RF_INT_PIN PORTBbits.RB2
79  #define PHY_CS LATBbits.LATB3
80  #define PHY_CS_TRIS TRISBbits.TRISB3
81  #define PHY_RESETn LATBbits.LATB4
82  #define PHY_RESETn_TRIS TRISBbits.TRISB4
83  #define PHY_WAKE LATBbits.LATB5
84  #define PHY_WAKE_TRIS TRISBbits.TRISB5
85
86  #define PA_LEVEL 0x00  // -0.00 dBm
87  #define FREQUENCY_BAND 2400
88  #define ALLOWED_CHANNELS
89  CHANNEL_11,CHANNEL_12,CHANNEL_13,CHANNEL_14,CHANNEL_15,CHANNEL_16,
90  CHANNEL_17,CHANNEL_18,CHANNEL_19,CHANNEL_20,CHANNEL_21,CHANNEL_22,
91  CHANNEL_23,CHANNEL_24,CHANNEL_25,CHANNEL_26
92  #define AVAILABLE_CHANNELS_SIZE 16
93
94  // Device Configuration Definitions
95  #define SUPPORT_CLUSTER_SOCKETS #define SUPPORT_EUI_ADDRESS_SEARCH
96
97  // Message Buffers
98  #define TX_BUFFER_SIZE 40 #define RX_BUFFER_SIZE 40
99
100 // Timeouts
101 #define NETWORK_DISCOVERY_TIMEOUT 0x00007A12 #define
102 OPEN_CLUSTER_SOCKET_TIMEOUT 0x0002DC6C
103
104 // Additional NWK/MAC Constants
105 #define RFD_DATA_WAIT 0x00007A12 #define NETWORK_TABLE_SIZE 10
106 #define MAX_HOPS 4 #define RSSI_SAMPLES_PER_CHANNEL 5
107
108 //Definiciones
109
110     #define PUSH_BUTTON_1 PORTBbits.RB6
111     #define PUSH_BUTTON_2 PORTBbits.RB7
112     #define PUSH_BUTTON_3 PORTAbits.RA2
113     #define PUSH_BUTTON_4 PORTAbits.RA5
114
115     #define LED_1 PORTCbits.RC1
116     #define LED_2 PORTCbits.RC2
117
118     #define PUSH_BUTTON_1_TRIS TRISBbits.TRISB6
119     #define PUSH_BUTTON_2_TRIS TRISBbits.TRISB7
120     #define PUSH_BUTTON_3_TRIS TRISAbits.TRISA2
121     #define PUSH_BUTTON_4_TRIS TRISAbits.TRISA5
122
123     #define LED_1_TRIS TRISCbits.TRISC1
124     #define LED_2_TRIS TRISCbits.TRISC2
125
126
127 // Validation
128
129 // ZENA(TM) will automatically range check the entered values.  These range
130 // checks are included here in cases the application developer manually
131 // adjusts the values.
132
133 #if (RX_BUFFER_SIZE > 127)
134     #error RX BUFFER SIZE too large. Must be <= 127.
135 #endif
136
137 #if (TX_BUFFER_SIZE > 127)
138     #error TX BUFFER SIZE too large. Must be <= 127.
139 #endif
140
141 #if (RX_BUFFER_SIZE < 25)
142     #error RX BUFFER SIZE too small. Must be >= 25.
143 #endif
144
145 #if (TX_BUFFER_SIZE < 25)
146     #error TX BUFFER SIZE too small. Must be >= 25.
147 #endif
148
149 #if (NETWORK_TABLE_SIZE == 0)
```

```
150         #error NETWORK TABLE SIZE too small.
151     #endif
152
153     #if (NETWORK_TABLE_SIZE > 0xFE)
154         #error NETWORK TABLE SIZE too large.  Must be < 0xFF.
155     #endif
156
157     #if (INDIRECT_BUFFER_SIZE > 0xFE)
158         #error INDIRECT BUFFER SIZE too large.  Must be < 0xFF.
159     #endif
160
161     #if defined( PICDEMZ) || defined( CUSTOM_PIC18 )
162         #if defined(__dsPIC30F__) || defined(__dsPIC33F__)
163             || defined(__PIC24F__) || defined(__PIC24H__)
164             #error "Incorrect board/processor combination."
165         #endif
166     #endif
167
168     #if defined( EXPLORER16 ) || defined( CUSTOM_16BIT )
169         #if defined(__18CXX)
170             #error "Incorrect board/processor combination."
171         #endif
172     #endif
173
174
175     #endif
176
```

E.1.2. MiWi.h

Se hacen las definiciones de las funciones del protocolo, de los usos de la memoria del micro por el módulo y se crean variables necesarias para el envío de datos por RF.

MiWi.h

```
/*********************************************************************
* FileName:     MiWi.h
* Dependencies: SymbolTime.h
* Processor:    PIC18, PIC24F, PIC24H, dsPIC30, dsPIC33
*               tested with 18F4620, dsPIC33FJ256GP710
* Hardware:     PICDEM Z, Explorer 16
* Complier:     Microchip C18 v3.04 or higher
*               Microchip C30 v2.03 or higher
* Company:      Microchip Technology, Inc.
*
* Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.
*
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller and used with a Microchip radio frequency transceiver,
* which are integrated into your product or third party product (pursuant
* to the terms in the accompanying license agreement).
*
* You should refer to the license agreement accompanying this Software for
* additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
* WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A
* PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE
* LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,
* CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY
* DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO
* ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES,
* LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,
* TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT
* NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*
*********************************************************************
* File Description:
*
*  This file provides the interface into the MiWi stack
*
* Change History:
*  Rev   Date         Description
*  0.1   11/09/2006   Initial revision
*  1.0   01/09/2007   Initial release
********************************************************************/

#ifndef __MIWI_H_ #define __MIWI_H_

/************************ HEADERS
**********************************/ #include "MiWiDefs.h" #include
".\MiWi\SymbolTime.h"

/************************ FUNCTION PROTOTYPES **********************/
void MiWiInit(void);
void MiWiTasks(void);
void DiscardPacket(void);
#ifdef SUPPORT_SECURITY
    BOOL SendTxBuffer(BOOL AckRequested, BOOL SecurityOn);
#else
    BOOL SendTxBuffer(BOOL AckRequested);
#endif
BYTE findNextNetworkEntry(void);
void DumpNetworkTable(void);
void SetChannel(BYTE channel);
void ClearNetworkTable(BYTE options);
void FormNetwork(WORD PANID);
void JoinNetwork(BYTE handle);
BOOL SendIndirectMessage(BYTE handle);
BYTE SearchForLongAddress(void);
```

```
68  BYTE SearchForShortAddress(BYTE entryType);
69  void SendIndirectPacket(BYTE handle, BYTE dataRequestType);
70  BYTE AddNodeToNetworkTable(void);
71  BYTE SendReportByShortAddress(WORD_VAL PANID, WORD_VAL ShortAddress,
72  BOOL forwardPacket);
73  BYTE SendReportByLongAddress(BYTE *pLongAddress);
74  BYTE SendReportByHandle(BYTE handle, BOOL forwardPacket);
75  void DiscoverNodeByEUI(void);
76  void OpenSocket(BYTE socketType);
77  void initMRF24J40(void);
78  void MRF24J40Sleep(void);
79  void MRF24J40Wake(void);
80
81  void PHYSetLongRAMAddr(WORD address, BYTE value);
82  void PHYSetShortRAMAddr(BYTE address, BYTE value);
83  BYTE PHYGetShortRAMAddr(BYTE address);
84  BYTE PHYGetLongRAMAddr(WORD address);
85
86  void PHYDumpRegisters(void);
87  void PHYDumpTxFIFO(void);
88
89  /************************ DEFINITIONS ******************************/
90  #ifdef __PIC32MX
91      #define FRAME_TYPE_COMMAND  0x03
92      #define FRAME_TYPE_DATA     0x01
93      #define FRAME_TYPE_BEACON   0x00
94      #define FRAME_TYPE_ACK      0x02
95  #else
96      #define FRAME_TYPE_COMMAND  0b011
97      #define FRAME_TYPE_DATA     0b001
98      #define FRAME_TYPE_BEACON   0b000
99      #define FRAME_TYPE_ACK      0b010
100 #endif
101
102 #define MAC_COMMAND_ASSOCIATION_REQUEST             0x01
103 #define MAC_COMMAND_ASSOCIATION_RESPONSE            0x02
104 #define ASSOCIATION_SUCCESSFUL                      0x00
105 #define ASSOCIATION_PAN_FULL                        0x01
106 #define ASSOCIATION_ACCESS_DENIED                   0x02
107 #define MAC_COMMAND_DISASSOCIATION_NOTIFICATION     0x03
108 #define MAC_COMMAND_DATA_REQUEST                    0x04
109 #define MAC_COMMAND_PAN_ID_CONFLICT_NOTIFICATION    0x05
110 #define MAC_COMMAND_ORPHAN_NOTIFICATION             0x06
111 #define MAC_COMMAND_BEACON_REQUEST                  0x07
112 #define MAC_COMMAND_COORDINATOR_REALIGNMENT         0x08
113
114 #define MIWI_PROTOCOL_ID 0x4D
115 #define MIWI_VERSION_NUM 0x10 //v1.0
116 #define MIWI_ACK_REQ 0x04
117
118 #ifdef __PIC32MX
119     #define CLEAR_ALL_ENTRIES 0xFF
120     #define CLEAR_NON_DIRECT_CONNECTIONS 0x01
121     #define CLEAR_NO_LONG_ADDRESS 0x02
122     #define CLEAR_NO_SHORT_ADDRESS 0x04  //this does not effect P2P nodes
123     #define CLEAR_P2P 0x08
124     #define CLEAR_NETWORKS 0x10
125     #define CLEAR_NEIGHBORS 0x20
126 #else
127     #define CLEAR_ALL_ENTRIES 0b11111111
128     #define CLEAR_NON_DIRECT_CONNECTIONS 0b00000001
129     #define CLEAR_NO_LONG_ADDRESS 0b00000010
130     #define CLEAR_NO_SHORT_ADDRESS 0b00000100  //this does not effect P2P nodes
131     #define CLEAR_P2P 0b00001000
132     #define CLEAR_NETWORKS 0b00010000
133     #define CLEAR_NEIGHBORS 0b00100000
134 #endif
135
136 #define NETWORK 0 #define NEIGHBOR 1
137
138 #define aResponseWaitTime 30720   //(16*32*60)
139
140 #define CLUSTER_SOCKET 0x00 #define P2P_SOCKET 0x01
141
142 #define ROLE_FFD_END_DEVICE 0x00 #define ROLE_COORDINATOR 0x01
143 #define ROLE_PAN_COORDINATOR 0x02
144
145 #define DATA_REQUEST_ASSOCIATION_RESPONSE 0x00 #define
146 DATA_REQUEST_SHORT_ADDRESSES 0x01
```

```
147
148   #define NETWORKED_TRANSMISSION 0x00 #define P2P_TRANSMISSION 0x01
149   #define BY_HANDLE_TRANSMISSION 0x02 #define
150   BY_LONG_ADDRESS_TRANSMISSION 0x04
151
152   /* Report type and ID definitions */ /* as a user you are able to
153   use Report types 0x10 - 0xFF */
154
155   #define MIWI_STACK_REPORT_TYPE 0x00
156       #define OPEN_CLUSTER_SOCKET_REQUEST 0x10
157       #define OPEN_CLUSTER_SOCKET_RESPONSE 0x11
158       #define OPEN_P2P_SOCKET_REQUEST 0x12
159       #define OPEN_P2P_SOCKET_RESPONSE 0x13
160       #define EUI_ADDRESS_SEARCH_REQUEST 0x20
161       #define EUI_ADDRESS_SEARCH_RESPONSE 0x21
162       #define ACK_REPORT_TYPE 0x30
163   /* Report Type 0x06-0x0F are reserved */ #define USER_REPORT_TYPE
164   0x12 #define USER_REPORT_TYPE_1 0x14 #define USER_REPORT_TYPE_2
165   0x16 #define USER_REPORT_TYPE_3 0x18
166
167
168
169       #define LIGHT_REPORT 0x34
170           #define LIGHT_TOGGLE 0x55
171           #define LIGHT_OFF 0x00
172           #define LIGHT_ON 0xFF
173
174   #define CIPHER_RETRY 3
175
176   /************************ DATA TYPE ******************************/
177
178   typedef union _MAC_FRAME_CONTROL {
179       BYTE Val;
180       struct _MAC_FRAME_CONTROL_bits
181       {
182           BYTE frameType :3;
183           BYTE securityEnabled :1;
184           BYTE framePending :1;
185           BYTE ACKRequest :1;
186           BYTE intraPAN :1;
187           BYTE :1;
188       }
189       bits;
190   }
191   MAC_FRAME_CONTROL;
192
193   typedef union _BEACON_SUPERFRAME {
194       WORD_VAL word;
195
196       struct _BEACON_SUPERFRAME_bits
197       {
198           BYTE beaconOrder        :4;
199           BYTE superframeOrder    :4;
200           BYTE finalCAPslot       :4;
201           BYTE batteryLifeExt     :1;
202           BYTE :1;
203           BYTE PANCoordinator     :1;
204           BYTE associationPermit  :1;
205       }
206       bits;
207   }
208   BEACON_SUPERFRAME;
209
210
211   typedef union _SEND_REPORT_STATUS {
212       struct _SEND_REPORT_STATUS_bits
213       {
214           BYTE inUse :1;
215           BYTE macAckValid :1;
216           BYTE macAck :1;
217           BYTE appAckValid :1;
218           BYTE appAck :1;
219       } bits;
220       BYTE Val;
221   } SEND_REPORT_STATUS;
222
223   typedef union _MRF24J40_IFS {
224       BYTE Val;
225       struct _MRF24J40_IFS_bits
```

```
226         {
227             BYTE RF_TXIF :1;
228             BYTE :2;
229             BYTE RF_RXIF :1;
230     //      BYTE SECIF :1;
231             BYTE :4;
232         }
233         bits;
234     }
235     MRF24J40_IFREG;
236
237     typedef union _NETWORK_TABLE_ENTRY_status {
238         BYTE Val;
239         struct _NETWORK_TABLE_ENTRY_bits
240         {
241             BYTE isValid            :1; //1 = this entry is valid
242                                         //0 = this entry is not valid
243             BYTE directConnection   :1; //1 = can talk to this device directly
244                                         //0 = must route to this device
245             BYTE longAddressValid   :1; //1 = long address valid
246                                         //0 = long address unknown
247             BYTE shortAddressValid  :1; //1 = short address valid
248                                         //0 = short address unknown
249             BYTE P2PConnection      :1; //1 = P2P connection
250                                         //0 = network device
251             BYTE NeighborOrNetwork  :1; //1 = it is a neighbor
252                                         //0 = it is a network
253             BYTE RXOnWhenIdle       :1; //1 = RxOnWhenIdle
254                                         //0 = RxOffWhenIdle
255             BYTE isFamily           :1; //1 = either my parent or a child
256                                         //0 = someone else
257         }
258         bits;
259     }
260     NETWORK_STATUS_INFO;
261
262
263     typedef struct _NETWORK_TABLE_ENTRY {
264         //NETWORK_STATUS_INFO status;
265         WORD_VAL PANID;
266         WORD_VAL ShortAddress;
267     #ifdef SUPPORT_SECURITY
268         DWORD_VAL IncomingFrameCounter;
269     #elif !defined(__C30__)
270         BYTE filler[4];
271     #endif
272         union _NETWORK_TABLE_ENTRY_INFO
273         {
274             BYTE LongAddress[8];
275             struct _NETWORK_TABLE_ENTRY_INFO_NETWORK
276             {
277                 BYTE Channel;
278                 BYTE sampleRSSI;
279                 BYTE Protocol;
280                 BYTE Version;
281                 union _NETWORK_TABLE_ENTRY_INFO_NETWORK_STRUCT
282                 {
283                     BYTE Val;
284                     struct _NETWORK_TABLE_ENTRY_INFO_NETWORK_STRUCT_bits
285                     {
286                         BYTE :6;
287                         BYTE PANcoordinator     :1;
288                         BYTE associationPermit  :1;
289                     }
290                     bits;
291                 }
292                 flags;
293             }
294             networkInfo;
295         }
296         info;
297     }
298     NETWORK_TABLE_ENTRY;
299
300     typedef struct _MIWI_STATE_MACHINE
301     {
302         BYTE searchingForNetwork    :1;
303         BYTE attemptingToJoin       :1;
304         BYTE memberOfNetwork        :1;
```

```
305         BYTE beaconRequestSent      :1;
306         BYTE dataRequestSent        :1;
307         BYTE attemptingToRejoin     :1;
308         BYTE RxHasUserData          :1;
309         BYTE dataRequestNeeded      :1;
310         BYTE dataRequestComplete    :1;
311     }
312     MIWI_STATE_MACHINE;
313
314     typedef union _MIWI_STATUS
315     {
316         WORD Val;
317         struct _MIWI_STATUS_bits
318         {
319             BYTE RX_PENDING         :1;
320             BYTE RX_BUFFERED        :1;
321             BYTE RX_ENABLED         :1;
322             BYTE TX_BUSY            :1;
323             BYTE TX_PENDING_ACK     :1;
324             BYTE TX_SECURITY        :1;
325             BYTE PHY_SLEEPING       :1;
326         }
327         bits;
328     }
329     MIWI_STATUS;
330
331     typedef union _DEST_INFO {
332         BYTE *pLongAddr;
333         BYTE handle;
334         struct _DEST_INFO_NETWORKED
335         {
336             WORD_VAL PANID;
337             WORD_VAL ShortAddress;
338         } network;
339     } DEST_INFO;
340
341     typedef struct _OPEN_SOCKET {
342         union _OPEN_SOCKET_STATUS
343         {
344             struct _OPEN_SOCKET_STATUS_bits
345             {
346                 BYTE matchFound :1;
347                 BYTE requestIsOpen :1;
348                 BYTE itIsMe         :1;
349                 BYTE type :1; //CLUSTER_SOCKET = cluster socket, P2P_SOCKET = P2P socket
350             } bits;
351             BYTE Val;
352         } status;
353         BYTE socketHandle;
354         WORD_VAL ShortAddress1;
355         #if defined(I_AM_COORDINATOR_CAPABLE)
356             WORD_VAL ShortAddress2;
357             BYTE LongAddress2[8];
358         #endif
359         BYTE LongAddress1[8];
360         TICK socketStart;
361     } OPEN_SOCKET;
362
363     #if defined(SUPPORT_SECURITY) typedef enum _CIPHER_MODE {
364         MODE_ENCRYPTION,
365         MODE_DECRYPTION
366     } CIPHER_MODE;
367
368     typedef enum _CIPHER_STATUS {
369         CIPHER_SUCCESS = 0,
370         CIPHER_ERROR,
371         CIPHER_MIC_ERROR
372     } CIPHER_STATUS;
373
374     typedef struct _SECURITY_INPUT {
375         BYTE    *InputText;
376         BYTE    TextLen;
377         BYTE    *Header;
378         BYTE    HeaderLen;
379         BYTE    SourceAddress[8];
380         DWORD_VAL   FrameCounter;
381     } SECURITY_INPUT; #endif //#if defined(SUPPORT_SECURITY)
382     /************************ EXTERNAL VARIABLES
383     **********************/
```

```
384
385   extern MIWI_STATE_MACHINE MiWiStateMachine;
386   extern BYTE currentSearchChannel;
387   extern TICK startTick;
388   extern NETWORK_TABLE_ENTRY networkTable[NETWORK_TABLE_SIZE];
389   extern NETWORK_STATUS_INFO networkStatus[NETWORK_TABLE_SIZE];
390   extern BYTE TxBuffer[];
391   extern BYTE TxData;
392   extern WORD_VAL myPANID;
393   extern WORD_VAL myShortAddress;
394   extern BYTE *pRxData;
395   extern BYTE RxSize;
396   extern BYTE myParent;
397   extern BYTE tempLongAddress[8];
398   extern WORD_VAL tempShortAddress;
399   extern OPEN_SOCKET openSocketInfo;
400   extern BYTE currentRxByte;
401   extern BYTE currentTxByte; extern BYTE phyIsIdle;
402
403   /************************ MACROS **********************************/
404   #define DiscoverNetworks()  MiWiStateMachine.searchingForNetwork = 1;\
405                               MiWiStateMachine.beaconRequestSent = 1;\
406                               currentSearchChannel = 0;\
407                               startTick = TickGet();
408   #define RejoinNetwork()     MiWiStateMachine.attemptingToRejoin = 1;\
409                               MiWiStateMachine.beaconRequestSent = 1;\
410                               currentSearchChannel = 0;\
411                               startTick = TickGet();
412   #define SearchingForNetworks() MiWiStateMachine.searchingForNetwork
413   #define MemberOfNetwork() MiWiStateMachine.memberOfNetwork
414   #define AttemptingToJoinNetwork() MiWiStateMachine.attemptingToJoin
415   #define AttemptingToRejoinNetwork()MiWiStateMachine.attemptingToRejoin
416   #define RxPacket() MiWiStateMachine.RxHasUserData
417   #define ClearToSend() !MiWiStatus.bits.TX_BUSY
418   #define OpenSocketComplete() (!openSocketInfo.status.bits.requestIsOpen)
419   #define OpenSocketSuccessful() openSocketInfo.status.bits.matchFound
420   #define OpenSocketHandle() openSocketInfo.socketHandle
421   #define CheckForData() MiWiStateMachine.dataRequestNeeded = 1;
422   #define CheckForDataComplete() MiWiStateMachine.dataRequestComplete
423   #define WriteData(a) TxBuffer[TxData++] = a
424   #define FlushTx(){TxData = 0; TxHeader = TX_BUFFER_SIZE - 1;}
425   #define TxPayLoad()TxData = 11
426
427   #endif
```

E.1.3. MRF24J40.h

Se hacen las definiciones del módulo inalámbrico, el MRF24J40. Todos los registros y las direcciones de memoria que utiliza se definen en este archivo.

MRF24J40.h

```
/*********************************************************************
* FileName:     MRF24J40.h
* Dependencies: none
* Processor:    PIC18, PIC24F, PIC24H, dsPIC30, dsPIC33
*               tested with 18F4620, dsPIC33FJ256GP710
* Hardware:     PICDEM Z, Explorer 16
* Complier:     Microchip C18 v3.04 or higher
*               Microchip C30 v2.03 or higher
* Company:      Microchip Technology, Inc.
*
* Copyright and Disclaimer Notice
*
* Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.
*
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller and used with a Microchip radio frequency transceiver,
* which are integrated into your product or third party product (pursuant
* to the terms in the accompanying license agreement).
*
* You should refer to the license agreement accompanying this Software for
* additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
* WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A
* PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE
* LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,
* CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY
* DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO
* ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES,
* LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,
* TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT
* NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*
*********************************************************************
* File Description:
*
*  Defines all of the address and channel settings for the registers
*   in the MRF24J40
*
* Change History:
*  Rev   Date         Description
*  0.1   11/09/2006   Initial revision
*  1.0   01/09/2007   Initial release
********************************************************************/

#if !defined(_ZMRF24J40_H_)
#define _ZMRF24J40_H_

//long address registers
#define RFCTRL0 (0x200)
#define RFCTRL1 (0x201)
#define RFCTRL2 (0x202)
#define RFCTRL3 (0x203)
#define RFCTRL4 (0x204)
#define RFCTRL5 (0x205)
#define RFCTRL6 (0x206)
#define RFCTRL7 (0x207)
#define RFCTRL8 (0x208)
#define CAL1 (0x209)
#define CAL2 (0x20a)
#define CAL3 (0x20b)
#define SFCNTRH (0x20c)
#define SFCNTRM (0x20d)
#define SFCNTRL (0x20e)
#define RFSTATE (0x20f)
#define RSSI (0x210)
#define CLKIRQCR (0x211)
```

```
70   #define SRCADRMODE (0x212)
71   #define SRCADDR0 (0x213)
72   #define SRCADDR1 (0x214)
73   #define SRCADDR2 (0x215)
74   #define SRCADDR3 (0x216)
75   #define SRCADDR4 (0x217)
76   #define SRCADDR5 (0x218)
77   #define SRCADDR6 (0x219)
78   #define SRCADDR7 (0x21a)
79   #define RXFRAMESTATE (0x21b)
80   #define SECSTATUS (0x21c)
81   #define STCCMP (0x21d)
82   #define HLEN (0x21e)
83   #define FLEN (0x21f)
84   #define SCLKDIV (0x220)
85   #define reserved (0x221)
86   #define WAKETIMEL (0x222)
87   #define WAKETIMEH (0x223)
88   #define TXREMCNTL (0x224)
89   #define TXREMCNTH (0x225)
90   #define TXMAINCNTL (0x226)
91   #define TXMAINCNTM (0x227)
92   #define TXMAINCNTH0 (0x228)
93   #define TXMAINCNTH1 (0x229)
94   #define RFMANUALCTRLEN (0x22a)
95   #define RFMANUALCTRL (0x22b)
96   #define RFRXCTRL RFMANUALCTRL
97   #define TxDACMANUALCTRL (0x22c)
98   #define RFMANUALCTRL2 (0x22d)
99   #define TESTRSSI (0x22e)
100  #define TESTMODE (0x22f)
101
102  #define NORMAL_TX_FIFO  (0x000)
103  #define BEACON_TX_FIFO  (0x080)
104  #define GTS1_TX_FIFO    (0x100)
105  #define GTS2_TX_FIFO    (0x180)
106
107  #define RX_FIFO         (0x300)
108
109  #define SECURITY_FIFO   (0x280)
110
111  //short address registers for reading
112  #define READ_RXMCR (0x00)
113  #define READ_PANIDL (0x02)
114  #define READ_PANIDH (0x04)
115  #define READ_SADRL (0x06)
116  #define READ_SADRH (0x08)
117  #define READ_EADR0 (0x0A)
118  #define READ_EADR1 (0x0C)
119  #define READ_EADR2 (0x0E)
120  #define READ_EADR3 (0x10)
121  #define READ_EADR4 (0x12)
122  #define READ_EADR5 (0x14)
123  #define READ_EADR6 (0x16)
124  #define READ_EADR7 (0x18)
125  #define READ_RXFLUSH (0x1a)
126  #define READ_TXSTATE0 (0x1c)
127  #define READ_TXSTATE1 (0x1e)
128  #define READ_ORDER (0x20)
129  #define READ_TXMCR (0x22)
130  #define READ_ACKTMOUT (0x24)
131  #define READ_SLALLOC (0x26)
132  #define READ_SYMTICKL (0x28)
133  #define READ_SYMTICKH (0x2A)
134  #define READ_PAONTIME (0x2C)
135  #define READ_PAONSETUP (0x2E)
136  #define READ_FFOEN (0x30)
137  #define READ_CSMACR (0x32)
138  #define READ_TXBCNTRIG (0x34)
139  #define READ_TXNMTRIG (0x36)
140  #define READ_TXG1TRIG (0x38)
141  #define READ_TXG2TRIG (0x3A)
142  #define READ_ESLOTG23 (0x3C)
143  #define READ_ESLOTG45 (0x3E)
144  #define READ_ESLOTG67 (0x40)
145  #define READ_TXPEND (0x42)
146  #define READ_TXBCNINTL (0x44)
147  #define READ_FRMOFFSET (0x46)
148  #define READ_TXSR (0x48)
```

```
149  #define READ_TXLERR (0x4A)
150  #define READ_GATE_CLK (0x4C)
151  #define READ_TXOFFSET (0x4E)
152  #define READ_HSYMTMR0 (0x50)
153  #define READ_HSYMTMR1 (0x52)
154  #define READ_SOFTRST (0x54)
155  #define READ_BISTCR (0x56)
156  #define READ_SECCR0 (0x58)
157  #define READ_SECCR1 (0x5A)
158  #define READ_TXPEMISP (0x5C)
159  #define READ_SECISR (0x5E)
160  #define READ_RXSR (0x60)
161  #define READ_ISRSTS (0x62)
162  #define READ_INTMSK (0x64)
163  #define READ_GPIO (0x66)
164  #define READ_GPIODIR (0x68)
165  #define READ_SLPACK (0x6A)
166  #define READ_RFCTL (0x6C)
167  #define READ_SECCR2 (0x6E)
168  #define READ_BBREG0 (0x70)
169  #define READ_BBREG1 (0x72)
170  #define READ_BBREG2 (0x74)
171  #define READ_BBREG3 (0x76)
172  #define READ_BBREG4 (0x78)
173  #define READ_BBREG5 (0x7A)
174  #define READ_BBREG6 (0x7C)
175  #define READ_RSSITHCCA (0x7E)
176
177
178  //short address registers for reading
179  #define WRITE_RXMCR (0x01)
180  #define WRITE_PANIDL (0x03)
181  #define WRITE_PANIDH (0x05)
182  #define WRITE_SADRL (0x07)
183  #define WRITE_SADRH (0x09)
184  #define WRITE_EADR0 (0x0B)
185  #define WRITE_EADR1 (0x0D)
186  #define WRITE_EADR2 (0x0F)
187  #define WRITE_EADR3 (0x11)
188  #define WRITE_EADR4 (0x13)
189  #define WRITE_EADR5 (0x15)
190  #define WRITE_EADR6 (0x17)
191  #define WRITE_EADR7 (0x19)
192  #define WRITE_RXFLUSH (0x1B)
193  #define WRITE_TXSTATE0 (0x1D)
194  #define WRITE_TXSTATE1 (0x1F)
195  #define WRITE_ORDER (0x21)
196  #define WRITE_TXMCR (0x23)
197  #define WRITE_ACKTMOUT (0x25)
198  #define WRITE_SLALLOC (0x27)
199  #define WRITE_SYMTICKL (0x29)
200  #define WRITE_SYMTICKH (0x2B)
201  #define WRITE_PAONTIME (0x2D)
202  #define WRITE_PAONSETUP (0x2F)
203  #define WRITE_FFOEN (0x31)
204  #define WRITE_CSMACR (0x33)
205  #define WRITE_TXBCNTRIG (0x35)
206  #define WRITE_TXNMTRIG (0x37)
207  #define WRITE_TXG1TRIG (0x39)
208  #define WRITE_TXG2TRIG (0x3B)
209  #define WRITE_ESLOTG23 (0x3D)
210  #define WRITE_ESLOTG45 (0x3F)
211  #define WRITE_ESLOTG67 (0x41)
212  #define WRITE_TXPEND (0x43)
213  #define WRITE_TXBCNINTL (0x45)
214  #define WRITE_FRMOFFSET (0x47)
215  #define WRITE_TXSR (0x49)
216  #define WRITE_TXLERR (0x4B)
217  #define WRITE_GATE_CLK (0x4D)
218  #define WRITE_TXOFFSET (0x4F)
219  #define WRITE_HSYMTMR0 (0x51)
220  #define WRITE_HSYMTMR1 (0x53)
221  #define WRITE_SOFTRST (0x55)
222  #define WRITE_BISTCR (0x57)
223  #define WRITE_SECCR0 (0x59)
224  #define WRITE_SECCR1 (0x5B)
225  #define WRITE_TXPEMISP (0x5D)
226  #define WRITE_SECISR (0x5F)
227  #define WRITE_RXSR (0x61)
```

```
228  #define WRITE_ISRSTS (0x63)
229  #define WRITE_INTMSK (0x65)
230  #define WRITE_GPIO (0x67)
231  #define WRITE_GPIODIR (0x69)
232  #define WRITE_SLPACK (0x6B)
233  #define WRITE_RFCTL (0x6D)
234  #define WRITE_SECCR2 (0x6F)
235  #define WRITE_BBREG0 (0x71)
236  #define WRITE_BBREG1 (0x73)
237  #define WRITE_BBREG2 (0x75)
238  #define WRITE_BBREG3 (0x77)
239  #define WRITE_BBREG4 (0x79)
240  #define WRITE_BBREG5 (0x7B)
241  #define WRITE_BBREG6 (0x7D)
242  #define WRITE_RSSITHCCA (0x7F)
243
244  #define CHANNEL_11 0x00
245  #define CHANNEL_12 0x10
246  #define CHANNEL_13 0x20
247  #define CHANNEL_14 0x30
248  #define CHANNEL_15 0x40
249  #define CHANNEL_16 0x50
250  #define CHANNEL_17 0x60
251  #define CHANNEL_18 0x70
252  #define CHANNEL_19 0x80
253  #define CHANNEL_20 0x90
254  #define CHANNEL_21 0xa0
255  #define CHANNEL_22 0xb0
256  #define CHANNEL_23 0xc0
257  #define CHANNEL_24 0xd0
258  #define CHANNEL_25 0xe0
259  #define CHANNEL_26 0xf0
260
261  #endif
262
```

E.1.4. MSPI.c

Se crean las funciones necesarias para la comunicación SPI entre el módulo y el micro.

MSPI.c

```
/*****************************************************************
* FileName:     MSPI.c
* Dependencies: MSPI.h
* Processor:    PIC18, PIC24F, PIC24H, dsPIC30, dsPIC33
*               tested with 18F4620, dsPIC33FJ256GP710
* Hardware:     PICDEM Z, Explorer 16
* Complier:     Microchip C18 v3.04 or higher
*               Microchip C30 v2.03 or higher
* Company:      Microchip Technology, Inc.
*
* Copyright and Disclaimer Notice
*
* Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.
*
* Microchip licenses to you the right to use, modify, copy and distribute
* Software only when embedded on a Microchip microcontroller or digital
* signal controller and used with a Microchip radio frequency transceiver,
* which are integrated into your product or third party product (pursuant
* to the terms in the accompanying license agreement).
*
* You should refer to the license agreement accompanying this Software for
* additional information regarding your rights and obligations.
*
* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
* WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A
* PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE
* LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,
* CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY
* DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO
* ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES,
* LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,
* TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT
* NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
*
******************************************************************
* File Description:
*
*   Configures and enables usage of the SPI ports
*
* Change History:
*  Rev   Date         Description
*  0.1   11/09/2006   Initial revision
*  1.0   01/09/2007   Initial release
******************************************************************/

/************************ HEADERS **********************************/
#include "Compiler.h"
#include "GenericTypeDefs.h"

#if defined(__dsPIC30F__) || defined(__dsPIC33F__)
 || defined(__PIC24F__) || defined(__PIC24H__) || defined(__PIC32MX__)

/************************ FUNCTIONS ********************************/

/******************************************************************
* Function:         void SPIPut(BYTE v)
*
* PreCondition:     SPI has been configured
*
* Input:            v - is the byte that needs to be transfered
*
* Output:           none
*
* Side Effects:     SPI transmits the byte
*
* Overview:         This function will send a byte over the SPI
*
* Note:             None
```

```
70  ******************************************************************/
71  void SPIPut(BYTE v)
72  {
73      BYTE dummy;
74
75      #if defined(__PIC32MX__)
76          putcSPI1(v);
77          dummy = (BYTE)getcSPI1();
78      #else
79          IFS0bits.SPI1IF = 0;
80          dummy = SPI1BUF;
81          SPI1BUF = v;
82          while(IFS0bits.SPI1IF == 0){}
83      #endif
84  }
85
86  /*******************************************************************
87  * Function:         BYTE SPIGet(void)
88  *
89  * PreCondition:     SPI has been configured
90  *
91  * Input:            none
92  *
93  * Output:           BYTE - the byte that was last received by the SPI
94  *
95  * Side Effects:     none
96  *
97  * Overview:         This function will read a byte over the SPI
98  *
99  * Note:             None
100 *******************************************************************/
101 BYTE SPIGet(void)
102 {
103     #if defined(__PIC32MX__)
104         BYTE dummy;
105         putcSPI1(0x00);
106         dummy = (BYTE)getcSPI1();
107         return(dummy);
108     #else
109         SPIPut(0x00);
110         return SPI1BUF;
111     #endif
112 }
113
114 #elif defined(__18CXX)
115
116 /************************ FUNCTIONS ******************************/
117
118 /*******************************************************************
119 * Function:         void SPIPut(BYTE v)
120 *
121 * PreCondition:     SPI has been configured
122 *
123 * Input:            v - is the byte that needs to be transfered
124 *
125 * Output:           none
126 *
127 * Side Effects:     SPI transmits the byte
128 *
129 * Overview:         This function will send a byte over the SPI
130 *
131 * Note:             None
132 *******************************************************************/
133 void SPIPut(BYTE v)
134 {
135     RCSTAbits.SPEN = 0;
136     TXSTAbits.TXEN = 0;
137     PIR1bits.SSPIF = 0;
138     do
139     {
140         SSPCON1bits.WCOL = 0;
141         SSPBUF = v;
142     } while( SSPCON1bits.WCOL );
143
144     while( PIR1bits.SSPIF == 0 );
145     RCSTAbits.SPEN = 1;
146     TXSTAbits.TXEN = 1;
147 }
148
```

```
149  /*******************************************************************
150  * Function:         BYTE SPIGet(void)
151  *
152  * PreCondition:     SPI has been configured
153  *
154  * Input:            none
155  *
156  * Output:           BYTE - the byte that was last received by the SPI
157  *
158  * Side Effects:     none
159  *
160  * Overview:         This function will read a byte over the SPI
161  *
162  * Note:             None
163  *******************************************************************/
164  BYTE SPIGet(void)
165  {
166      SPIPut(0x00);
167      return SSPBUF;
168  }
169
170  #else
171      #error Unknown processor.  See Compiler.h
172  #endif
```

E.1.5. 18f2550.lkr

Es el archivo que necesita el compilador para poder linkar el programa con el microcontrolador. Es único para cada PIC. Microchip proporciona estos archivos de forma gratuita para su uso con el programa MPLAB IDE(Sección B.1.2) y con el compilador MCC18.

18f2550.lkr

```

// File: 18f2550.lkr
// Sample linker script for the PIC18F2550 processor

LIBPATH .

FILES c018i.o
FILES clib.lib
FILES p18f2550.lib

CODEPAGE   NAME=page       START=0x0               END=0x7FFF
CODEPAGE   NAME=idlocs     START=0x200000          END=0x200007   PROTECTED
CODEPAGE   NAME=config     START=0x300000          END=0x30000D   PROTECTED
CODEPAGE   NAME=devid      START=0x3FFFFE          END=0x3FFFFF   PROTECTED
CODEPAGE   NAME=eedata     START=0xF00000          END=0xF000FF   PROTECTED

ACCESSBANK NAME=accessram  START=0x0            END=0x5F
DATABANK   NAME=gpr0       START=0x60           END=0xFF
DATABANK   NAME=gpr1       START=0x100          END=0x1FF
DATABANK   NAME=gpr2       START=0x200          END=0x2FF
DATABANK   NAME=gpr3       START=0x300          END=0x3FF
DATABANK   NAME=usb4       START=0x400          END=0x4FF         PROTECTED
DATABANK   NAME=usb5       START=0x500          END=0x5FF         PROTECTED
DATABANK   NAME=usb6       START=0x600          END=0x6FF         PROTECTED
DATABANK   NAME=usb7       START=0x700          END=0x7FF         PROTECTED
ACCESSBANK NAME=accesssfr  START=0xF60          END=0xFFF         PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK SIZE=0x100 RAM=gpr3

```

E.1.6. 18f2550.h

Por último también es necesario incluir la librería del PIC a utilizar. También proporcionada con el MCC18 por Microchip. Viene incluida en las librerías del propio compilador y resulta necesaria para los usos de los registros de memoria, etc. Esta librería es necesaria incluirla en todos los programas pero debido a su extensión se ha colocado en este anexo.

18f2550.h

```
1
2   /*----------------------------------------------------------------------------
3    * MPLAB-Cxx  PIC18F2550 processor header
4    *
5    * (c) Copyright 1999-2007 Microchip Technology, All rights reserved
6    *----------------------------------------------------------------------------*/
7
8   #ifndef __18F2550_H
9   #define __18F2550_H
10
11  extern volatile near unsigned           UFRM;
12  extern volatile near unsigned char      UFRML;
13  extern volatile near struct {
14    unsigned FRM0:1;
15    unsigned FRM1:1;
16    unsigned FRM2:1;
17    unsigned FRM3:1;
18    unsigned FRM4:1;
19    unsigned FRM5:1;
20    unsigned FRM6:1;
21    unsigned FRM7:1;
22  } UFRMLbits;
23  extern volatile near unsigned char      UFRMH;
24  extern volatile near struct {
25    unsigned FRM8:1;
26    unsigned FRM9:1;
27    unsigned FRM10:1;
28  } UFRMHbits;
29  extern volatile near unsigned char      UIR;
30  extern volatile near struct {
31    unsigned URSTIF:1;
32    unsigned UERRIF:1;
33    unsigned ACTVIF:1;
34    unsigned TRNIF:1;
35    unsigned IDLEIF:1;
36    unsigned STALLIF:1;
37    unsigned SOFIF:1;
38  } UIRbits;
39  extern volatile near unsigned char      UIE;
40  extern volatile near struct {
41    unsigned URSTIE:1;
42    unsigned UERRIE:1;
43    unsigned ACTVIE:1;
44    unsigned TRNIE:1;
45    unsigned IDLEIE:1;
46    unsigned STALLIE:1;
47    unsigned SOFIE:1;
48  } UIEbits;
49  extern volatile near unsigned char      UEIR;
50  extern volatile near struct {
51    unsigned PIDEF:1;
52    unsigned CRC5EF:1;
53    unsigned CRC16EF:1;
54    unsigned DFN8EF:1;
55    unsigned BTOEF:1;
56    unsigned :2;
57    unsigned BTSEF:1;
58  } UEIRbits;
59  extern volatile near unsigned char      UEIE;
60  extern volatile near struct {
61    unsigned PIDEE:1;
62    unsigned CRC5EE:1;
63    unsigned CRC16EE:1;
64    unsigned DFN8EE:1;
```

```
65      unsigned BTOEE:1;
66      unsigned :2;
67      unsigned BTSEE:1;
68    } UEIEbits;
69    extern volatile near unsigned char       USTAT;
70    extern volatile near struct {
71      unsigned :1;
72      unsigned PPBI:1;
73      unsigned DIR:1;
74      unsigned ENDP0:1;
75      unsigned ENDP1:1;
76      unsigned ENDP2:1;
77      unsigned ENDP3:1;
78    } USTATbits;
79    extern volatile near unsigned char       UCON;
80    extern volatile near struct {
81      unsigned :1;
82      unsigned SUSPND:1;
83      unsigned RESUME:1;
84      unsigned USBEN:1;
85      unsigned PKTDIS:1;
86      unsigned SE0:1;
87      unsigned PPBRST:1;
88    } UCONbits;
89    extern volatile near unsigned char       UADDR;
90    extern volatile near struct {
91      unsigned ADDR0:1;
92      unsigned ADDR1:1;
93      unsigned ADDR2:1;
94      unsigned ADDR3:1;
95      unsigned ADDR4:1;
96      unsigned ADDR5:1;
97      unsigned ADDR6:1;
98    } UADDRbits;
99    extern volatile near unsigned char       UCFG;
100   extern volatile near struct {
101     unsigned PPB0:1;
102     unsigned PPB1:1;
103     unsigned FSEN:1;
104     unsigned UTRDIS:1;
105     unsigned UPUEN:1;
106     unsigned :1;
107     unsigned UOEMON:1;
108     unsigned UTEYE:1;
109   } UCFGbits;
110   extern volatile near unsigned char       UEP0;
111   extern volatile near struct {
112     unsigned EPSTALL:1;
113     unsigned EPINEN:1;
114     unsigned EPOUTEN:1;
115     unsigned EPCONDIS:1;
116     unsigned EPHSHK:1;
117   } UEP0bits;
118   extern volatile near unsigned char       UEP1;
119   extern volatile near struct {
120     unsigned EPSTALL:1;
121     unsigned EPINEN:1;
122     unsigned EPOUTEN:1;
123     unsigned EPCONDIS:1;
124     unsigned EPHSHK:1;
125   } UEP1bits;
126   extern volatile near unsigned char       UEP2;
127   extern volatile near struct {
128     unsigned EPSTALL:1;
129     unsigned EPINEN:1;
130     unsigned EPOUTEN:1;
131     unsigned EPCONDIS:1;
132     unsigned EPHSHK:1;
133   } UEP2bits;
134   extern volatile near unsigned char       UEP3;
135   extern volatile near struct {
136     unsigned EPSTALL:1;
137     unsigned EPINEN:1;
138     unsigned EPOUTEN:1;
139     unsigned EPCONDIS:1;
140     unsigned EPHSHK:1;
141   } UEP3bits;
142   extern volatile near unsigned char       UEP4;
143   extern volatile near struct {
```

```
144     unsigned EPSTALL:1;
145     unsigned EPINEN:1;
146     unsigned EPOUTEN:1;
147     unsigned EPCONDIS:1;
148     unsigned EPHSHK:1;
149   } UEP4bits;
150   extern volatile near unsigned char       UEP5;
151   extern volatile near struct {
152     unsigned EPSTALL:1;
153     unsigned EPINEN:1;
154     unsigned EPOUTEN:1;
155     unsigned EPCONDIS:1;
156     unsigned EPHSHK:1;
157   } UEP5bits;
158   extern volatile near unsigned char       UEP6;
159   extern volatile near struct {
160     unsigned EPSTALL:1;
161     unsigned EPINEN:1;
162     unsigned EPOUTEN:1;
163     unsigned EPCONDIS:1;
164     unsigned EPHSHK:1;
165   } UEP6bits;
166   extern volatile near unsigned char       UEP7;
167   extern volatile near struct {
168     unsigned EPSTALL:1;
169     unsigned EPINEN:1;
170     unsigned EPOUTEN:1;
171     unsigned EPCONDIS:1;
172     unsigned EPHSHK:1;
173   } UEP7bits;
174   extern volatile near unsigned char       UEP8;
175   extern volatile near struct {
176     unsigned EPSTALL:1;
177     unsigned EPINEN:1;
178     unsigned EPOUTEN:1;
179     unsigned EPCONDIS:1;
180     unsigned EPHSHK:1;
181   } UEP8bits;
182   extern volatile near unsigned char       UEP9;
183   extern volatile near struct {
184     unsigned EPSTALL:1;
185     unsigned EPINEN:1;
186     unsigned EPOUTEN:1;
187     unsigned EPCONDIS:1;
188     unsigned EPHSHK:1;
189   } UEP9bits;
190   extern volatile near unsigned char       UEP10;
191   extern volatile near struct {
192     unsigned EPSTALL:1;
193     unsigned EPINEN:1;
194     unsigned EPOUTEN:1;
195     unsigned EPCONDIS:1;
196     unsigned EPHSHK:1;
197   } UEP10bits;
198   extern volatile near unsigned char       UEP11;
199   extern volatile near struct {
200     unsigned EPSTALL:1;
201     unsigned EPINEN:1;
202     unsigned EPOUTEN:1;
203     unsigned EPCONDIS:1;
204     unsigned EPHSHK:1;
205   } UEP11bits;
206   extern volatile near unsigned char       UEP12;
207   extern volatile near struct {
208     unsigned EPSTALL:1;
209     unsigned EPINEN:1;
210     unsigned EPOUTEN:1;
211     unsigned EPCONDIS:1;
212     unsigned EPHSHK:1;
213   } UEP12bits;
214   extern volatile near unsigned char       UEP13;
215   extern volatile near struct {
216     unsigned EPSTALL:1;
217     unsigned EPINEN:1;
218     unsigned EPOUTEN:1;
219     unsigned EPCONDIS:1;
220     unsigned EPHSHK:1;
221   } UEP13bits;
222   extern volatile near unsigned char       UEP14;
```

```
223   extern volatile near struct {
224     unsigned EPSTALL:1;
225     unsigned EPINEN:1;
226     unsigned EPOUTEN:1;
227     unsigned EPCONDIS:1;
228     unsigned EPHSHK:1;
229   } UEP14bits;
230   extern volatile near unsigned char       UEP15;
231   extern volatile near struct {
232     unsigned EPSTALL:1;
233     unsigned EPINEN:1;
234     unsigned EPOUTEN:1;
235     unsigned EPCONDIS:1;
236     unsigned EPHSHK:1;
237   } UEP15bits;
238   extern volatile near unsigned char       PORTA;
239   extern volatile near union {
240     struct {
241       unsigned RA0:1;
242       unsigned RA1:1;
243       unsigned RA2:1;
244       unsigned RA3:1;
245       unsigned RA4:1;
246       unsigned RA5:1;
247       unsigned RA6:1;
248     };
249     struct {
250       unsigned AN0:1;
251       unsigned AN1:1;
252       unsigned AN2:1;
253       unsigned AN3:1;
254       unsigned T0CKI:1;
255       unsigned AN4:1;
256       unsigned OSC2:1;
257     };
258     struct {
259       unsigned :2;
260       unsigned VREFM:1;
261       unsigned VREFP:1;
262       unsigned :1;
263       unsigned LVDIN:1;
264     };
265     struct {
266       unsigned :5;
267       unsigned HLVDIN:1;
268     };
269   } PORTAbits;
270   extern volatile near unsigned char       PORTB;
271   extern volatile near union {
272     struct {
273       unsigned RB0:1;
274       unsigned RB1:1;
275       unsigned RB2:1;
276       unsigned RB3:1;
277       unsigned RB4:1;
278       unsigned RB5:1;
279       unsigned RB6:1;
280       unsigned RB7:1;
281     };
282     struct {
283       unsigned INT0:1;
284       unsigned INT1:1;
285       unsigned INT2:1;
286     };
287     struct {
288       unsigned :5;
289       unsigned PGM:1;
290       unsigned PGC:1;
291       unsigned PGD:1;
292     };
293   } PORTBbits;
294   extern volatile near unsigned char       PORTC;
295   extern volatile near union {
296     struct {
297       unsigned RC0:1;
298       unsigned RC1:1;
299       unsigned RC2:1;
300       unsigned :1;
301       unsigned RC4:1;
```

```
302       unsigned RC5:1;
303       unsigned RC6:1;
304       unsigned RC7:1;
305     };
306     struct {
307       unsigned T1OSO:1;
308       unsigned T1OSI:1;
309       unsigned CCP1:1;
310       unsigned :3;
311       unsigned TX:1;
312       unsigned RX:1;
313     };
314     struct {
315       unsigned T13CKI:1;
316       unsigned :1;
317       unsigned P1A:1;
318       unsigned :3;
319       unsigned CK:1;
320       unsigned DT:1;
321     };
322   } PORTCbits;
323   extern volatile near unsigned char       PORTE;
324   extern volatile near struct {
325     unsigned :3;
326     unsigned RE3:1;
327   } PORTEbits;
328   extern volatile near unsigned char       LATA;
329   extern volatile near struct {
330     unsigned LATA0:1;
331     unsigned LATA1:1;
332     unsigned LATA2:1;
333     unsigned LATA3:1;
334     unsigned LATA4:1;
335     unsigned LATA5:1;
336     unsigned LATA6:1;
337   } LATAbits;
338   extern volatile near unsigned char       LATB;
339   extern volatile near struct {
340     unsigned LATB0:1;
341     unsigned LATB1:1;
342     unsigned LATB2:1;
343     unsigned LATB3:1;
344     unsigned LATB4:1;
345     unsigned LATB5:1;
346     unsigned LATB6:1;
347     unsigned LATB7:1;
348   } LATBbits;
349   extern volatile near unsigned char       LATC;
350   extern volatile near struct {
351     unsigned LATC0:1;
352     unsigned LATC1:1;
353     unsigned LATC2:1;
354     unsigned :3;
355     unsigned LATC6:1;
356     unsigned LATC7:1;
357   } LATCbits;
358   extern volatile near unsigned char       DDRA;
359   extern volatile near struct {
360     unsigned RA0:1;
361     unsigned RA1:1;
362     unsigned RA2:1;
363     unsigned RA3:1;
364     unsigned RA4:1;
365     unsigned RA5:1;
366     unsigned RA6:1;
367   } DDRAbits;
368   extern volatile near unsigned char       TRISA;
369   extern volatile near struct {
370     unsigned TRISA0:1;
371     unsigned TRISA1:1;
372     unsigned TRISA2:1;
373     unsigned TRISA3:1;
374     unsigned TRISA4:1;
375     unsigned TRISA5:1;
376     unsigned TRISA6:1;
377   } TRISAbits;
378   extern volatile near unsigned char       DDRB;
379   extern volatile near struct {
380     unsigned RB0:1;
```

```
381     unsigned RB1:1;
382     unsigned RB2:1;
383     unsigned RB3:1;
384     unsigned RB4:1;
385     unsigned RB5:1;
386     unsigned RB6:1;
387     unsigned RB7:1;
388   } DDRBbits;
389   extern volatile near unsigned char       TRISB;
390   extern volatile near struct {
391     unsigned TRISB0:1;
392     unsigned TRISB1:1;
393     unsigned TRISB2:1;
394     unsigned TRISB3:1;
395     unsigned TRISB4:1;
396     unsigned TRISB5:1;
397     unsigned TRISB6:1;
398     unsigned TRISB7:1;
399   } TRISBbits;
400   extern volatile near unsigned char       DDRC;
401   extern volatile near struct {
402     unsigned RC0:1;
403     unsigned RC1:1;
404     unsigned RC2:1;
405     unsigned :3;
406     unsigned RC6:1;
407     unsigned RC7:1;
408   } DDRCbits;
409   extern volatile near unsigned char       TRISC;
410   extern volatile near struct {
411     unsigned TRISC0:1;
412     unsigned TRISC1:1;
413     unsigned TRISC2:1;
414     unsigned :3;
415     unsigned TRISC6:1;
416     unsigned TRISC7:1;
417   } TRISCbits;
418   extern volatile near unsigned char       OSCTUNE;
419   extern volatile near struct {
420     unsigned TUN0:1;
421     unsigned TUN1:1;
422     unsigned TUN2:1;
423     unsigned TUN3:1;
424     unsigned TUN4:1;
425     unsigned :2;
426     unsigned INTSRC:1;
427   } OSCTUNEbits;
428   extern volatile near unsigned char       PIE1;
429   extern volatile near struct {
430     unsigned TMR1IE:1;
431     unsigned TMR2IE:1;
432     unsigned CCP1IE:1;
433     unsigned SSPIE:1;
434     unsigned TXIE:1;
435     unsigned RCIE:1;
436     unsigned ADIE:1;
437   } PIE1bits;
438   extern volatile near unsigned char       PIR1;
439   extern volatile near struct {
440     unsigned TMR1IF:1;
441     unsigned TMR2IF:1;
442     unsigned CCP1IF:1;
443     unsigned SSPIF:1;
444     unsigned TXIF:1;
445     unsigned RCIF:1;
446     unsigned ADIF:1;
447   } PIR1bits;
448   extern volatile near unsigned char       IPR1;
449   extern volatile near struct {
450     unsigned TMR1IP:1;
451     unsigned TMR2IP:1;
452     unsigned CCP1IP:1;
453     unsigned SSPIP:1;
454     unsigned TXIP:1;
455     unsigned RCIP:1;
456     unsigned ADIP:1;
457   } IPR1bits;
458   extern volatile near unsigned char       PIE2;
459   extern volatile near union {
```

```
460     struct {
461       unsigned CCP2IE:1;
462       unsigned TMR3IE:1;
463       unsigned LVDIE:1;
464       unsigned BCLIE:1;
465       unsigned EEIE:1;
466       unsigned USBIE:1;
467       unsigned CMIE:1;
468       unsigned OSCFIE:1;
469     };
470     struct {
471       unsigned :2;
472       unsigned HLVDIE:1;
473     };
474   } PIE2bits;
475   extern volatile near unsigned char       PIR2;
476   extern volatile near union {
477     struct {
478       unsigned CCP2IF:1;
479       unsigned TMR3IF:1;
480       unsigned LVDIF:1;
481       unsigned BCLIF:1;
482       unsigned EEIF:1;
483       unsigned USBIF:1;
484       unsigned CMIF:1;
485       unsigned OSCFIF:1;
486     };
487     struct {
488       unsigned :2;
489       unsigned HLVDIF:1;
490     };
491   } PIR2bits;
492   extern volatile near unsigned char       IPR2;
493   extern volatile near union {
494     struct {
495       unsigned CCP2IP:1;
496       unsigned TMR3IP:1;
497       unsigned LVDIP:1;
498       unsigned BCLIP:1;
499       unsigned EEIP:1;
500       unsigned USBIP:1;
501       unsigned CMIP:1;
502       unsigned OSCFIP:1;
503     };
504     struct {
505       unsigned :2;
506       unsigned HLVDIP:1;
507     };
508   } IPR2bits;
509   extern volatile near unsigned char       EECON1;
510   extern volatile near struct {
511     unsigned RD:1;
512     unsigned WR:1;
513     unsigned WREN:1;
514     unsigned WRERR:1;
515     unsigned FREE:1;
516     unsigned :1;
517     unsigned CFGS:1;
518     unsigned EEPGD:1;
519   } EECON1bits;
520   extern volatile near unsigned char       EECON2;
521   extern volatile near unsigned char       EEDATA;
522   extern volatile near unsigned char       EEADR;
523   extern volatile near unsigned char       RCSTA;
524   extern volatile near union {
525     struct {
526       unsigned RX9D:1;
527       unsigned OERR:1;
528       unsigned FERR:1;
529       unsigned ADDEN:1;
530       unsigned CREN:1;
531       unsigned SREN:1;
532       unsigned RX9:1;
533       unsigned SPEN:1;
534     };
535     struct {
536       unsigned :3;
537       unsigned ADEN:1;
538     };
```

```
539   } RCSTAbits;
540   extern volatile near unsigned char       TXSTA;
541   extern volatile near struct {
542     unsigned TX9D:1;
543     unsigned TRMT:1;
544     unsigned BRGH:1;
545     unsigned SENDB:1;
546     unsigned SYNC:1;
547     unsigned TXEN:1;
548     unsigned TX9:1;
549     unsigned CSRC:1;
550   } TXSTAbits;
551   extern volatile near unsigned char       TXREG;
552   extern volatile near unsigned char       RCREG;
553   extern volatile near unsigned char       SPBRG;
554   extern volatile near unsigned char       SPBRGH;
555   extern volatile near unsigned char       T3CON;
556   extern volatile near union {
557     struct {
558       unsigned TMR3ON:1;
559       unsigned TMR3CS:1;
560       unsigned T3SYNC:1;
561       unsigned T3CCP1:1;
562       unsigned T3CKPS0:1;
563       unsigned T3CKPS1:1;
564       unsigned T3CCP2:1;
565       unsigned RD16:1;
566     };
567     struct {
568       unsigned :2;
569       unsigned T3NSYNC:1;
570     };
571     struct {
572       unsigned :2;
573       unsigned NOT_T3SYNC:1;
574     };
575   } T3CONbits;
576   extern volatile near unsigned char       TMR3L;
577   extern volatile near unsigned char       TMR3H;
578   extern volatile near unsigned char       CMCON;
579   extern volatile near struct {
580     unsigned CM0:1;
581     unsigned CM1:1;
582     unsigned CM2:1;
583     unsigned CIS:1;
584     unsigned C1INV:1;
585     unsigned C2INV:1;
586     unsigned C1OUT:1;
587     unsigned C2OUT:1;
588   } CMCONbits;
589   extern volatile near unsigned char       CVRCON;
590   extern volatile near union {
591     struct {
592       unsigned CVR0:1;
593       unsigned CVR1:1;
594       unsigned CVR2:1;
595       unsigned CVR3:1;
596       unsigned CVREF:1;
597       unsigned CVRR:1;
598       unsigned CVROE:1;
599       unsigned CVREN:1;
600     };
601     struct {
602       unsigned :4;
603       unsigned CVRSS:1;
604     };
605   } CVRCONbits;
606   extern volatile near unsigned char       CCP1AS;
607   extern volatile near struct {
608     unsigned :2;
609     unsigned PSSAC0:1;
610     unsigned PSSAC1:1;
611     unsigned ECCPAS0:1;
612     unsigned ECCPAS1:1;
613     unsigned ECCPAS2:1;
614     unsigned ECCPASE:1;
615   } CCP1ASbits;
616   extern volatile near unsigned char       ECCP1AS;
617   extern volatile near struct {
```

```
618     unsigned :2;
619     unsigned PSSAC0:1;
620     unsigned PSSAC1:1;
621     unsigned ECCPAS0:1;
622     unsigned ECCPAS1:1;
623     unsigned ECCPAS2:1;
624     unsigned ECCPASE:1;
625   } ECCP1ASbits;
626   extern volatile near unsigned char       CCP1DEL;
627   extern volatile near struct {
628     unsigned :7;
629     unsigned PRSEN:1;
630   } CCP1DELbits;
631   extern volatile near unsigned char       ECCP1DEL;
632   extern volatile near struct {
633     unsigned :7;
634     unsigned PRSEN:1;
635   } ECCP1DELbits;
636   extern volatile near unsigned char       BAUDCON;
637   extern volatile near union {
638     struct {
639       unsigned ABDEN:1;
640       unsigned WUE:1;
641       unsigned :1;
642       unsigned BRG16:1;
643       unsigned SCKP:1;
644       unsigned :1;
645       unsigned RCIDL:1;
646       unsigned ABDOVF:1;
647     };
648     struct {
649       unsigned :4;
650       unsigned TXCKP:1;
651       unsigned RXDTP:1;
652       unsigned RCMT:1;
653     };
654   } BAUDCONbits;
655   extern volatile near unsigned char       CCP2CON;
656   extern volatile near struct {
657     unsigned CCP2M0:1;
658     unsigned CCP2M1:1;
659     unsigned CCP2M2:1;
660     unsigned CCP2M3:1;
661     unsigned DC2B0:1;
662     unsigned DC2B1:1;
663   } CCP2CONbits;
664   extern volatile near unsigned            CCPR2;
665   extern volatile near unsigned char       CCPR2L;
666   extern volatile near unsigned char       CCPR2H;
667   extern volatile near unsigned char       CCP1CON;
668   extern volatile near struct {
669     unsigned CCP1M0:1;
670     unsigned CCP1M1:1;
671     unsigned CCP1M2:1;
672     unsigned CCP1M3:1;
673     unsigned DC1B0:1;
674     unsigned DC1B1:1;
675   } CCP1CONbits;
676   extern volatile near unsigned            CCPR1;
677   extern volatile near unsigned char       CCPR1L;
678   extern volatile near unsigned char       CCPR1H;
679   extern volatile near unsigned char       ADCON2;
680   extern volatile near struct {
681     unsigned ADCS0:1;
682     unsigned ADCS1:1;
683     unsigned ADCS2:1;
684     unsigned ACQT0:1;
685     unsigned ACQT1:1;
686     unsigned ACQT2:1;
687     unsigned :1;
688     unsigned ADFM:1;
689   } ADCON2bits;
690   extern volatile near unsigned char       ADCON1;
691   extern volatile near struct {
692     unsigned PCFG0:1;
693     unsigned PCFG1:1;
694     unsigned PCFG2:1;
695     unsigned PCFG3:1;
696     unsigned VCFG0:1;
```

```
697     unsigned VCFG1:1;
698   } ADCON1bits;
699   extern volatile near unsigned char       ADCON0;
700   extern volatile near union {
701     struct {
702       unsigned ADON:1;
703       unsigned GO_DONE:1;
704       unsigned CHS0:1;
705       unsigned CHS1:1;
706       unsigned CHS2:1;
707       unsigned CHS3:1;
708     };
709     struct {
710       unsigned :1;
711       unsigned DONE:1;
712     };
713     struct {
714       unsigned :1;
715       unsigned GO:1;
716     };
717     struct {
718       unsigned :1;
719       unsigned NOT_DONE:1;
720     };
721   } ADCON0bits;
722   extern volatile near unsigned            ADRES;
723   extern volatile near unsigned char       ADRESL;
724   extern volatile near unsigned char       ADRESH;
725   extern volatile near unsigned char       SSPCON2;
726   extern volatile near struct {
727     unsigned SEN:1;
728     unsigned RSEN:1;
729     unsigned PEN:1;
730     unsigned RCEN:1;
731     unsigned ACKEN:1;
732     unsigned ACKDT:1;
733     unsigned ACKSTAT:1;
734     unsigned GCEN:1;
735   } SSPCON2bits;
736   extern volatile near unsigned char       SSPCON1;
737   extern volatile near struct {
738     unsigned SSPM0:1;
739     unsigned SSPM1:1;
740     unsigned SSPM2:1;
741     unsigned SSPM3:1;
742     unsigned CKP:1;
743     unsigned SSPEN:1;
744     unsigned SSPOV:1;
745     unsigned WCOL:1;
746   } SSPCON1bits;
747   extern volatile near unsigned char       SSPSTAT;
748   extern volatile near union {
749     struct {
750       unsigned BF:1;
751       unsigned UA:1;
752       unsigned R_W:1;
753       unsigned S:1;
754       unsigned P:1;
755       unsigned D_A:1;
756       unsigned CKE:1;
757       unsigned SMP:1;
758     };
759     struct {
760       unsigned :2;
761       unsigned I2C_READ:1;
762       unsigned I2C_START:1;
763       unsigned I2C_STOP:1;
764       unsigned I2C_DAT:1;
765     };
766     struct {
767       unsigned :2;
768       unsigned NOT_W:1;
769       unsigned :2;
770       unsigned NOT_A:1;
771     };
772     struct {
773       unsigned :2;
774       unsigned NOT_WRITE:1;
775       unsigned :2;
```

```
776       unsigned NOT_ADDRESS:1;
777     };
778     struct {
779       unsigned :2;
780       unsigned READ_WRITE:1;
781       unsigned :2;
782       unsigned DATA_ADDRESS:1;
783     };
784     struct {
785       unsigned :2;
786       unsigned R:1;
787       unsigned :2;
788       unsigned D:1;
789     };
790   } SSPSTATbits;
791   extern volatile near unsigned char       SSPADD;
792   extern volatile near unsigned char       SSPBUF;
793   extern volatile near unsigned char       T2CON;
794   extern volatile near struct {
795     unsigned T2CKPS0:1;
796     unsigned T2CKPS1:1;
797     unsigned TMR2ON:1;
798     unsigned T2OUTPS0:1;
799     unsigned T2OUTPS1:1;
800     unsigned T2OUTPS2:1;
801     unsigned T2OUTPS3:1;
802   } T2CONbits;
803   extern volatile near unsigned char       PR2;
804   extern volatile near unsigned char       TMR2;
805   extern volatile near unsigned char       T1CON;
806   extern volatile near union {
807     struct {
808       unsigned TMR1ON:1;
809       unsigned TMR1CS:1;
810       unsigned T1SYNC:1;
811       unsigned T1OSCEN:1;
812       unsigned T1CKPS0:1;
813       unsigned T1CKPS1:1;
814       unsigned T1RUN:1;
815       unsigned RD16:1;
816     };
817     struct {
818       unsigned :2;
819       unsigned NOT_T1SYNC:1;
820     };
821   } T1CONbits;
822   extern volatile near unsigned char       TMR1L;
823   extern volatile near unsigned char       TMR1H;
824   extern volatile near unsigned char       RCON;
825   extern volatile near union {
826     struct {
827       unsigned NOT_BOR:1;
828       unsigned NOT_POR:1;
829       unsigned NOT_PD:1;
830       unsigned NOT_TO:1;
831       unsigned NOT_RI:1;
832       unsigned :1;
833       unsigned SBOREN:1;
834       unsigned NOT_IPEN:1;
835     };
836     struct {
837       unsigned BOR:1;
838       unsigned POR:1;
839       unsigned PD:1;
840       unsigned TO:1;
841       unsigned RI:1;
842       unsigned :2;
843       unsigned IPEN:1;
844     };
845   } RCONbits;
846   extern volatile near unsigned char       WDTCON;
847   extern volatile near union {
848     struct {
849       unsigned SWDTEN:1;
850     };
851     struct {
852       unsigned SWDTE:1;
853     };
854   } WDTCONbits;
```

```
extern volatile near unsigned char       HLVDCON;
extern volatile near union {
  struct {
    unsigned LVDL0:1;
    unsigned LVDL1:1;
    unsigned LVDL2:1;
    unsigned LVDL3:1;
    unsigned LVDEN:1;
    unsigned IRVST:1;
  };
  struct {
    unsigned LVV0:1;
    unsigned LVV1:1;
    unsigned LVV2:1;
    unsigned LVV3:1;
    unsigned :1;
    unsigned BGST:1;
  };
  struct {
    unsigned HLVDL0:1;
    unsigned HLVDL1:1;
    unsigned HLVDL2:1;
    unsigned HLVDL3:1;
    unsigned HLVDEN:1;
    unsigned :2;
    unsigned VDIRMAG:1;
  };
  struct {
    unsigned :5;
    unsigned IVRST:1;
  };
} HLVDCONbits;
extern volatile near unsigned char       LVDCON;
extern volatile near union {
  struct {
    unsigned LVDL0:1;
    unsigned LVDL1:1;
    unsigned LVDL2:1;
    unsigned LVDL3:1;
    unsigned LVDEN:1;
    unsigned IRVST:1;
  };
  struct {
    unsigned LVV0:1;
    unsigned LVV1:1;
    unsigned LVV2:1;
    unsigned LVV3:1;
    unsigned :1;
    unsigned BGST:1;
  };
  struct {
    unsigned HLVDL0:1;
    unsigned HLVDL1:1;
    unsigned HLVDL2:1;
    unsigned HLVDL3:1;
    unsigned HLVDEN:1;
    unsigned :2;
    unsigned VDIRMAG:1;
  };
  struct {
    unsigned :5;
    unsigned IVRST:1;
  };
} LVDCONbits;
extern volatile near unsigned char       OSCCON;
extern volatile near union {
  struct {
    unsigned SCS0:1;
    unsigned SCS1:1;
    unsigned IOFS:1;
    unsigned OSTS:1;
    unsigned IRCF0:1;
    unsigned IRCF1:1;
    unsigned IRCF2:1;
    unsigned IDLEN:1;
  };
  struct {
    unsigned :2;
    unsigned FLTS:1;
```

```
934       };
935     } OSCCONbits;
936     extern volatile near unsigned char       T0CON;
937     extern volatile near struct {
938       unsigned T0PS0:1;
939       unsigned T0PS1:1;
940       unsigned T0PS2:1;
941       unsigned PSA:1;
942       unsigned T0SE:1;
943       unsigned T0CS:1;
944       unsigned T08BIT:1;
945       unsigned TMR0ON:1;
946     } T0CONbits;
947     extern volatile near unsigned char       TMR0L;
948     extern volatile near unsigned char       TMR0H;
949     extern          near unsigned char       STATUS;
950     extern          near struct {
951       unsigned C:1;
952       unsigned DC:1;
953       unsigned Z:1;
954       unsigned OV:1;
955       unsigned N:1;
956     } STATUSbits;
957     extern          near unsigned            FSR2;
958     extern          near unsigned char       FSR2L;
959     extern          near unsigned char       FSR2H;
960     extern volatile near unsigned char       PLUSW2;
961     extern volatile near unsigned char       PREINC2;
962     extern volatile near unsigned char       POSTDEC2;
963     extern volatile near unsigned char       POSTINC2;
964     extern          near unsigned char       INDF2;
965     extern          near unsigned char       BSR;
966     extern          near unsigned            FSR1;
967     extern          near unsigned char       FSR1L;
968     extern          near unsigned char       FSR1H;
969     extern volatile near unsigned char       PLUSW1;
970     extern volatile near unsigned char       PREINC1;
971     extern volatile near unsigned char       POSTDEC1;
972     extern volatile near unsigned char       POSTINC1;
973     extern          near unsigned char       INDF1;
974     extern          near unsigned char       WREG;
975     extern          near unsigned            FSR0;
976     extern          near unsigned char       FSR0L;
977     extern          near unsigned char       FSR0H;
978     extern volatile near unsigned char       PLUSW0;
979     extern volatile near unsigned char       PREINC0;
980     extern volatile near unsigned char       POSTDEC0;
981     extern volatile near unsigned char       POSTINC0;
982     extern          near unsigned char       INDF0;
983     extern volatile near unsigned char       INTCON3;
984     extern volatile near union {
985       struct {
986         unsigned INT1IF:1;
987         unsigned INT2IF:1;
988         unsigned :1;
989         unsigned INT1IE:1;
990         unsigned INT2IE:1;
991         unsigned :1;
992         unsigned INT1IP:1;
993         unsigned INT2IP:1;
994       };
995       struct {
996         unsigned INT1F:1;
997         unsigned INT2F:1;
998         unsigned :1;
999         unsigned INT1E:1;
1000        unsigned INT2E:1;
1001        unsigned :1;
1002        unsigned INT1P:1;
1003        unsigned INT2P:1;
1004      };
1005    } INTCON3bits;
1006    extern volatile near unsigned char       INTCON2;
1007    extern volatile near union {
1008      struct {
1009        unsigned RBIP:1;
1010        unsigned :1;
1011        unsigned TMR0IP:1;
1012        unsigned :1;
```

```
1013        unsigned INTEDG2:1;
1014        unsigned INTEDG1:1;
1015        unsigned INTEDG0:1;
1016        unsigned NOT_RBPU:1;
1017      };
1018      struct {
1019        unsigned :2;
1020        unsigned T0IP:1;
1021        unsigned :4;
1022        unsigned RBPU:1;
1023      };
1024    } INTCON2bits;
1025    extern volatile near unsigned char       INTCON;
1026    extern volatile near union {
1027      struct {
1028        unsigned RBIF:1;
1029        unsigned INT0IF:1;
1030        unsigned TMR0IF:1;
1031        unsigned RBIE:1;
1032        unsigned INT0IE:1;
1033        unsigned TMR0IE:1;
1034        unsigned PEIE:1;
1035        unsigned GIE:1;
1036      };
1037      struct {
1038        unsigned :1;
1039        unsigned INT0F:1;
1040        unsigned T0IF:1;
1041        unsigned :1;
1042        unsigned INT0E:1;
1043        unsigned T0IE:1;
1044        unsigned GIEL:1;
1045        unsigned GIEH:1;
1046      };
1047    } INTCONbits;
1048    extern          near unsigned            PROD;
1049    extern          near unsigned char       PRODL;
1050    extern          near unsigned char       PRODH;
1051    extern volatile near unsigned char       TABLAT;
1052    extern volatile near unsigned short long TBLPTR;
1053    extern volatile near unsigned char       TBLPTRL;
1054    extern volatile near unsigned char       TBLPTRH;
1055    extern volatile near unsigned char       TBLPTRU;
1056    extern volatile near unsigned short long PC;
1057    extern volatile near unsigned char       PCL;
1058    extern volatile near unsigned char       PCLATH;
1059    extern volatile near unsigned char       PCLATU;
1060    extern volatile near unsigned char       STKPTR;
1061    extern volatile near struct {
1062      unsigned STKPTR0:1;
1063      unsigned STKPTR1:1;
1064      unsigned STKPTR2:1;
1065      unsigned STKPTR3:1;
1066      unsigned STKPTR4:1;
1067      unsigned :1;
1068      unsigned STKUNF:1;
1069      unsigned STKFUL:1;
1070    } STKPTRbits;
1071    extern          near unsigned short long TOS;
1072    extern          near unsigned char       TOSL;
1073    extern          near unsigned char       TOSH;
1074    extern          near unsigned char       TOSU;
1075
1076
1077    /*-----------------------------------------------------------------------
1078     * Some useful defines for inline assembly stuff
1079     *----------------------------------------------------------------------*/
1080    #define ACCESS 0
1081    #define BANKED 1
1082
1083    /*-----------------------------------------------------------------------
1084     * Some useful macros for inline assembly stuff
1085     *----------------------------------------------------------------------*/
1086    #define Nop()    {_asm nop _endasm}
1087    #define ClrWdt() {_asm clrwdt _endasm}
1088    #define Sleep()  {_asm sleep _endasm}
1089    #define Reset()  {_asm reset _endasm}
1090
1091    #define Rlcf(f,dest,access)  {_asm movlb f rlcf f,dest,access _endasm}
```

```
1092  #define Rlncf(f,dest,access) {_asm movlb f rlncf f,dest,access _endasm}
1093  #define Rrcf(f,dest,access)  {_asm movlb f rrcf f,dest,access _endasm}
1094  #define Rrncf(f,dest,access) {_asm movlb f rrncf f,dest,access _endasm}
1095  #define Swapf(f,dest,access) {_asm movlb f swapf f,dest,access _endasm }
1096
1097  /*-----------------------------------------------------------------------
1098   * A fairly inclusive set of registers to save for interrupts.
1099   * These are locations which are commonly used by the compiler.
1100   *-----------------------------------------------------------------------*/
1101  #define INTSAVELOCS TBLPTR, TABLAT, PROD
1102
1103
1104  #endif
1105
```

E.2. Programa de comunicación via MiWi

Por último, en esta sección se expondrá el código completo del programa de comunicación via MiWi:

```
/*************************************************************************/
/*Programa realizado por:
*/ /*
*/ /*         - Enrique Holgado de Frutos
*/ /*         - Francisco Ramos de la Flor
*/ /*
*/
/*************************************************************************/
/*                                                                       */
/*   Este programa tiene como objetivo la comunicación inalámbrica con   */
/*el coordinador de la red. Esta red ha sido previamente creada por el coor-*/
/*dinador en MiWi. Se van a monitorizar 4 sensores:                      */
/*         · 2 Sensores de contacto                                     */
/*         · 2 Sensores IR                                              */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*                                                                       */
/*   Fecha: 20-Octubre-2009 Versión del programa: 3.0                    */
/*                                                                       */
/*   Mejoras con respecto a las versiones anteriores: Se pueden monitorizar */
/*4 sensores de manera simultánea.                                       */
/*                                                                       */
/*************************************************************************/
/*************************************************************************/
/*                                                                       */
/*                     Bajo la lincencia de Microchip.                   */
/*                                                                       */
/*************************************************************************/
/* FileName:        main.c                                               */
/* Dependencies:                                                         */
/* Processor:       PIC18F                                               */
/* Compiler:        C18 02.20.00 or higher                               */
/* Linker:          MPLINK 03.40.00 or higher                            */
/* Company:         Microchip Technology Incorporated                    */
/*************************************************************************/
/* Software License Agreement                                            */
/*************************************************************************/
/* Copyright © 2007-2008 Microchip Technology Inc.  All rights reserved.    */
/*                                                                       */
/* Microchip licenses to you the right to use, modify, copy and distribute  */
/* Software only when embedded on a Microchip microcontroller or digital    */
/* signal controller and used with a Microchip radio frequency transceiver, */
/* which are integrated into your product or third party product (pursuant  */
/* to the terms in the accompanying license agreement).                  */
/*                                                                       */
/* You should refer to the license agreement accompanying this Software for */
/* additional information regarding your rights and obligations.         */
/*                                                                       */
/* SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY  */
/* KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY    */
/* WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A   */
/* PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE   */
/* LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,     */
/* CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY    */
/* DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO   */
/* ANY INCIDENTAL,SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES,  */
/* LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,   */
/* TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT   */
/* NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.          */
/*************************************************************************/
/*                                                                       */
/* Change History:                                                       */
/*  Rev   Date         Author   Description                              */
/*  0.1   11/09/2006   df       Initial revision                         */
/*  1.0   01/09/2007   yfy      Initial release                          */
/*************************************************************************/
// Especificamos las librerias necesarias
#include "Console.h"
```

```
70  #include "MiWi.h"
71  #include "MRF24J40.h"
72  #include "SymbolTime.h"
73  #include "libreria_UCLMin.h"
74
75  // Definicion de la placa que se utiliza
76  #define _UCLMin
77
78  #pragma romdata longAddressLocation = 0x0E ROM unsigned char
79  myLongAddress[8] =
80  {EUI_0,EUI_1,EUI_2,EUI_3,EUI_4,EUI_5,EUI_6,EUI_7}; #pragma romdata
81
82  #ifdef SUPPORT_SECURITY #pragma romdata securityKey = 0x2A ROM
83  unsigned char mySecurityKey[16] = {SECURITY_KEY_00,
84  SECURITY_KEY_01, SECURITY_KEY_02, SECURITY_KEY_03,
85  SECURITY_KEY_04,
86     SECURITY_KEY_05, SECURITY_KEY_06, SECURITY_KEY_07, SECURITY_KEY_08,
87     SECURITY_KEY_09, SECURITY_KEY_10, SECURITY_KEY_11,
88     SECURITY_KEY_12, SECURITY_KEY_13, SECURITY_KEY_14, SECURITY_KEY_15};
89  #pragma romdata #endif
90
91  #if defined(SUPPORT_SECURITY) ROM unsigned char mySecurityLevel =
92  SECURITY_LEVEL; ROM unsigned char myKeySequenceNumber =
93  KEY_SEQUENCE_NUMBER; #endif
94
95  ROM BYTE
96  availableChannels[AVAILABLE_CHANNELS_SIZE]={ALLOWED_CHANNELS}; ROM
97  unsigned char myManufacturerString[]="Company ABC";
98
99
100 #define MAX_PA_OUTPUT 0x00   // 0dB output
101 //all values between 0x00 and 0x1F are valid and
102 //results in a -1.25dB output per unit above 0 #define
103 MIN_PA_OUTPUT 0x1F   // -38.75dB output #define
104 MRF24J40PAOutputAdjust(a) {PHYSetLongRAMAddr(RFCTRL3,(a<<3));}
105
106
107 /********************** FUNCTION PROTOTYPES
108 *************************/ void BoardInit(void);
109
110 #ifdef __PIC32MX__
111
112     #define PIC32MX_SPI1_SDO_SCK_MASK_VALUE      (0x00000140)
113
114     #define PIC32MX_SPI1_SDI_MASK_VALUE          (0x00000080)
115
116     #define PIC32MX_INT1_MASK_VALUE              (0x00000008)
117
118     /* MAX SPI CLOCK FREQ SUPPORTED FOR MIWI TRANSCIEVER */
119     #define MAX_SPI_CLK_FREQ_FOR_MIWI            (20000000)
120
121     #define RCON_SLEEP_MASK (0x8)
122
123     #define RCON_IDLE_MASK  (0x4)
124
125     #define        Sleep()      PowerSaveSleep()
126
127     unsigned char Pushbutton_1_Wakeup_Pressed = 0;
128
129     unsigned char Pushbutton_2_Wakeup_Pressed = 0;
130
131     static void Enable_PB_1_2_Interrupts(void);
132
133 #endif
134
135 /********************** CONSTANTS
136 ************************************/ #define
137 MAX_DISCOVERY_ATTEMPTS 3 #define DEBOUNCE_TIME 0x00000C35
138
139 #if !defined(P2P_ONLY) #if defined(__18CXX) void main(void) #else
140 int main(void) #endif {
141     BYTE i;                         //Creamos las variables necesarias
142     BYTE numDiscoveryAttempts;      //del robot a la red MiWi
143     BOOL requestedSocket;
144     //Creamos las variables que representan si los sensores están activados o no
145     BOOL PUSH_BUTTON_1_pressed;     //Sensor de contacto derecho
146     BOOL PUSH_BUTTON_2_pressed;     //Sensor de contacto izquierdo
147     BOOL PUSH_BUTTON_3_pressed;     //Sensor IR derecho
148     BOOL PUSH_BUTTON_4_pressed;     //Sensor IR derecho
```

```
149
150         //Creamos las variables que representan
151         //el tiempo que los sensores están activados.
152         //Utilizando el mismo orden que para si están presionados
153         TICK PUSH_BUTTON_1_press_time;
154         TICK PUSH_BUTTON_2_press_time;
155         TICK PUSH_BUTTON_3_press_time;
156         TICK PUSH_BUTTON_4_press_time;
157         TICK tickDifference;
158         BYTE myFriend = 0xFF;
159
160         requestedSocket = FALSE;
161
162         numDiscoveryAttempts = 0;
163
164        /*Inicializando la placa despues de la elección de la placa que esté conectada*/
165
166         ConsoleInit();
167         BoardInit();
168         MiWiInit();
169         ConfigurarUCLMin();        //Configuración de entradas, salidas y temporizador
170         INTCONbits.GIEH = 1;
171
172         if(PUSH_BUTTON_1 == 0)
173         {
174             DumpNetworkTable();
175             while(PUSH_BUTTON_1==0){}
176             RejoinNetwork();
177         }
178
179
180
181         while(1)
182         {
183             MiWiTasks();
184
185
186
187             if(MemberOfNetwork())
188             {
189
190
191                 //Si el end device forma parte de una red,
192                 //se ejecuta el siguiente código de recepción de datos
193                 if(RxPacket())
194                 {
195                     BYTE *pRxData2;
196                     BYTE i;
197
198                     //Inserte aquí el código de usuario para el
199                     //procesamiento de paquetes según van llegando
200
201                     pRxData2 = pRxData;
202                     *pRxData++;     //take off the seq number
203
204                     switch(*pRxData++)      //report type
205                     {
206                         case USER_REPORT_TYPE:
207                             switch(*pRxData++)      //report id
208                             {
209                                 case LIGHT_REPORT:
210                                     switch(*pRxData++)      //first byte of payload
211                                     {
212                                         case LIGHT_ON:
213                                             LED_2 = 1;
214                                             break;
215                                         case LIGHT_OFF:
216                                             LED_2 = 0;
217                                             break;
218                                         case LIGHT_TOGGLE:
219                                             LED_2 ^= 1;
220                                             ConsolePutROMString((ROM char*)
221                                                 "Received Report to Toggle Light\r\n");
222                                             break;
223                                     }
224                                     break;
225                             }
226                             break;
227                         case MIWI_STACK_REPORT_TYPE:
```

```
228                         switch(*pRxData)
229                         {
230                             case ACK_REPORT_TYPE:
231                                 //ConsolePutROMString((ROM char*)
232                                         "Got MiWi ACK for my packet\r\n");
233                                 break;
234                         }
235                         break;
236                     }
237
238                     //need to discard this packet before
239                     //we are able to receive any other packets
240                     DiscardPacket();
241                 }
242
243                 #if defined(SUPPORT_CLUSTER_SOCKETS) || defined(SUPPORT_P2P_SOCKETS)
244                 if(requestedSocket == TRUE)
245                 {
246                     if(OpenSocketComplete())
247                     {
248                         requestedSocket = FALSE;
249
250                         if(OpenSocketSuccessful())
251                         {
252                             LED_1 = 1;
253                             ConsolePutROMString((ROM char*)"Found a socket: ");
254                             myFriend = OpenSocketHandle();
255                             PrintChar(myFriend);
256                             ConsolePutROMString((ROM char*)"\r\n");
257                         }
258                         else
259                         {
260                             LED_1 = 0;
261                             myFriend = 0xFF;
262                             ConsolePutROMString((ROM char*)"socket request failed\r\n");
263                         }
264                     }
265                 }
266                 #endif
267
268
269     ....
270
271
272
273
274
275     ....
276
277                 #ifdef __PIC32MX__
278
279                 if((PUSH_BUTTON_2 == 0) || Pushbutton_2_Wakeup_Pressed)
280
281                 #else
282
283                 if(PUSH_BUTTON_2 == 0)
284
285                 #endif
286                 {
287                     #ifdef __PIC32MX__
288
289                     if((PUSH_BUTTON_2_pressed == FALSE) || Pushbutton_2_Wakeup_Pressed)
290
291                     #else
292
293                     if(PUSH_BUTTON_2_pressed == FALSE)
294
295                     #endif
296                     {
297                         static BYTE transmitMode = 0;
298
299                         PUSH_BUTTON_2_pressed = TRUE;
300
301                         #ifdef __PIC32MX__
302
303                         Pushbutton_2_Wakeup_Pressed = 0;
304
305                         #endif
306
```

```
307
308                     TxPayLoad();
309                     WriteData(USER_REPORT_TYPE);
310                     WriteData(LIGHT_REPORT);
311                     WriteData(LIGHT_TOGGLE);
312
313
314
315                     if( myFriend != 0xFF )  // socket has already been established
316                     {
317                         SendReportByHandle(myFriend, FALSE);
318                         ConsolePutROMString((ROM char*)
319                         "Send Report by Handle(Socket)\r\n");
320                     }
321                     else
322                     {
323                         // if no socket, send report by long or short address
324                         if( (transmitMode++ % 2) == 0 )
325                         {
326                             tempLongAddress[0] = 0x93;
327                             tempLongAddress[1] = 0x78;
328                             tempLongAddress[2] = 0x56;
329                             tempLongAddress[3] = 0x34;
330                             tempLongAddress[4] = 0x12;
331                             tempLongAddress[5] = 0xA3;
332                             tempLongAddress[6] = 0x04;
333                             tempLongAddress[7] = 0x00;
334
335                             SendReportByLongAddress(tempLongAddress);
336                             ConsolePutROMString((ROM char*)
337                             "Send Report by Long Address\r\n");
338                         }
339                         else
340                         {
341                             tempShortAddress.Val = 0x0000;
342                             SendReportByShortAddress(myPANID,
343                                                      tempShortAddress, FALSE);
344                             ConsolePutROMString((ROM char*)
345                             "Send Report by Short Address\r\n");
346                         }
347                     }
348                 }
349                 PUSH_BUTTON_2_press_time = TickGet();
350             }
351             else
352             {
353                 TICK t = TickGet();
354
355                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_2_press_time);
356
357                 if(tickDifference.Val > DEBOUNCE_TIME)
358                 {
359                     PUSH_BUTTON_2_pressed = FALSE;
360                 }
361             }
362
363
364         #ifdef __PIC32MX__
365
366             if((PUSH_BUTTON_1 == 0) || Pushbutton_2_Wakeup_Pressed)
367
368             #else
369
370             if(PUSH_BUTTON_1 == 0)
371
372             #endif
373             {
374                 #ifdef __PIC32MX__
375
376                 if((PUSH_BUTTON_1_pressed == FALSE) || Pushbutton_2_Wakeup_Pressed)
377
378                 #else
379
380                 if(PUSH_BUTTON_1_pressed == FALSE)
381
382                 #endif
383                 {
384                     static BYTE transmitMode = 0;
385
```

```
                    PUSH_BUTTON_1_pressed = TRUE;

                    #ifdef __PIC32MX__

                    Pushbutton_1_Wakeup_Pressed = 0;

                    #endif



                    TxPayLoad();
                    WriteData(USER_REPORT_TYPE_1);
                    WriteData(LIGHT_REPORT);
                    WriteData(LIGHT_TOGGLE);



                    if( myFriend != 0xFF )  // socket has already been established
                    {
                        SendReportByHandle(myFriend, FALSE);
                        ConsolePutROMString((ROM char*)
                        "Send Report by Handle(Socket)\r\n");
                    }
                    else
                    {
                        // if no socket, send report by long or short address
                        if( (transmitMode++ % 2) == 0 )
                        {
                            tempLongAddress[0] = 0x93;
                            tempLongAddress[1] = 0x78;
                            tempLongAddress[2] = 0x56;
                            tempLongAddress[3] = 0x34;
                            tempLongAddress[4] = 0x12;
                            tempLongAddress[5] = 0xA3;
                            tempLongAddress[6] = 0x04;
                            tempLongAddress[7] = 0x00;

                            SendReportByLongAddress(tempLongAddress);
                            ConsolePutROMString((ROM char*)
                            "Send Report by Long Address\r\n");
                        }
                        else
                        {
                            tempShortAddress.Val = 0x0000;
                            SendReportByShortAddress(myPANID,
                                                     tempShortAddress, FALSE);
                            ConsolePutROMString((ROM char*)
                            "Send Report by Short Address\r\n");
                        }
                    }
                }
                PUSH_BUTTON_1_press_time = TickGet();
            }
            else
            {
                TICK t = TickGet();

                tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_1_press_time);

                if(tickDifference.Val > DEBOUNCE_TIME)
                {
                    PUSH_BUTTON_1_pressed = FALSE;
                }
            }




#ifdef __PIC32MX__

            if((PUSH_BUTTON_3 == 0) || Pushbutton_2_Wakeup_Pressed)

            #else

            if(PUSH_BUTTON_3 == 0)

            #endif
            {
```

```
465                 #ifdef __PIC32MX__
466
467                 if((PUSH_BUTTON_3_pressed == FALSE) || Pushbutton_3_Wakeup_Pressed)
468
469                 #else
470
471                 if(PUSH_BUTTON_3_pressed == FALSE)
472
473                 #endif
474                 {
475                     static BYTE transmitMode = 0;
476
477                     PUSH_BUTTON_3_pressed = TRUE;
478
479                     #ifdef __PIC32MX__
480
481                     Pushbutton_3_Wakeup_Pressed = 0;
482
483                     #endif
484
485                     TxPayLoad();
486                     WriteData(USER_REPORT_TYPE_2);
487                     WriteData(LIGHT_REPORT);
488                     WriteData(LIGHT_TOGGLE);
489
490                     if( myFriend != 0xFF )  // socket has already been established
491                     {
492                         SendReportByHandle(myFriend, FALSE);
493                         ConsolePutROMString((ROM char*)
494                                                 "Send Report by Handle(Socket)\r\n");
495                     }
496                     else
497                     {
498                         // if no socket, send report by long or short address
499                         if( (transmitMode++ % 2) == 0 )
500                         {
501                             tempLongAddress[0] = 0x93;
502                             tempLongAddress[1] = 0x78;
503                             tempLongAddress[2] = 0x56;
504                             tempLongAddress[3] = 0x34;
505                             tempLongAddress[4] = 0x12;
506                             tempLongAddress[5] = 0xA3;
507                             tempLongAddress[6] = 0x04;
508                             tempLongAddress[7] = 0x00;
509
510                             SendReportByLongAddress(tempLongAddress);
511                             ConsolePutROMString((ROM char*)
512                             "Send Report by Long Address\r\n");
513                         }
514                         else
515                         {
516                             tempShortAddress.Val = 0x0000;
517                             SendReportByShortAddress(myPANID,
518                                                     tempShortAddress, FALSE);
519                             ConsolePutROMString((ROM char*)
520                             "Send Report by Short Address\r\n");
521                         }
522                     }
523                 }
524                 PUSH_BUTTON_3_press_time = TickGet();
525             }
526             else
527             {
528                 TICK t = TickGet();
529
530                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_3_press_time);
531
532                 if(tickDifference.Val > DEBOUNCE_TIME)
533                 {
534                     PUSH_BUTTON_3_pressed = FALSE;
535                 }
536             }
537
538
539         #ifdef __PIC32MX__
540
541             if((PUSH_BUTTON_4 == 0) || Pushbutton_4_Wakeup_Pressed)
542
543             #else
```

```
544
545             if(PUSH_BUTTON_4 == 0)
546
547             #endif
548             {
549                 #ifdef __PIC32MX__
550
551                 if((PUSH_BUTTON_4_pressed == FALSE) || Pushbutton_4_Wakeup_Pressed)
552
553                 #else
554
555                 if(PUSH_BUTTON_4_pressed == FALSE)
556
557                 #endif
558                 {
559                     static BYTE transmitMode = 0;
560
561                     PUSH_BUTTON_4_pressed = TRUE;
562
563                     #ifdef __PIC32MX__
564
565                     Pushbutton_4_Wakeup_Pressed = 0;
566
567                     #endif
568
569                     TxPayLoad();
570                     WriteData(USER_REPORT_TYPE_3);
571                     WriteData(LIGHT_REPORT);
572                     WriteData(LIGHT_TOGGLE);
573
574                     if( myFriend != 0xFF )  // socket has already been established
575                     {
576                         SendReportByHandle(myFriend, FALSE);
577                         ConsolePutROMString((ROM char*)
578                         "Send Report by Handle(Socket)\r\n");
579                     }
580                     else
581                     {
582                         // if no socket, send report by long or short address
583                         if( (transmitMode++ % 2) == 0 )
584                         {
585                             tempLongAddress[0] = 0x93;
586                             tempLongAddress[1] = 0x78;
587                             tempLongAddress[2] = 0x56;
588                             tempLongAddress[3] = 0x34;
589                             tempLongAddress[4] = 0x12;
590                             tempLongAddress[5] = 0xA3;
591                             tempLongAddress[6] = 0x04;
592                             tempLongAddress[7] = 0x00;
593
594                             SendReportByLongAddress(tempLongAddress);
595                             ConsolePutROMString((ROM char*)
596                             "Send Report by Long Address\r\n");
597                         }
598                         else
599                         {
600                             tempShortAddress.Val = 0x0000;
601                             SendReportByShortAddress(myPANID,
602                                                      tempShortAddress, FALSE);
603                             ConsolePutROMString((ROM char*)
604                             "Send Report by Short Address\r\n");
605                         }
606                     }
607                 }
608                 PUSH_BUTTON_4_press_time = TickGet();
609             }
610             else
611             {
612                 TICK t = TickGet();
613
614                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_4_press_time);
615
616                 if(tickDifference.Val > DEBOUNCE_TIME)
617                 {
618                     PUSH_BUTTON_4_pressed = FALSE;
619                 }
620             }
621
622 ....
```

```
623
624
625
626
627
628   ...
629
630
631             //Si el dispositivo es un END DEVICE se realiza el siguiente código
632             #if defined(I_AM_RFD)
633
634                 #ifdef __PIC32MX__
635
636                 if(CheckForDataComplete()&&(PUSH_BUTTON_1!= 0)&& (PUSH_BUTTON_2!= 0))
637
638                 #else
639
640                 if(CheckForDataComplete())
641
642                 #endif
643                 {
644                     MRF24J40Sleep();
645                     while( !ConsoleIsPutReady() );
646
647                     #if defined(__18CXX)
648                         WDTCONbits.SWDTEN = 1;      //enable the WDT to wake me up
649                         INTCONbits.RBIF = 0;
650                         INTCONbits.RBIE = 1;
651                     #elif defined(__dsPIC30F__) || defined(__dsPIC33F__)
652                             || defined(__PIC24F__) || defined(__PIC24H__)
653                         ClrWdt();
654                         RCONbits.SWDTEN = 1;        //enable the WDT to wake me up
655                         IEC1bits.CNIE = 1;
656                     #elif defined(__PIC32MX__)
657                 /* Enable Change Notice Push Button 1 & 2 interrupt,
658                    so that on pushing button 1 or 2 the controller wakes up.
659                    Disabling of Change notice module &
660                    interrupt is done in ISR */
661
662                         Enable_PB_1_2_Interrupts();
663
664                         EnableWDT();   // Enable Watchdog
665                         ClearWDT();   // Clear the Watchdog Timer
666                     #endif
667
668                     Sleep();            //goto sleep
669
670                     #if defined(__18CXX)
671                         INTCONbits.RBIE = 0;
672                         WDTCONbits.SWDTEN = 0;      //disable WDT
673                     #elif defined(__dsPIC30F__) || defined(__dsPIC33F__)
674                             || defined(__PIC24F__) || defined(__PIC24H__)
675                         RCONbits.SWDTEN = 0;        //disable WDT
676                         IEC1bits.CNIE = 0;
677                     #elif defined(__PIC32MX__)
678                         RCONCLR = RCON_SLEEP_MASK | RCON_IDLE_MASK;
679                         // clear the Sleep and Idle bits
680                         /* Disable Watchdog Timer */
681                         DisableWDT();
682                     #endif
683
684                     MRF24J40Wake();
685
686                     CheckForData();     //when I wake up do a data request
687                 }
688             #endif
689
690             //END OF THE CODE AFTER YOU BECAME A MEMBER
691         }
692         else
693         {
694             #if !defined(P2P_ONLY)
695             //If I don't have a network yet and
696             //I am not currently trying to join a network
697             if((!SearchingForNetworks()) &&
698                 (!AttemptingToJoinNetwork()) && (!AttemptingToRejoinNetwork()))
699             {
700                 //I am not actively searching for a network
701                 //so lets choise one from the list
```

```
702                     BYTE handleOfBestNetwork;
703                     BYTE i;
704                     //I will show the example of picking a network based on RSSI
705                     BYTE maxRSSI;
706
707                     //initialize the handle to none(0xFF)
708                     handleOfBestNetwork = 0xFF;
709                     //RSSI example
710                     maxRSSI = 0x00;
711
712                     for(i=0;i<NETWORK_TABLE_SIZE;i++)
713                     {
714                     if(networkStatus[i].bits.isValid)
715                     {
716                         if(networkStatus[i].bits.NeighborOrNetwork == NETWORK)
717                         {
718                         //make your PAN choice here
719                            if(networkTable[i].info.networkInfo.Protocol ==
720                                                                   MIWI_PROTOCOL_ID)
721                            {
722                            //make sure they are allowing joining
723                               if(networkTable[i].info.networkInfo.flags.
724                                                             bits.associationPermit == 1)
725                               {
726                               //first I want to make sure it is a MiWi network
727                                   if(networkTable[i].info.networkInfo.
728                                                             sampleRSSI >= maxRSSI)
729                                   {
730                                       handleOfBestNetwork = i;
731                                       maxRSSI = networkTable[i].
732                                                       info.networkInfo.sampleRSSI;
733                                   }
734                               }
735                            }
736                     }
737                     }
738                 }
739
740                     //now that I picked a network let me join to it
741                     if(handleOfBestNetwork == 0xFF)
742                     {
743                         //I couldn't find a suitable network
744                         ConsolePutROMString((ROM char*)"Trying to find a suitable network\r\n");
745
746                         if(numDiscoveryAttempts++ > MAX_DISCOVERY_ATTEMPTS)
747                         {
748                             #if defined(I_AM_COORDINATOR_CAPABLE)
749                                 //form a network with the specified PANID
750                                 FormNetwork(0xFFFF);
751                                 {
752                                     WORD i;
753
754                                     LED_1 = 1;
755
756                                     for(i=0;i<20000;i++)
757                                     {
758                                     }
759
760                                     LED_1 = 0;
761
762                                     for(i=0;i<20000;i++)
763                                     {
764                                     }
765
766                                     LED_1 = 1;
767
768                                     for(i=0;i<20000;i++)
769                                     {
770                                     }
771
772                                     LED_1 = 0;
773                                 }
774                             #endif
775
776                             #if !defined(P2P_ONLY)
777                             //clear all of the network entries
778                                 ClearNetworkTable(CLEAR_NETWORKS);
779                             #endif
780
```

```
                        }
                        else
                        {
                            #if !defined(P2P_ONLY)
                                //clear all of the network entries
                                ClearNetworkTable(CLEAR_NETWORKS);
                            #endif

                            //and start a new search
                            DiscoverNetworks();
                        }
                    }
                    else
                    {
                        #if !defined(P2P_ONLY)
                            //I found a network I would like to join
                            //Join the network that you found to be the best
                            ConsolePutROMString((ROM char*)"Trying to join network: ");
                            PrintChar(handleOfBestNetwork);
                            ConsolePutROMString((ROM char*)"\r\n");
                            JoinNetwork(handleOfBestNetwork);

                            {
                                WORD i;

                                LED_2 = 1;

                                for(i=0;i<20000;i++)
                                {
                                }

                                LED_2 = 0;

                                for(i=0;i<20000;i++)
                                {
                                }

                                LED_2 = 1;

                                for(i=0;i<20000;i++)
                                {
                                }

                                LED_2 = 0;
                            }
                        #endif
                    }

                    if(MemberOfNetwork())
                    {
                        requestedSocket = TRUE;
                        #if defined(SUPPORT_CLUSTER_SOCKETS)
                            ConsolePutROMString((ROM char *)"Opening Cluster Socket\r\n");
                            OpenSocket(CLUSTER_SOCKET);
                        #elif defined(SUPPORT_P2P_SOCKETS)
                            ConsolePutROMString((ROM char*)"Opening P2P Socket\r\n");
                            OpenSocket(P2P_SOCKET);
                        #endif
                }


                }
                #endif //P2P_ONLY
                //if I am searching for a network then
                //I will leave it alone so I can continue searching
                //or if I am trying to join a network then
                //I will let that join proccess finish
            }
        }
    }

    #else

    /*******************************************************************************
    *
    *                       DEMO code for P2P only devices
    *
    *******************************************************************************/
    //DEMO for P2P Only devices
```

```
860  void main(void) {
861      BOOL PUSH_BUTTON_1_pressed;
862      TICK PUSH_BUTTON_1_press_time, tickDifference;
863      BOOL FoundSocket;
864      BYTE myFriend;
865
866      ConsoleInit();
867      BoardInit();
868      MiWiInit();
869
870      #if defined(PICDEMZ)
871      INTCONbits.GIEH = 1;
872      #elif defined(EXPLORER16)
873      #else
874          #error "Unknown board.  Please initialize board as required."
875      #endif
876
877      SetChannel(CHANNEL_12);
878
879      FoundSocket = FALSE;
880
881      while(1)
882      {
883          MiWiTasks();
884
885          if(FoundSocket == TRUE)
886          {
887              if(RxPacket())
888              {
889                  BYTE *pRxData2;
890                  BYTE i;
891
892                  //insert user code here for processing packets as they come in.
893
894                  pRxData2 = pRxData;
895                  *pRxData++;     //take off the seq number
896
897                  switch(*pRxData++)      //report type
898                  {
899                      case USER_REPORT_TYPE:
900                          switch(*pRxData++)      //report id
901                          {
902                              case LIGHT_REPORT:
903                                  switch(*pRxData++)      //first byte of payload
904                                  {
905                                      case LIGHT_ON:
906                                          LED_2 = 1;
907                                          break;
908                                      case LIGHT_OFF:
909                                          LED_2 = 0;
910                                          break;
911                                      case LIGHT_TOGGLE:
912                                          LED_2 ^= 1;
913                                          break;
914                                  }
915                                  break;
916                          }
917                          break;
918                      case MIWI_STACK_REPORT_TYPE:
919                          switch(*pRxData)
920                          {
921                              case ACK_REPORT_TYPE:
922                                  //ConsolePutROMString((ROM char*)
923                                              "Got MiWi ACK for my packet\r\n");
924                                  break;
925                          }
926                          break;
927                  }
928
929                  //need to discard this packet before
930                  //we are able to receive any other packets
931                  DiscardPacket();
932              }
933
934              if(PUSH_BUTTON_1 == 0)
935              {
936                  if(PUSH_BUTTON_1_pressed == FALSE)
937                  {
938                      PUSH_BUTTON_1_pressed = TRUE;
```

```
939
940                 TxPayLoad();
941                     WriteData(USER_REPORT_TYPE);
942                     WriteData(LIGHT_REPORT);
943                     WriteData(LIGHT_TOGGLE);
944
945                     SendReportByLongAddress(openSocketInfo.LongAddress1);
946
947                 }
948                 PUSH_BUTTON_1_press_time = TickGet();
949             }
950             else
951             {
952                 TICK t = TickGet();
953
954                 tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_1_press_time);
955
956                 if(tickDifference.Val > DEBOUNCE_TIME)
957                 {
958                     PUSH_BUTTON_1_pressed = FALSE;
959                 }
960             }
961
962         }
963         else
964         {
965             if(OpenSocketComplete())
966             {
967                 if(OpenSocketSuccessful())
968                 {
969                     FoundSocket = TRUE;
970                     LED_1 = 1;
971                     ConsolePutROMString((ROM char*)"Found a socket: ");
972                     ConsolePutROMString((ROM char*)"\r\n");
973                 }
974                 else
975                 {
976                     ConsolePutROMString((ROM char*)"sending request\r\n");
977                     //else we need to try again (or for the first time)
978                     OpenSocket(P2P_SOCKET);
979                 }
980             }
981         }
982     }
983 } #endif
984
985 /*******************************************************************
986  * Function:        void BoardInit( void )
987  *
988  * PreCondition:    None
989  *
990  * Input:           None
991  *
992  * Output:          None
993  *
994  * Side Effects:    Board is initialized for MiWi usage
995  *
996  * Overview:        This function configures the board for the PICDEM-z
997  *                  MRF24J40 usage
998  *
999  * Note:            This routine needs to be called before the function
1000 *                  to initialize MiWi stack or any other function that
1001 *                  operates on the stack
1002 *******************************************************************/
1003  #if defined(PICDEMZ) && !defined(__PICC__)
1004 void BoardInit(void) {
1005     WDTCONbits.SWDTEN = 0; //disable WDT
1006
1007     // Switches S2 and S3 are on RB5 and RB4 respectively.
1008     //We want interrupt-on-change
1009     INTCON = 0x00;
1010
1011     // There is no external pull-up resistors on S2 and S3.
1012     //We will use internal pull-ups.
1013     // The MRF24J40 is using INT0 for interrupts
1014     // Enable PORTB internal pullups
1015     INTCON2 = 0x00;
1016     INTCON3 = 0x00;
1017
```

```
1018        // Make PORTB as input - this is the RESET default
1019        TRISB = 0xff;
1020
1021        // Set PORTC control signal direction and initial states
1022        // disable chip select
1023        LATC = 0xfd;
1024
1025        // Set the SPI module for use by Stack
1026        TRISC = 0xD0;
1027
1028        // Set the SPI module
1029        SSPSTAT = 0xC0;
1030        SSPCON1 = 0x20;
1031
1032        // D1 and D2 are on RA0 and RA1 respectively, and CS of TC77 is on RA2.
1033        // Make PORTA as digital I/O.
1034        // The TC77 temp sensor CS is on RA2.
1035        ADCON1 = 0x0F;
1036
1037        // Deselect TC77 (RA2)
1038        LATA = 0x04;
1039
1040        // Make RA0, RA1, RA2 and RA4 as outputs.
1041        TRISA = 0xF8;
1042
1043        PHY_CS = 1;             //deselect the MRF24J40
1044        PHY_CS_TRIS = 0;        //make chip select an output
1045
1046        RFIF = 0;               //clear the interrupt flag
1047
1048        RCONbits.IPEN = 1;
1049
1050        INTCON2bits.INTEDG0 = 0;
1051    } #elif defined(__PICC__) void BoardInit(void) {
1052        // Switches S2 and S3 are on RB5 and RB4 respectively.
1053        //We want interrupt-on-change
1054        INTCON = 0x00;
1055
1056        // Make PORTB as input - this is the RESET default
1057        TRISB = 0xff;
1058
1059        INTEDG = 0;
1060
1061        // Set the SPI module
1062        SSPSTAT = 0xC0;
1063        SSPCON = 0x20;
1064
1065        // D1 and D2 are on RA0 and RA1 respectively,
1066        //and CS of TC77 is on RA2.
1067        // Make PORTA as digital I/O.
1068        // The TC77 temp sensor CS is on RA2.
1069        ADCON1 = 0x0F;
1070
1071        // Make RA0, RA1, RA2 and RA4 as outputs.
1072        TRISA = 0xF8;
1073
1074        TRISC = 0xD6;
1075
1076        PHY_CS = 1;             //deselect the MRF24J40
1077        PHY_CS_TRIS = 0;        //make chip select an output
1078
1079        RFIF = 0;               //clear the interrupt flag
1080
1081        if(RB0 == 0)
1082        {
1083            RFIF = 1;
1084        }
1085    }
1086
1087    //Parte programada con la configuración del hardware de la placa
1088
1089    #elif defined(_UCLMin) void BoardInit(void) {
1090        WDTCONbits.SWDTEN = 0; //disable WDT
1091
1092       // Switches S2 and S3 are on RB5 and RB4 respectively.
1093       //We want interrupt-on-change
1094        INTCON = 0x00;
1095
1096        // Make PORTB as input - this is the RESET default
```

```
1097
1098      TRISA = 0xE4;//11100100
1099      TRISB = 0xC5;//11000101
1100      TRISC = 0x18;//00011000
1101
1102       INTCON2 = 0x00;
1103       INTCON3 = 0x80;   //0b10010000
1104         //bit 7 INT2IP: INT2 External Interrupt Priority bit
1105            //1 = High priority
1106            //0 = Low priority
1107         //bit 6 INT1IP: INT1 External Interrupt Priority bit
1108            //1 = High priority
1109            //0 = Low priority
1110         //bit 5 Unimplemented: Read as '0'
1111         //bit 4 INT2IE: INT2 External Interrupt Enable bit
1112            //1 = Enables the INT2 external interrupt
1113            //0 = Disables the INT2 external interrupt
1114         //bit 3 INT1IE: INT1 External Interrupt Enable bit
1115            //1 = Enables the INT1 external interrupt
1116            //0 = Disables the INT1 external interrupt
1117         //bit 2 Unimplemented: Read as '0'
1118         //bit 1 INT2IF: INT2 External Interrupt Flag bit
1119            //1 = The INT2 external interrupt occurred (must be cleared in software)
1120            //0 = The INT2 external interrupt did not occur
1121         //bit 0 INT1IF: INT1 External Interrupt Flag bit
1122            //1 = The INT1 external interrupt occurred (must be cleared in software)
1123            //0 = The INT1 external interrupt did not occur
1124
1125       // Set the SPI module
1126       SSPSTAT = 0xC0;
1127  //     SSPCON1 = 0x21   ; //Cambiado por que en el 2550 se llama
1128                            //SSPCON1 y no SSPCON como el 4620
1129
1130       SSPCON1 = 0x20; //0b00100010
1131         //bit 7 WCOL: Write Collision Detect bit (Transmit mode only)
1132            //1 = The SSPBUF register is written while it is still
1133            //transmitting the previous word
1134            //(must be cleared in software)
1135            //0 = No collision
1136         //bit 6 SSPOV: Receive Overflow Indicator bit(1)
1137            //SPI Slave mode:
1138            //1 = A new byte is received while the SSPBUF register is
1139            //still holding the previous data. In case of overflow,
1140            //the data in SSPSR is lost. Overflow can only occur in Slave mode.
1141            //The user must read the SSPBUF, even if only transmitting data,
1142            //to avoid setting overflow (must be cleared in software).
1143            //0 = No overflow
1144         //bit 5 SSPEN: Master Synchronous Serial Port Enable bit
1145            //1 = Enables serial port and configures
1146            //SCK, SDO, SDI and SS as serial port pins(2)
1147            //0 = Disables serial port and configures
1148            //these pins as I/O port pins(2)
1149         //bit 4 CKP: Clock Polarity Select bit
1150            //1 = Idle state for clock is a high level
1151            //0 = Idle state for clock is a low level
1152         //bit 3-0 SSPM3:SSPM0: Master Synchronous Serial Port Mode Select bits
1153            //0101 = SPI Slave mode, clock = SCK pin,
1154            //SS pin control disabled, SS can be used as I/O pin(3)
1155            //0100 = SPI Slave mode, clock = SCK pin, SS pin control enabled(3)
1156            //0011 = SPI Master mode, clock = TMR2 output/2(3)
1157            //0010 = SPI Master mode, clock = FOSC/64(3)
1158            //0001 = SPI Master mode, clock = FOSC/16(3)
1159            //0000 = SPI Master mode, clock = FOSC/4(3)
1160
1161       // D1 and D2 are on RA0 and RA1 respectively, and CS of TC77 is on RA2.
1162       // Make PORTA as digital I/O.
1163       // The TC77 temp sensor CS is on RA2.
1164       ADCON1 = 0x0F;
1165
1166       // Make RA0, RA1, RA2 and RA4 as outputs.
1167
1168       PHY_CS = 1;             //deselect the MRF24J40
1169       PHY_CS_TRIS = 0;        //make chip select an output
1170
1171       RFIF = 0;               //clear the interrupt flag
1172
1173
1174
1175       PHY_RESETn = 0;
```

```
1176        PHY_RESETn_TRIS = 0;
1177        PHY_CS = 1;
1178        PHY_CS_TRIS = 0;
1179        PHY_WAKE = 1;
1180        PHY_WAKE_TRIS = 0;
1181
1182        LED_1_TRIS = 0;
1183        LED_2_TRIS = 0;
1184        PUSH_BUTTON_1_TRIS = 1;
1185        PUSH_BUTTON_2_TRIS = 1;
1186        RFIF = 0;
1187        RFIE = 1;
1188       LED_1 = 0;
1189       LED_2 = 0;
1190
1191       PORTAbits.RA0=0;
1192       PORTAbits.RA1=0;
1193       PORTAbits.RA3=0;
1194       PORTAbits.RA4=0;
1195
1196
1197    }
1198
1199    #elif defined(EXPLORER16) void BoardInit(void) {
1200        #ifdef __PIC32MX__
1201
1202           unsigned int pbFreq;
1203
1204           /* Clear SPI1CON register */
1205           SPI1CONCLR = 0xFFFFFFFF;
1206
1207           /* Enable SPI1, Set to Master Mode & Set CKE bit :
1208              Serial output data changes on transition
1209              from active clock state to Idle clock state */
1210           SPI1CON = 0x00008120;
1211
1212           /* Peripheral Bus Frequency = System Clock / PB Divider */
1213           pbFreq = (DWORD)CLOCK_FREQ / (1 << mOSCGetPBDIV() );
1214
1215           /* PB Frequency can be maximum 40 MHz */
1216           if( pbFreq > ( 2 * MAX_SPI_CLK_FREQ_FOR_MIWI) )
1217           {
1218               {
1219                   unsigned int SPI_Clk_Freq;
1220
1221                   unsigned char SPI_Brg = 1;
1222
1223                   /* Continue the loop till you find SPI Baud Rate Register Value */
1224                   while(1)
1225                   {
1226                      /* SPI Clock Calculation as per PIC32 Manual */
1227                      SPI_Clk_Freq = pbFreq / ( 2 * ( SPI_Brg + 1) );
1228
1229                      if( SPI_Clk_Freq <= MAX_SPI_CLK_FREQ_FOR_MIWI )
1230                      {
1231                          break;
1232                      }
1233
1234                      SPI_Brg++;
1235                   }
1236
1237                  mSpiChnSetBrg(1,SPI_Brg);
1238
1239               }
1240           }
1241           else
1242           {
1243               /* Set SPI1 Baud Rate */
1244               mSpiChnSetBrg(1,0);
1245           }
1246
1247           /* Set the Port Directions of SDO, SDI, Clock & Slave Select Signal */
1248           mPORTFSetPinsDigitalOut(PIC32MX_SPI1_SDO_SCK_MASK_VALUE);
1249           mPORTFSetPinsDigitalIn(PIC32MX_SPI1_SDI_MASK_VALUE);
1250
1251           /* Set the Interrupt Priority */
1252           mINT2SetIntPriority(4);
1253
1254           /* Set Interrupt Subpriority Bits for INT2 */
```

```
1255            mINT2SetIntSubPriority(2);
1256
1257            /* Set INT2 to falling edge */
1258            mINT2SetEdgeMode(0);
1259
1260            /* Set the Interrupt Priority for Push Button 2 & 3 as '5' */
1261            mCNSetIntPriority(5);
1262
1263            /* Set the Interrupt Priority for Push Button 2 & 3 as '1' */
1264            mCNSetIntSubPriority(1);
1265
1266            /* Enable Multi Vectored Interrupts */
1267            INTEnableSystemMultiVectoredInt();
1268
1269        #else
1270
1271            SPI1CON1 = 0b0000000100111110;
1272            SPI1STAT = 0x8000;
1273
1274            CNEN1bits.CN15IE = 1;
1275            CNEN2bits.CN16IE = 1;
1276
1277        #endif
1278
1279        PHY_RESETn = 0;
1280        PHY_RESETn_TRIS = 0;
1281        PHY_CS = 1;
1282        PHY_CS_TRIS = 0;
1283        PHY_WAKE = 1;
1284        PHY_WAKE_TRIS = 0;
1285
1286        LED_1_TRIS = 0;
1287        LED_2_TRIS = 0;
1288        PUSH_BUTTON_1_TRIS = 1;
1289        PUSH_BUTTON_2_TRIS = 1;
1290        RFIF = 0;
1291        RFIE = 1;
1292    }
1293
1294    #else
1295        #error "Unknown demo board.  Please properly initialize the part for the board."
1296    #endif
1297
1298
1299    /*******************************************************************
1300     * Function:        void Enable_PB_1_2_Interrupts()
1301     *
1302     * PreCondition:    None
1303     *
1304     * Input:           None
1305     *
1306     * Output:          None
1307     *
1308     * Side Effects:    None
1309     *
1310     * Overview:        Configure/Enable PB 1 & 2 interrupts.
1311     *
1312     * Note:            None
1313     *******************************************************************/
1314
1315    #ifdef __PIC32MX__
1316
1317    static void Enable_PB_1_2_Interrupts(void) {
1318        unsigned int value;
1319
1320        CNCON = 0x8000;  // Enable Change Notice module
1321
1322        /* Configure Change Notice Registers for Push Button 1 & 2. These buttons can be
1323           used to wake the controller from sleep state. */
1324        CNEN  = 0x00018000; // Enable CN15 and CN16 pins
1325
1326        /* Read port to clear mismatch on change notice pins */
1327        value = PORTD;
1328
1329        mCNClearIntFlag(); // Clear the CN interrupt flag status bit
1330
1331        mCNIntEnable(1); // Enable Change Notice interrupts
1332    }
1333
```

```
1334  #endif
1335
1336  /*********************************************************************
1337   * Function:        void _CN_Interrupt_ISR(void)
1338   *
1339   * PreCondition:    None
1340   *
1341   * Input:           None
1342   *
1343   * Output:          None
1344   *
1345   * Side Effects:    None.
1346   *
1347   * Overview:        This is the interrupt handler for push button 1 & 2 to
1348   *                  wake up the controller in Sleep State.
1349   ********************************************************************/
1350  #if defined(__dsPIC30F__) || defined(__dsPIC33F__) ||
1351  defined(__PIC24F__) || defined(__PIC24H__)
1352      void _ISRFAST __attribute__((interrupt, auto_psv)) _CNInterrupt(void)
1353      {
1354          IFS1bits.CNIF = 0;
1355      }
1356  #elif defined(__PIC32MX__)
1357
1358  void __ISR(_CHANGE_NOTICE_VECTOR, ipl5) _CN_Interrupt_ISR(void) {
1359      /* Disable Watchdog Timer to make sure that reset doesn't happen */
1360      DisableWDT();
1361
1362      {
1363          unsigned int value1;
1364
1365          value1 = PORTD; // Read PORTD to clear CN15/16 mismatch condition
1366
1367          if(value1 & 0x80 )
1368          {
1369             Pushbutton_1_Wakeup_Pressed = 1;
1370          }
1371          else
1372          {
1373             Pushbutton_2_Wakeup_Pressed = 1;
1374          }
1375      }
1376
1377
1378      mCNIntEnable(0); // Disable the CN Interrupt
1379
1380      CNCON = 0x0000; // Disable CN Module
1381
1382      CNEN = 0x0000;
1383
1384      mCNClearIntFlag(); // Clear the CN interrupt flag status bit
1385
1386  }
1387
1388  #endif
1389
```

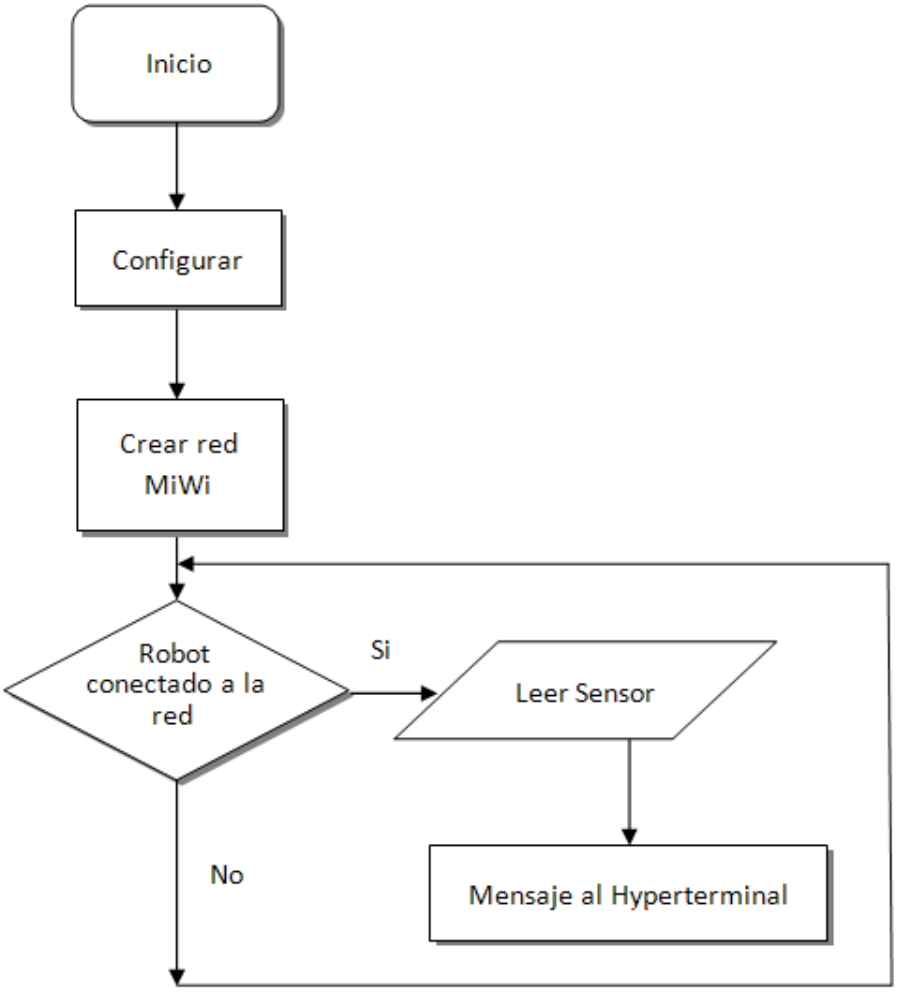


Figura E.1: Digrama de flujo de datos de programa miwi

F

CARACTERÍSTICAS TÉCNICAS DE LOS DISTINTOS ELEMENTOS DE LA PLACA

En este anexo se van a exponer las características más importantes de la placa, sobre los cuales no se ha dicho nada. Estos componentes son:

- Regulador de tensión TPS795xx
- Conector usb
- Regulador de tensión MAX603cpa+

F.1. Regulador de tensión TPS795xx

TEXAS INSTRUMENTS
www.ti.com

TPS795xx

SLVS350G–OCTOBER 2002–REVISED JULY 2006

ULTRALOW-NOISE, HIGH-PSRR, FAST, RF, 500-mA LOW-DROPOUT LINEAR REGULATORS

FEATURES

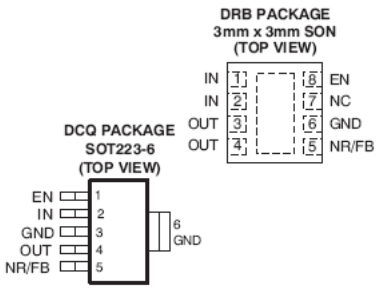
- **500-mA Low-Dropout Regulator With Enable**
- **Available in Fixed and Adjustable (1.2-V to 5.5-V) Versions**
- **High PSRR (50 dB at 10 kHz)**
- **Ultralow Noise (33 μV_{RMS}, TPS79530)**
- **Fast Start-Up Time (50 µs)**
- **Stable With a 1-µF Ceramic Capacitor**
- **Excellent Load/Line Transient Response**
- **Very Low Dropout Voltage (110 mV at Full Load, TPS79530)**
- **6-Pin SOT223 and 3 × 3 SON Packages**

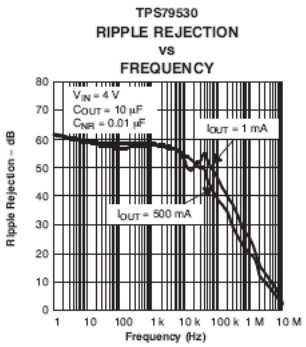
APPLICATIONS

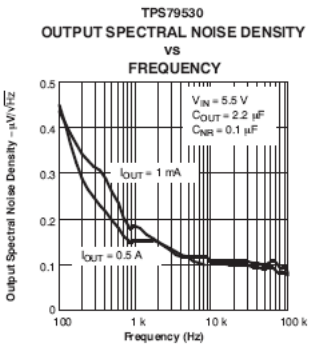
- **RF: VCOs, Receivers, ADCs**
- **Audio**
- **Bluetooth®, Wireless LAN**
- **Cellular and Cordless Telephones**
- **Handheld Organizers, PDAs**

DESCRIPTION

The TPS795xx family of low-dropout (LDO), low-power linear voltage regulators features high power-supply rejection ratio (PSRR), ultralow noise, fast start-up, and excellent line and load transient responses in small outline, SOT223-6 and 3 x 3 SON packages. Each device in the family is stable with a small 1-µF ceramic capacitor on the output. The family uses an advanced, proprietary BiCMOS fabrication process to yield extremely low dropout voltages (for example, 110 mV at 500 mA). Each device achieves fast start-up times (approximately 50 µs with a 0.001-µF bypass capacitor) while consuming very low quiescent current (265 µA, typical). Moreover, when the device is placed in standby mode, the supply current is reduced to less than 1 µA. The TPS79530 exhibits approximately 33 μV_{RMS} of output voltage noise at 3.0 V output with a 0.1-µF bypass capacitor. Applications with analog components that are noise-sensitive, such as portable RF electronics, benefit from the high-PSRR and low-noise features, as well as from the fast response time.







F.2. Conector USB

molex
one company › a world of innovation

This document was generated on 09/10/2009

PLEASE CHECK WWW.MOLEX.COM FOR LATEST PART INFORMATION

Part Number: 0475900001
Status: Active
Overview: Micro USB
Description: 0.65mm (.026") Pitch Micro-USB AB Receptacle, Right Angle, Top Mount, SMT, Lead Free

Series image - Reference only

Documents:
3D Model
Drawing (PDF)
Product Specification PS-47589-001 (PDF)
Related Catalog Page (PDF)

EU RoHS
ELV and RoHS Compliant
REACH SVHC Not Reviewed

China RoHS

Need more information on product environmental compliance?

Email productcompliance@molex.com
Please visit the Contact Us section for any non-product compliance questions.

Search Parts in this Series
47590 Series

Mates With
47516 Micro-USB Plug

General	
Product Family	I/O Connectors
Series	47590
Application	Wire-to-Board
Comments	Top Mount Type
Component Type	Receptacle
Overview	Micro USB
Product Name	Micro-USB
Type	AB
Physical	
Boot Color	N/A
Circuits (Loaded)	5
Circuits (maximum)	5
Color - Resin	Black
Gender	Male
Keying to Mating Part	Yes
Lock to Mating Part	Yes
Material - Metal	Copper Alloy
Material - Plating Mating	Gold
Material - Plating Termination	Gold
Material - Resin	High Temperature Thermoplastic
Number of Rows	1
Orientation	Right Angle
PCB Locator	Yes
PCB Retention	Yes
Packaging Type	Embossed Tape on Reel
Panel Mount	No
Pitch - Mating Interface (in)	0.026 In
Pitch - Mating Interface (mm)	0.65 mm
Pitch - Term. Interface (in)	0.026 In
Pitch - Term. Interface (mm)	0.65 mm
Plating min: Mating (µin)	43
Plating min: Mating (µm)	0.11
Plating min: Termination (µin)	2
Plating min: Termination (µm)	0.05
Polarized to Mating Part	Yes
Polarized to PCB	Yes
Ports	1
Surface Mount Compatible (SMC)	Yes
Temperature Range - Operating	-20°C to +85°C
Termination Interface: Style	Surface Mount
Waterproof / Dustproof	No
Electrical	
Current - Maximum	1.8A
Current - Maximum	1A
Grounding to Panel	Yes
Shield Type	Full Shield
Shielded	Yes
Voltage - Maximum	100V AC (RMS)
Material Info	
Reference - Drawing Numbers	
Product Specification	PS-47589-001
Sales Drawing	SD-47590-001

F.3. Regulador de tensión MAX603cpa+

19-0269; Rev 0; 9/94

MAXIM

5V/3.3V or Adjustable, Low-Dropout, Low I_Q, 500mA Linear Regulators

MAX603/MAX604

General Description

The MAX603/MAX604 low-dropout, low quiescent current, linear regulators supply 5V, 3.3V, or an adjustable output for currents up to 500mA. They are available in a 1.8W SO package. Typical dropouts are 320mV at 5V and 500mA, or 240mV at 3.3V and 200mA. Quiescent currents are 15µA typ and 35µA max. Shutdown turns off all circuitry and puts the regulator in a 2µA off mode. A unique protection scheme limits reverse currents when the input voltage falls below the output. Other features include foldback current limiting and thermal overload protection.

The output is preset at 3.3V for the MAX604 and 5V for the MAX603. In addition, both devices employ Dual Mode™ operation, allowing user-adjustable outputs from 1.25V to 11V using external resistors. The input voltage supply range is 2.7V to 11.5V.

The MAX603/MAX604 feature a 500mA P-channel MOSFET pass transistor. This transistor allows the devices to draw less than 35µA over temperature, independent of the output current. The supply current remains low because the P-channel MOSFET pass transistor draws no base currents (unlike the PNP transistors of conventional bipolar linear regulators). Also, when the input-to-output voltage differential becomes small, the internal P-channel MOSFET does not suffer from excessive base current losses that occur with saturated PNP transistors.

Applications

5V and 3.3V Regulators
1.25V to 11V Adjustable Regulators
Battery-Powered Devices
Pagers and Cellular Phones
Portable Instruments
Solar-Powered Instruments

Features

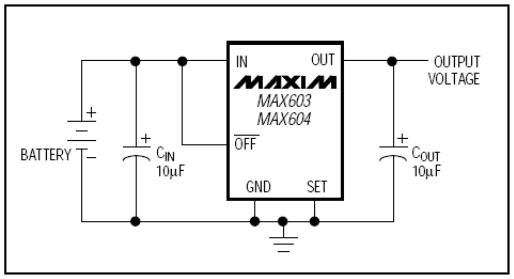
- **500mA Output Current, with Foldback Current Limiting**
- **High-Power (1.8W) 8-Pin SO Package**
- **Dual Mode™ Operation: Fixed or Adjustable Output from 1.25V to 11V**
- **Large Input Range (2.7V to 11.5V)**
- **Internal 500mA P-Channel Pass Transistor**
- **15µA Typical Quiescent Current**
- **2µA (Max) Shutdown Mode**
- **Thermal Overload Protection**
- **Reverse-Current Protection**

Ordering Information

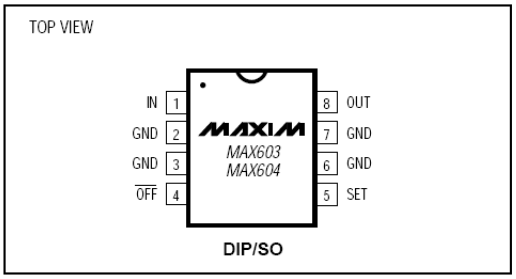
PART	TEMP. RANGE	PIN-PACKAGE
MAX603CPA	0°C to +70°C	8 Plastic DIP
MAX603CSA	0°C to +70°C	8 SO
MAX603C/D	0°C to +70°C	Dice*
MAX603EPA	-40°C to +85°C	8 Plastic DIP
MAX603ESA	-40°C to +85°C	8 SO
MAX603MJA	-55°C to +125°C	8 CERDIP**
MAX604CPA	0°C to +70°C	8 Plastic DIP
MAX604CSA	0°C to +70°C	8 SO
MAX604C/D	0°C to +70°C	Dice*
MAX604EPA	-40°C to +85°C	8 Plastic DIP
MAX604ESA	-40°C to +85°C	8 SO
MAX604MJA	-55°C to +125°C	8 CERDIP**

* *Dice are tested at T_A = +25°C, DC parameters only.*
** *Contact factory for availability.*

Typical Operating Circuit



Pin Configuration



™ Dual Mode is a trademark of Maxim Integrated Products.

G

CONVERSIÓN DE BINARIO A HEXADECIMAL

En este anexo se tratan de forma muy breve los sistemas binario y hexadecimal y su conversión ya que son útiles para trabajar con los PICs.

G.1. Definiciones

Un sistema de numeración es un conjunto de símbolos y reglas de generación que permiten construir todos los números válidos en el sistema.

Hay una gran cantidad de sistemas numéricos pero nos en este proyecto sólo nos interesan dos:Binario y Hexadecimal.

G.1.1. Binario

El sistema binario, en matemáticas e informática, es un sistema de numeración en el que los números se representan utilizando solamente las cifras cero y uno (0 y 1). Es el que se utiliza en los ordenadores, pues trabajan internamente con dos niveles de voltaje, por lo que su sistema de numeración natural es el sistema binario (encendido 1, apagado 0).

G.1.2. Hexadecimal

El sistema hexadecimal, a veces abreviado como hex, es el sistema de numeración posicional de base 16 —empleando por tanto 16 símbolos—. Su uso actual está muy vinculado a la informática y ciencias de la computación, pues los ordenadores suelen utilizar el byte u octeto como unidad básica de memoria. Sus símbolos básicos son: 1,2,3,4,5,6,7,8,9,0,A,B,C,D,E y F. Se debe notar que A=10[1], B=11, C=12, D=13,E=14 y F=15.

G.2. Tabla de conversión binario-hexadecimal

Por último se hará una tabla de conversión numérica entre binario y hexadecimal, necesaria para configuración de bits, definición de variables, etc. En los puertos del PIC.

Para la conversión de Binario a Hexadecimal se agrupa la cantidad binaria de 4 en 4 iniciándose por el lado derecho y completando con 0 por el izquierdo hasta conseguir un múltiplo de 4.

Decimal	Binario	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Tabla G.1: Tabla de conversión de Binario a Hexadecimal

[1] Equivalencias con el sistema decimal.