Facultad de Ingeniería
Universidad de la República Oriental del Uruguay

IIEMOTE

Análisis, diseño y elaboración de un sistema receptor y transmisor de bajo consumo, acorde a la norma IEEE 802.15.4 en las bandas 868/915 MHz

Ramiro C. Moreira Moreno [ramirocmoreira@gmail.com]
Federico Nuñez Artigas [fede@compendio.com]
Juan I. Rivero Rivero [mvjuani@gmail.com]

Tutores
Rafaella Fiorelli
Fernando Silveira

3 de agosto de 2007

Resumen

Este documento presenta las tecnologías involucradas, el análisis del problema y el diseño de un sistema receptor y transmisor acorde a la norma IEEE 802.15.4 en las bandas 868/915 MHz y de bajo consumo. Esencialmente el sistema diseñado y desarrollado esta compuesto por un microcontrolador, un transceiver y periféricos orientados a comunicarlo con el medio.

Se presenta los esquemáticos del sistema, el circuito impreso, la lista de partes, métodos de fabricación y bibliotecas de software realizadas para esta plataforma.

La plataforma desarrollada cumple con los requerimientos de un gran número de aplicaciones y líneas de investigación actuales: redes de sensores (sensor network), domótica, tecnología médica, tecnología agraria, entre otros.

El sistema incluye un microcontrolador MSP430 que cuenta con 60kb de ROM y 2kb de RAM disponible. La plataforma prevé puertos de expansión desde los cuales se puede acceder a 16 pines de entrada y salida, 8 pines de ADC y puertos RS232. El sistema es programable vía JTAG y se cuenta con un amplio kit de desarrollo (CompiladorGCC[1]) disponible bajo la licencia GPL[2]. El sistema puede comunicarse a una distancia de 40 metros con un $PER < 1\,\%$, consumo típico de 49,1mA en transmisión y de 56,7mA en recepción (@770kHz y RS-232 apagado), y su consumo típico en modo dormido es de $17\mu A$.

La información contenida en este documento posibilita la fabricación de esta plataforma y facilita el desarrollo de software para la misma.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Motivación

Actualmente hay muchas líneas de investigación dedicadas a sistemas embebidos interconectados en redes WPAN (Wireless Personal Area Network) de baja velocidad, orientadas a bajo consumo y bajo costo.

Contar con muchas computadoras pequeñas de bajo consumo, operando como un sistema integrado interconectado en forma inalámbrica, abre una gran cantidad de posibilidades y rompe paradigmas sobre la utilización de las mismas. Las mejoras tecnológicas hacen posible pensar en contar con dispositivos con estas características pero del tamaño de un grano de arena (*S*mart Dust[3]), lo cual dispara aun más las posibilidades de estos sistemas.

Estos sistemas deben tender a operar de una forma transparente al usuario, como expresa Mark Weiser en "The Computer for the 21st Century"[4]: "*Las tecnologías más profundas son aquellas que desaparecen. Ellas se disuelven a sí mismas en la fábrica de del día a día hasta ser indistinguibles*"[1]. Éste es el principal desafío que los sistemas embebidos deben afrontar.

Buscando sentar las bases de estas tecnologías, la IEEE forma un grupo de trabajo y desarrolla el estándar IEEE 802.15.4, el cual define la capa física y el acceso al medio de una red de alcance personal orientada a equipos embebidos de bajo consumo.

El creciente desarrollo de estos sistemas obliga a experimentar con estas tecnologías, para poder comprender de manera integral estos sistemas. Esto motiva a que se impulsen desarrollos en esta área, intentando siempre cumplir con los estándares en la materia.

[1] Versión original en ingles: "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*"

1.2. Antecedentes

El Grupo de Microelectrónica del IIE[2], en el marco del proyecto PDT: "Sensores Inalámbricos Integrados de Bajo Consumo", diseñó un transmisor de bajo consumo, para la banda ISM[3] de 915 MHz, integrado (en adelante TXIIE).

Esto generó la necesidad de contar con un receptor para dicho integrado. En busca de satisfacer esta necesidad, se propone desarrollar un sistema capaz de transmitir y recibir (transceiver) en las bandas 868/915 MHz compatible con la norma IEEE 802.15.4 y orientado a bajo consumo.

Hace ya tiempo que el Grupo de Microelectrónica del IIE viene impulsado investigaciones en esta área, habiendo ya realizado un proyecto de características similares al presente proyecto, pero en la banda de 433 MHz[5].

1.3. Requerimientos

Previo al inicio del proyecto se fijaron los siguientes requerimientos para el sistema:

- Debe transmitir y recibir acorde a la norma IEEE 802.15.4 en las bandas 868/915 MHz (ver reseña de la norma en el apéndice A).
- Debe poder operar con PER[4] $< 1\,\%$ hasta al menos 10m de distancia.
- La potencia de transmisión debe ser programable para distancias de 50m y menores.
- La alimentación debe ser de 2V a 2.8V (batería).
- El consumo debe ser tal que la duración de una batería de 1Ah debe ser mayor a un año con una transferencia de 10bytes/s.
- El sistema debe enviar y recibir correctamente aún frente a movimientos del dispositivo remoto, a una velocidad de por lo menos 0.4m/seg respecto de la base.
- El área del módulo debe ser menor a 16 cm^2.
- Es necesario que el diseño del sistema contemple la posibilidad de ser ampliado a una red de dispositivos remotos controlados por una base.

[2] IIE: Instituto de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de la República Oriental de Uruguay
[3] ver 6.2.1
[4] PER: packet error rate

- Es deseable que el receptor pueda comunicarse también con PSK "crudo" (o sea, sin utilizar Spread Spectrum como pide la norma), a un bit rate de hasta 40kb/s a los efectos de servir de receptor para transmisores experimentales.

1.4. Estructura de esta documentación

En esta sección se describe el contenido de la presente documentación.

- Capítulo 1, "Introducción": Resumen, motivación, antecedentes, contenido de la documentación, requerimientos del sistema y descripción de la norma IEEE 802.15.4.
- Capítulo 2, "Análisis": Comparación entre ComBlocks y modelo basado en transceiver, elección del transceiver, elección del microcontrolador, elección de los reguladores de voltaje y diseño conceptual del sistema.
- Capítulo 3, "Descripción del transceiver": Descripción general del transciver.
- Capítulo 4, "Descripción del microcontrolador": Descripción general del microcontrolador.
- Capítulo 5, "Antena y red de adaptación": Se realiza un estudio de las antenas elegidas, se presentan los cálculos para las redes de adaptación.
- Capítulo 6, "Diseño y elaboración del circuito": Contiene el diseño del esquemático, el diseño del PCB y las técnicas de soldado en el montaje del sistema.
- Capítulo 7, "Software de prueba": Describe el software de prueba y las librerías de acceso al hardware desarrolladas para la plataforma.
- Capítulo 8, "Pruebas y ajustes": Aquí se detallan los ajustes realizados durante el encendido del $\mathbf{I_{IE}M_{OTE}}$, las pruebas de alcance y las pruebas de consumo.
- Capítulo 9, "Diseño final": Se propone un diseño final para el $\mathbf{I_{IE}M_{OTE}}$, incorporando correcciones y mejoras a el prototipo.
- Capítulo 10, ' 'Evaluación del proyecto": Se realiza se hace una evaluación final, analizando el transcurso del proyecto, los costos y potenciales del mismo.

Capítulo 2

Análisis

2.1. Introducción

Este capítulo presenta un análisis entre dos posibles sistemas de recepción y transmisión de RF en las bandas de 868/915MHz. Para este análisis se tomaron en cuenta los requerimientos especificados en el capítulo anterior.

En una primera instancia se plantea un sistema utilizando bloques comerciales ComBlock. El segundo sistema estudiado, se basada en un transceiver e implica el desarrollo de un circuito a medida. Para este sistema, fue necesario realizar un diseño conceptual y analizar las distintas posibilidades que brinda el mercado para cumplir con los distintos subsistemas del mismo (transceivers, controladores, reguladores de voltaje, etc.) para así llegar a un diseño final.

Finalmente se comparan los 2 sistemas estudiados y se concluye cuál es el modelo a seguir.

2.2. ComBlocks

Los ComBlocks son pequeños módulos comerciales que están pre-programados con funciones de procesamiento orientado a las comunicaciones, incluyendo modulación, demodulación, conversión RF de digital a analógico, conversión RF de analógico a digital, corrección de errores en decodificación y codificación, interfaces banda base, etc.

En nuestro caso particular si se quiere realizar un diseño utilizando bloques ComBlock, lo primero a tener en cuenta es que no se podrá diseñar un sistema receptor y transmisor a la vez (transceiver), sino que el receptor y el transmisor serán bloques independientes. La razón, es que sencillamente no existen transceivers que trabajen dentro de nuestro rango de frecuencia de operación. Los únicos 2 transceivers ComBlock son el

COM-3501 (225-400MHz) y el COM-3502 (2.4GHz) y no cumplen los requerimientos especificados. Teniendo esto en consideración, se analizó entonces qué bloques se deberían utilizar en cada una de las 2 etapas (RX y TX). En la Figura 2.1 se esquematiza la solución que mejor se adapta a nuestro proyecto utilizando módulos ComBlock.

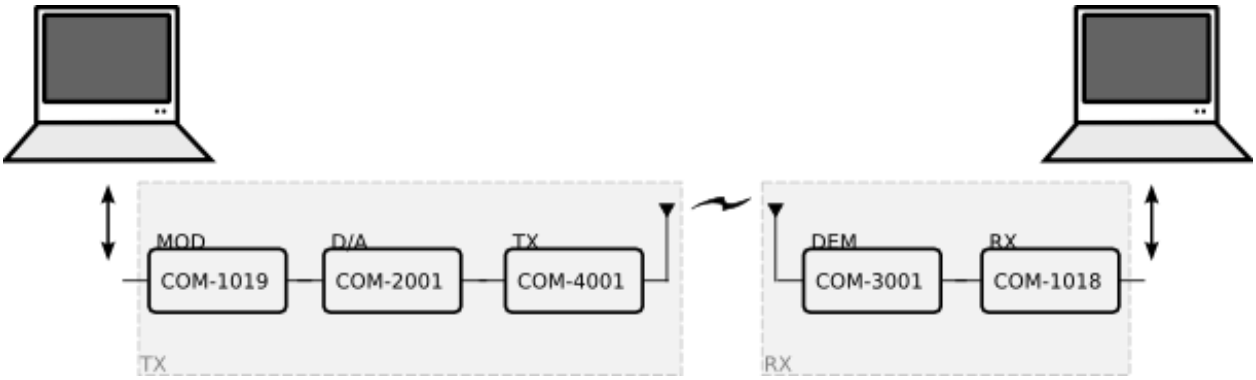

Figura 2.1: Sistema de recepción-transimisión utilizando ComBlocks

2.2.1. Bloques TX

A continuación se describen las principales características de los bloques utilizados en el sistema de transmisión utilizando ComBlocks (figura 2.1).

COM-1019[6] Recibe conexión serial (LVTTL 0-3.3V) y modula la señal en BPSK/QPSK/OQPSK a una frecuencia intermedia.
Características principales del bloque:

- Modulador DSSS (Direct Sequence Spread Spectrum).
- Alimentación: 5V.
- Consumo máximo: 600mA.
- Rate: 20 Mchip/s.
- Costo: 295 U$S.

COM-2001[7] Recibe la señal en fase y cuadratura y la convierte a analógica.
Características principales del bloque:

- Conversor D/A de 10 bits.

- Alimentación: 5V
- Rate: 125 Msamples/s
- Costo: 295 U$S

COM-4001[8] Recibe la señal analógica y la modula a 915MHz enviando la señal RF a la antena a través de un conector SMA.
Características principales del bloque:

- Modulador doble banda (915MHz/2.4GHz).
- Alimentación: 5V.
- Consumo máximo: 300mA.
- Potencia de salida: -2dBm @ 915MHz.
- Costo: 395 U$S.

2.2.2. Bloques RX

A continuación se describen los bloques utilizados en el sistema de recepción (figura 2.1) utilizando ComBlocks.

COM-3001[9] Recibe la señal a través de la antena y la digitaliza a través de un conversor A/D de 10 bits.
Características principales del bloque:

- Receptor doble banda (915MHz/2.4GHz).
- Alimentación: 5V.
- Consumo máximo: 180mA.
- Sensibilidad y potencia: -26dBm (para estar a fondo de escala a la salida del A/D), 70 dBm de rango dinámico AGC.
- Rate: 40 Msamples/s (10 bits output samples).
- Costo: 345 U$S.

COM-1018[10] Recibe la señal digital modulada y la lleva a banda base.

Características principales del bloque:

- Demodulador DSSS (Direct Sequence Spread Spectrum).
- Alimentación: 5V.
- Consumo máximo: 300mA.
- Rate: 20 MChip/s.
- Costo: 295 U$S.

2.2.3. Conclusión de los módulos ComBlocks

El sistema requiere un voltaje de alimentación y un consumo elevado para nuestras especificaciones y a su vez, no tiene una funcionalidad orientada a cumplir la norma IEEE 802.15.4. El sistema ocuparía un espacio físico grande al tener el receptor y el transmisor por separado, en el caso que se quiera utilizar el receptor y el transmisor en un mismo sistema, el armado implicaría invertir gran cantidad de tiempo en implementar esta integración (en el desarrollo y la implementación del circuito de control y conmutación). Por último, el sistema tiene un costo elevado, solamente con la compra de los ComBlock los costos sobrepasan los 1500 U$S sin tomar en cuenta impuestos y costos de envío.

2.3. Modelo Transceivers

2.3.1. Diseño conceptual

En la figura 2.2 se presenta el diagrama conceptual del sistema de recepción y transmisión basados en un transceiver.

El transceiver transmite y recibe en Radio Frecuencia, a través de la antena (conectada a él mediante una red de adaptación de impedancia) e interactúa con el controlador enviando y recibiendo datos, así como otros comandos de control, interrupciones, reset, etc. El controlador a su vez integra sensores y actuadores para interactuar con el ambiente, y brinda a través de un puerto RS232 una interfaz de comunicación con dispositivos de terceros (por ejemplo una computadora).

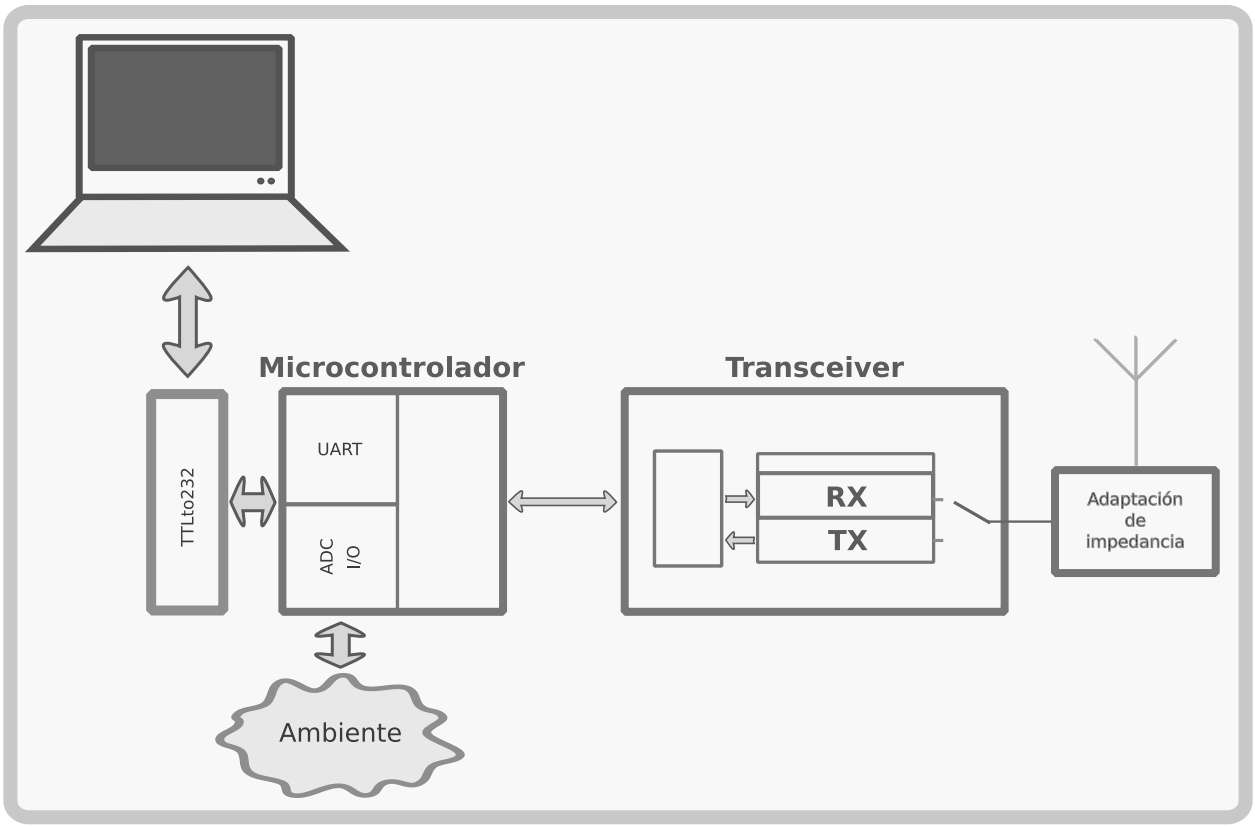

Figura 2.2: Diseño conceptual del sistema

2.3.2. Elección de transceiver

Búsqueda de fabricantes

Se indagó en una larga lista de fabricantes; en la tabla 2.1 se resume en una lista comparativa de lo encontrado.

Tabla 2.1: Lista de fabricantes de transceivers

Fabricante	Modulación	Frecuencia (MHz)	Norma
Atmel	BPSK	902-928	802.15.4
Chipcon (from TXI)	FSK	902-928MHz	NC
Ember	QFSK	2400	802.15.4
Linx Technologies	FSK	902-928	NC
Maxim MAX2424	BPSK	800-1000	NC
Melexis	FSK/ASK	868/915	NC
Micrel	FSK	700-1100	NC
Motorola (freescale)	QFSK	2400	802.15.4
Nordic Semiconductor	GFSK	915	NC
RF Micro Devices	GMSK	800-2000	NC
RFM	QFSK	2400	802.15.4
Semtech	FSK	902-928	NC
ZMD	BPSK	902-928	802.15.4

NC: No cumple con la norma 802.15.4

De la tabla 2.1 se desprende que los únicos fabricantes que tienen transceivers que cuentan con modulación BPSK, trabajan a una frecuencia de 868/915MHz y que en definitiva cumplen con la norma 802.15.4 son el de Atmel y el de ZMD.

Atmel El AT86RF210[11] es un transceiver compatible con el estándar ZigBee lo cual lo hace compatible con la norma IEEE 802.15.4, ya que Zigbee es una especificación de alto nivel, orientada a radios digitales de bajo consumo basadas en IEEE 802.15.4.

En la figura 2.3 se presenta la arquitectura interna del transceiver. La arquitectura es minimalista, y no provee ningún tipo de servicio para las capas altas, sólo implementa la capa física. No provee modos de operación autónomo, dependiendo siempre del microcontrolador para su funcionamiento.

La documentación aportada por el fabricante en su página web es pobre, solamente hay un hoja de datos de 10 carillas etiquetada como preliminar (actualizada por última vez en Octubre del 2003) y se limita a listar algunas características y ratings del transceiver.

Las características más relevantes del AT86RF210 se listan en el análisis comparativo de la página 18.

ZMD Tanto el ZMD44101[12] como el ZMD44102[13] son transceivers orientados a aplicaciones de bajo consumo dentro de las bandas 868/915MHz. Ambos comparten características similares, como por ejemplo integrar DSSS, sin embargo el ZMD44102 tiene la gran ventaja de estar diseñado para facilitar el cumplimiento de la norma IEEE 802.15.4, ya que este transceiver fue diseñado para ZigBee (al igual que el AT86RF210). Por esta razón orientaremos nuestro estudio en el ZMD44102.

La arquitectura interna se presenta en la figura 2.4, dentro de esto destaca el bloque de pre-procesamiento

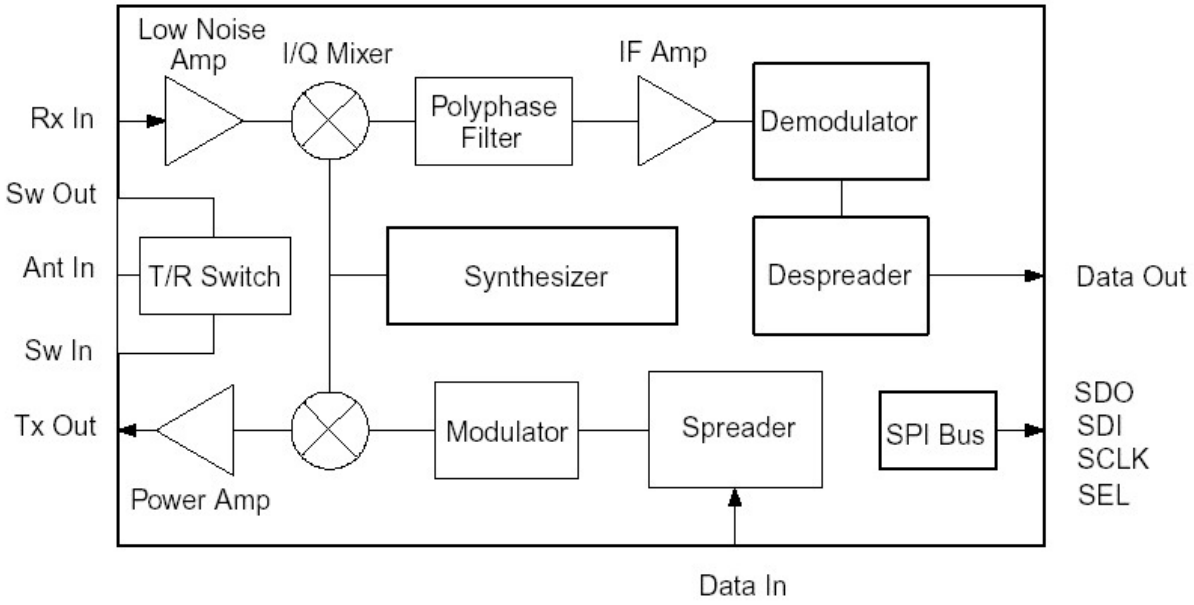

Figura 2.3: Diagrama de bloques del AT86RF21

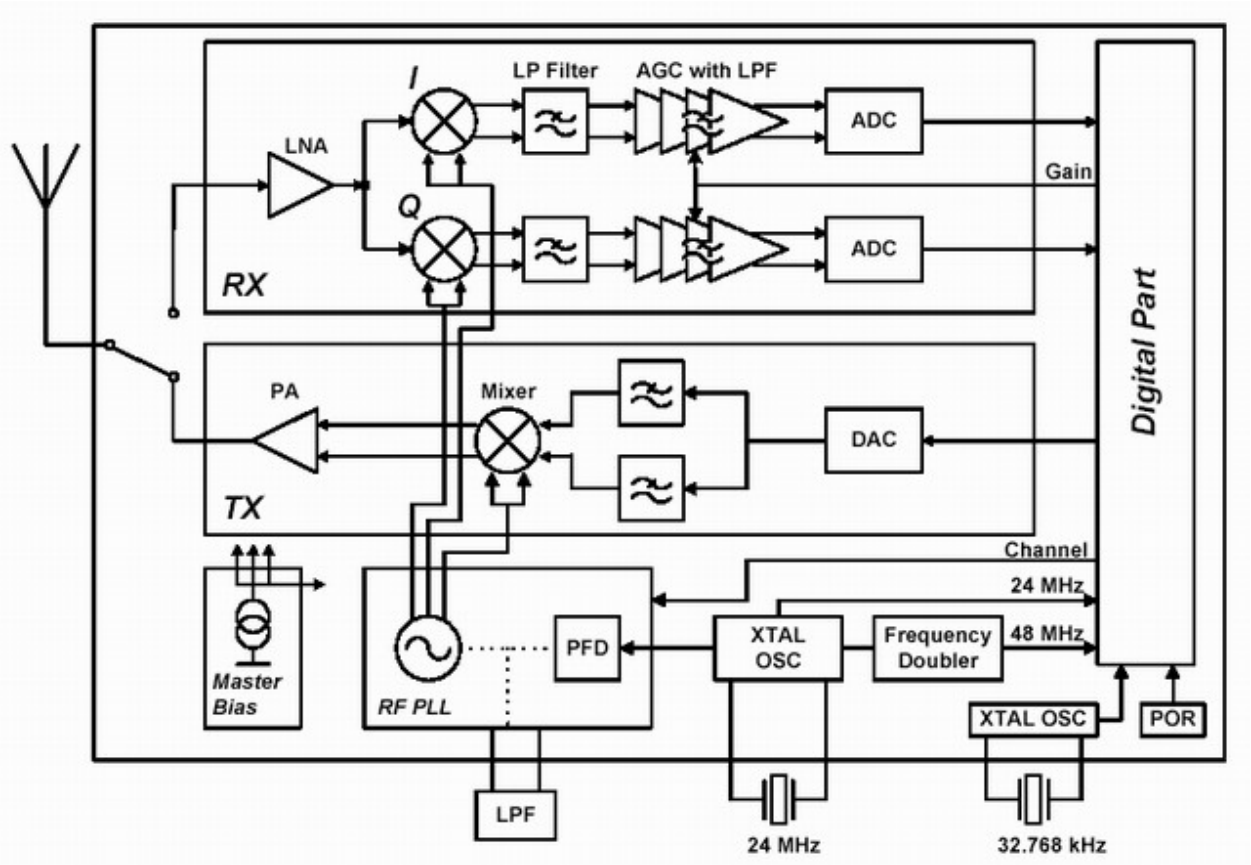

Figura 2.4: Diagrama de bloques de la capa PHY del ZMD44102

de capa MAC, el cual brinda servicios para facilitar el armado de tramas MAC. Algunos de los servicios provistos son: auto-generación beacons, respuesta automática de acknowledge, generación de CRC, filtro de paquetes. Son muchas las funciones que brinda este transceiver, y que facilitarán la compatibilidad con la norma liberando al microcontrolador de tareas rutinarias.

La documentación aportada por el fabricante es muy completa y actualizada. Además el fabricante facilita "Application Notes" en donde se describe un diseño de referencia completo, software y hardware.

Las características más relevantes del ZMD44102 se listan en la siguiente sección.

Análisis comparativo entre transceivers

Tabla 2.2: Comparación entre transceivers

Fabricante		**ZMD**	**Atmel**
Numero de parte		ZMD44102	AT86RF210
Frecuencia		906-924MHz	902-928MHz
Vcc		2.2-2.7V (2.4V típico)	1.8-3.6V (2.7V típico)
Vcc de I/O digitales		3.0-3.6V (3.3 típico)	1.8-3.6V (2.7V típico)
Consumo	**tx**	28mA @Vcc=2.4V	60mA @Vcc=3.3V
	rx	30mA @Vcc=2.4V	14.5mA @Vcc=3.3V
	dormido	2.3uA @Vcc=2.4V	1uA @Vcc=3.3V
Tasa		40Kbit/s	40Kbit/s
Pkg		48QFN	48QFN
Tiempo de retorno	Rx/Tx	180us	100us
	Tx/Rx	50us	100us
Sensibilidad		-95dBm @PER< 1 %	-95dBm @PER< 1 %
EVM		10 % (para 1000 chips)	35 % (para 1000 chips)
Costo		14.90 U$S	N/A

Como se puede apreciar en la Tabla 2.2, ambos transceivers cuentan con características muy similares (lo que era de esperar ya que ambos se basan en los mismos estándares).

El **AT86RF210** es de arquitectura minimalista y sólo resuelve la capa física, dejando a cargo del microcontrolador la capa MAC. La documentación provista por el fabricante es muy pobre, y no se encontró ningún distribuidor de este transceiver. Se intentó contactar con Atmel para buscar más información y nunca se obtuvo respuesta, lo cual no deja claro si el chip no es obsoleto.

El **ZMD44102** implementa muchos servicios para asistir en la implementación de la norma, resolviendo muchas cosas de capa MAC de forma autónoma. Todas estas ventajas se obtienen sin perdidas aparentes ante el AT86RF210, ni en consumo ni otras características.

Concluyendo, todas las ventajas técnicas, sumadas a la buena documentación, la accesibilidad a "Application Notes" y a la facilidad de compra, llevaron a optar por el transceiver **ZMD44102**.

2.3.3. Elección del microcontrolador

La elección del microcontrolador debe tener en cuenta los requerimientos del proyecto, debe ser de bajo consumo, y debe ser fácilmente integrable dentro del medio descrito.

Al momento de realizar la búsqueda, aún no estaba definida la aplicación final del producto, la cual nos daría ciertas pautas para limitar nuestra búsqueda, por este motivo, decidimos definir lo que queríamos poder hacer, para así ayudarnos a limitar la búsqueda sin tener que definir, en ese punto, una aplicación final concreta.

Lo que queríamos era centrar nuestro tiempo en el desarrollo del hardware y generar una plataforma donde fuera fácil realizar aplicaciones, por tanto se busco que el desarrollo del software fuera al más alto nivel posible, interesa que el desarrollo de software se pueda centrar en la aplicación y no así en las sutilezas del microcontrolador, por eso es necesario contar con al menos un compilador C para el microcontrolador seleccionado.

También se valoró la posibilidad de apoyar el desarrollo del firmware sobre algún sistema operativo embebido que brinde servicios básicos, como la administración de memoria, colas, procesos. Luego de estudiar las distintas líneas de investigación de Sensor Networks, parece claro que el sistema operativo más aplicable a los alcances del **IIEMOTE** es el TinyOS, si bien no es un requerimiento del proyecto, se centró la búsqueda en microprocesadores compatibles con TinyOS. Vale notar que si el controlador soporta TinyOS, entonces soporta los lenguajes de nestC y C.
Por los motivos aquí expuestos y luego de una intensa búsqueda, se determinó que los posibles candidatos se restringen a las siguientes familias:

- Atmel, familia AVR (ATMega128).
- Texas Instruments, familia MSP430.

De todos modos, para poder "medir el costo" de la decisión de incluir el requerimiento de soporte para TinyOS, también se analizó un microcontrolador PIC. Se buscó una familia comparable en número de periféricos y características, a las anteriores:

- Microchip, familia PIC18.

Análisis comparativo

Buscando contar con un sistema versátil, necesitamos que el microcontrolador tenga disponibles un buen número de entradas y salidas digitales, lo cual permite incluso poder pensar en módulos de expansión

hardware. A su vez se busca poder contar con entradas ADC que permitan conectar sensores para así poder "monitorear" el medio.

Por los motivos antes expuestos los puntos analizados en cada microcontrolador son:

- Bajo consumo
  - Distintos modos de ahorro de energía (active / partial sleep / sleep).
  - Tiempo de levantamiento (wakeup time).
  - Posibilidad de escalado del reloj (clock scaling). Esto se explica más abajo.
- Arquitectura de memoria.
- Riqueza del set de instrucciones.
- Disponibilidad de Entradas/Salidas.
- Disponibilidad de entradas ADC.
- Accesibilidad al kit de desarrollo.
- Costo.

Modos de ahorro de energía El MSP430 y el ATMega128L tienen varios modos de ahorro de energía, y pueden prender y apagar sus periféricos de forma independiente, sin embargo el PIC18 tiene sólo 2 modos de ahorro de energía (uno para los periféricos y otro para el core). El MSP430 tiene un tiempo de levantado de 6μs, inferior al del ATMega128L y el PIC18.

Escalado de la frecuencia del reloj Las tres arquitecturas implementan "clock scaling" manejado por software. El MSP430 usa una señal base de 32kHz la cual luego multiplica utilizando un "frequency-locked loop" digital (FLL), controlando por software el factor de multiplicación y logrando variar desde 32kHz hasta 6MHz (a 3V). El ATMega128L implementa un divisor digital de 7 bit y la frecuencia mínima del reloj es de 128KHz. El PIC18 puede usar un cristal externo o un oscilador RC interno de 8MHz y éste puede ser dividido a frecuencias entre 32kHz y 4MHz. El oscilador interno puede ser multiplicado por 4 usando un PLL llegando a generar frecuencias hasta 32MHz.

Arquitectura de la memoria El MSP430 es de arquitectura Von Neumann, con un solo espacio de direcciones, un puntero de 16 bits nos permite recorrer toda la RAM y ROM del mismo. El ATMega128L y el PIC18 usan una arquitectura Hardvard. Las características de memoria de La familia MSP430 son las siguientes:

- MSP430F133: 8KB + 256B de memoria Flash y 256B de memoria RAM.
- MSP430F135: 16KB + 256B de memoria Flash y 512B de memoria RAM.
- MSP430F147, MSP430F1471: 32KB + 256B de memoria Flash y 1KB de memoria RAM.
- MSP430F148, MSP430F1481: 48KB + 256B de memoria Flash y 2KB de memoria RAM.
- MSP430F149, MSP430F1491: 60KB + 256B de memoria Flash y 2KB de memoria RAM.

El ATMega128L cuenta con 128KB de memoria Flash, 4KB de memoria SRAM y 4KB de memoria EEPROM, mientras que el el PIC18F4525-1/PT cuenta con 48KB de memoria Flash, 3968B de memoria RAM y 1KB de memoria EEPROM.

Riqueza de set de instrucciones Tanto el MSP430 como el ATMega128L son muy ricos en el set de instrucciones disponibles. Tienen amplio rango de instrucciones aritméticas y muchos modos de direccionamiento. Sin embargo el PIC18 es más limitado en su set de instrucciones.

Entrada/Salida Los tres microcontroladores cuentan con diferentes versiones, con distintos periféricos incorporados (UART, ADC, contadores), el MSP430 cuenta con 40 pines dedicados a estos fines (8 de ADC), el ATMega128L cuenta con 53 (8 de ADC) y el PIC18 con 36 (de 10 a 13 de ADC).

Compilador El MSP430 y el ATMega128L cuentan con una versión del compilador C de GNU (GCC) portada para ellos. Mientras que el PIC18 tiene su propio compilador C comercial, por esto es relativamente más inflexible que los otros dos.

Costo Los costos son unitarios sacados de Digi-Key[14].

Microcontrolador	Costo
MSP430F149IPM	10.08 U$S
ATMega128L	15.05 U$S
PIC18F4525-I/PT	11.43 U$S

Conclusión

El MSP430 y el ATMega128L cumplen ampliamente todas nuestras demandas, la diferencia sustancial con el PIC18 radica en la pobreza de su set de instrucciones y que no cuenta con herramientas libres de desarrollo. La diferencias más importante entre el MSP430 y el ATMega128L está en base al consumo, que es uno de nuestros requerimientos más fuertes.

Como antes se expresó, todos los microcontroladores analizados tienen modos de ahorro de energía y están orientados al bajo consumo. En la siguiente tabla se expresa el consumo de los mismos en distintos modos de operación, tal como su hoja de datos lo reportan. En la misma se puede observar claramente que el MSP430 es el que tiene menor consumo en todos los casos, y el ATMega128L queda claramente relegado. También de esta comparación se desprende que el PIC18 no tiene ventajas frente al MSP430, ni en costo, ni en consumo.

Por estos motivos, el **MSP430F149** es el microcontrolador candidato para el "modelo basado en transceiver" y se concluye que la inclusión del requerimiento de compatibilidad con TinyOS no implica ningún "costo".

Tabla 2.3: Comparación de consumo entre los microcontroladores con $Vcc = 2V$

Microcontrolador		**AVR**	**MSP**	**PIC**
Palabra		8bit	16bit	8bit
Frecuencia máxima a 3V		8MHz	6MHz	20MHz
Consumo Dormido		8μA	1.8μA	2.6μA
Consumo Inactivo	**1MHz**	0.5mA	55μA	120μA
	8MHz	4mA	440μA	843μA
Consumo Activo	**32KHz**	88μA	19.2μA	35μA
	1MHz	2mA	240μA	480μA
	8MHz	8mA	1.9mA	2.4mA
Tiempo de levantamiento		2ms	6μs	10μs

2.3.4. Elección del regulador de voltaje

Para la elección de los reguladores de voltaje, se consideraron las siguientes características de diseño:

- Alcanzar los niveles de voltaje y corriente necesarios para alimentar el sistema.
- Alimentar la placa desde una batería de 2V a 2.8V.
- Emplear reguladores con bajos niveles de consumo sin carga ("low quiescent current").
- Emplear reguladores con modos de operación controlables y orientados a bajo consumo (sleep mode, power save mode, off mode, etc).
- Emplear reguladores que requieran pocos componentes externos, de forma de simplificar el diseño y mantenerlo lo más pequeño posible.

Estimativo de consumo máximo

Previo a la elección del regulador, es necesario estimar el consumo máximo del sistema. Este consumo está dado por la suma de los consumos de:

- Transceiver RF.
- Microcontrolador.
- Transceiver RS232 (adaptador de niveles de voltaje).
- Otros periféricos conectados a los puertos de expansión.

Transceiver | ZMD44102 El transceiver requiere que simultáneamente se alimente con dos niveles de tensión, los cuales fijamos en: 2.5V y 3.3V. La fuente de 3.3V se utiliza sólo para alimentar el subsistema de entradas/salidas digitales (I/O) mientras que la fuente de 2.5V alimenta el core digital y toda la parte analógica. El máximo consumo del transceiver se da en modo recepción y es de 30mA @2.5V. La documentación del ZMD44102 no da información del consumo a 3.3V, visto que sólo alimenta las entradas y salidas, estimamos su máximo en $10\mu A$.

Microcontrolador | MSP430F149 El microcontrolador se alimenta con una sola fuente de entre 1.8V a 3.6V, como referencia tomamos una alimentación de 3.3V. El consumo nominal, sin componentes externos y a la tensión de referencia, es de $I_{nom} = 472.5\mu A$, y el consumo máximo es $I_{max} = 612.5\mu A$.

También es importante evaluar el máximo consumo en modo dormido del microcontrolador, pues el regulador deberá poder cumplir con este requerimiento mientras el este dormido. El máximo consumo en modo dormido (LPM3) es de $I_{LPM3} = 1.9\mu A$.

RS232 El sistema requiere un adaptador de niveles, para convertir los niveles TTL del microcontrolador en niveles RS232. Como referencia tomamos el MAX3221E de Texas Instruments, este chip tiene un consumo máximo de $1mA@3.3V$.

Otros consumos El sistema cuenta con puertos de expansión, es necesario tomar en cuenta el consumo de los periféricos que se anexen, para esto reservamos 25mA.

Resumen Los requerimientos de consumo que deben de cumplir los reguladores a utilizar, se presentan en las tablas 2.4 y 2.5.

Tabla 2.4: Corriente de consumo @3.3V

Sub-sistema	**Consumo dormido**	**Consumo máximo**
Microcontrolador	$1.9\mu A$	$612.5\mu A$
Transceiver RF	$1\mu A$	$10\mu A$
Transceiver RS232	$10\mu A$	$1mA$
Otros	$2\mu A$	$25mA$
Totales	$5.9\mu A$	$26.6mA$

Tabla 2.5: Corriente de consumo @2.5V

Sub-sistema	**Consumo dormido**	**Consumo máximo**
Transceiver RF	$2.2\mu A$	$30mA$

Análisis comparativos y elección de reguladores

Teniendo en consideración que se va a alimentar la placa con una batería de 2 a 2.8V, es necesario entonces contar con reguladores STEP-UP/STEP-DOWN para alimentar 3.3V y 2.5V. Luego de definido el consumo, alimentación de entrada, y tensión de salida, se buscaron reguladores candidatos en muchos fabricantes. En una primera instancia se pensó utilizar un regulador dual, como el TPS70758[15] de Texas Instruments, pero no se encontró ninguno que pudiera utilizar alimentaciones inferiores al voltaje de salida (que incluya STEP-UP), por lo cual fue necesario utilizar 2 reguladores independientes, uno para 2.5V y otro para 3.3V.

Alimentación de 3.3V En la tabla 2.6 se presentan algunos de los reguladores que identificamos en el mercado y evaluamos como candidatos.

La familia de reguladores TPS6021x se alimenta de 1.8 a 3.6V, tiene 50mA o 100mA de salida máxima y es la única que cuenta con modo dormido ("snooze mode"), lo que permite bajar la corriente de reposo (o corriente quiescent) mientras el sistema está dormido. Los otros reguladores analizados tenían modos de apagado pero no "snooze mode", o sea, no continúan entregando corriente sino que se desconectan de la alimentación, lo cual no tendría utilidad en este diseño. Igualmente, si analizamos las corrientes de quiescent y cantidad de componentes externos necesarios, no encontramos ningún regulador que mejore las características de esta familia.

Dentro de esta familia elegimos los reguladores que tienen 100mA de salida mínima (TPS60210 y TPS60211), si bien nuestro estimativo de consumo es inferior a 50mA, al tener todos la misma corriente de quiescent, contar con 100mA le da más versatilidad al sistema. Finalmente, el regulador **TPS60210** tiene una salida digital que indica el estado de la batería lo cual puede ser utilizado por el $\mathbf{I_{IE}M_{OTE}}$ para reportar la necesidad del cambió de la misma, por esto es el elegido.

Tabla 2.6: Comparativa reguladores con voltaje de salida de 3.3V

Regulador	$\mathbf{I_{out}^{max}}$ **(A)**	$\mathbf{I_{out}^{snooze}}$ **(mA)**	$\mathbf{I_q}$ **(mA)**	$\mathbf{I_q^{sleep}}$ **(μA)**	$\mathbf{I_q^{sd}}$ **μA)**	$\mathbf{V_{in}^{min}}$ **(V)**	$\mathbf{V_{in}^{max}}$ **(V)**
TPS60210	0.1	<2	0.035	2	-	1.8	3.6
TPS60211	0.1	<2	0.035	2	-	1.8	3.6
TPS60212	0.05	<2	0.035	2	-	1.8	3.6
TPS60213	0.05	<2	0.035	2	-	1.8	3.6
TPS60200	0.1	-	0.04	-	0.05	1.8	3.6
TPS60201	0.1	-	0.04	-	0.05	1.8	3.6
TPS60202	0.05	-	0.04	-	0.05	1.8	3.6
TPS60203	0.05	-	0.04	-	0.05	1.8	3.6
TPS60204	0.1	-	0.035	-	0.05	1.6	3.6
TPS60205	0.1	-	0.035	-	0.05	1.6	3.6
REG710-33	0.03	-	0.065	-	0.01	1.8	5.5
REG711-33	0.05	-	0.06	-	0.01	1.8	5.5
LP2981-33	0.6	-	0.1	-	0.01	2.2	16
LP2981A-33	0.6	-	0.1	-	0.01	2.2	16
LTC3240-3.3	0.15	-	0.065	-	0.1	1.8	5.5
LTC3525-3.3	0.4	-	0.007	-	0.1	0.3	6

$\mathbf{I_q}$ corriente quiescent
$\mathbf{I_q^{sd}}$ corriente quiescent en modo apagado (shutdown)

Alimentación 2.5V En la tabla 2.7 se presentan algunos de los reguladores que identificamos en el mercado y evaluamos como candidatos.

En esta tensión no se identifico ningún regulador con modo dormido que cumpla con nuestros requerimientos, sólo se encontró con modo apagado.

Los REG7xx son una mejor opción, tienen menor corriente quiescent y necesitan menos cantidad de componentes externos, por lo cual el LTC3240-2.5 queda descartado. El **REG711-25** tiene una menor corriente de quiescent por lo cual es el seleccionado.

Tabla 2.7: Reguladores con voltaje de salida de 2.5V

Regulador	$\mathbf{I}_{out}^{max}$ **(A)**	$\mathbf{I}_{out}^{sleep}$ **(A)**	$\mathbf{I}_q$ **(mA)**	$\mathbf{I}_q^{sleep}$ **(μA)**	$\mathbf{I}_q^{sd}$ **μA)**	$\mathbf{V}_{in}^{min}$ **(V)**	$\mathbf{V}_{in}^{max}$ **(V)**
REG710-25	0.03		0.065	-	0.01	1.8	5.5
REG711-25	0.05		0.06	-	0.01	1.8	5.5
LTC3240-2.5	0.15		0.065	-	0.1	1.8	5.5

$\mathbf{I}_q$ corriente quiscient
$\mathbf{I}_q^{sd}$ corriente quiscient en modo apagado (shutdown)

2.4. Conclusión | Comparación de modelos

Se analizaron dos modelos, uno utilizando bloques discretos ComBlocks y otro desarrollando un circuito dedicado en basado en un transceiver.

El **modelo ComBlocks** permite tener una solución rápida, pero muy poco versátil. Con este enfoque, no es posible desarrollar de forma integrada un transmisor y un receptor, a no ser que se desarrolle un circuito a medida externo, que multiplexe ambas funciones. El consumo del sistema es alto, y no es útil en aplicaciones de bajo consumo. Por último, el costo del sistema es elevado.

El **modelo transceiver**, implica un esfuerzo de desarrollo mayor, pero permite llegar a una plataforma versátil y completamente compatible con la norma. El sistema descripto cumple con los requerimientos de bajo consumo y aplica a un gran número de funciones. El costo de fabricación de una unidad es más bajo que el del ComBlocks. También resulta alentador la posibilidad de desarrollar un sistema que pueda servir de plataforma de desarrollo para aplicaciones de terceros.

Concluyendo, del análisis de ambos modelos, tomando en cuenta lo descrito anteriormente, sobre consumo, tamaño, armado, precio y buscando el mayor cumplimiento de la norma IEEE 802.15.4.b, se desprende que el mejor modelo a utilizar es el del **circuito dedicado en basado en un transceiver (modelo transceiver)**. En la figura 2.5 se presenta el diagrama de bloques del sistema a implementar.

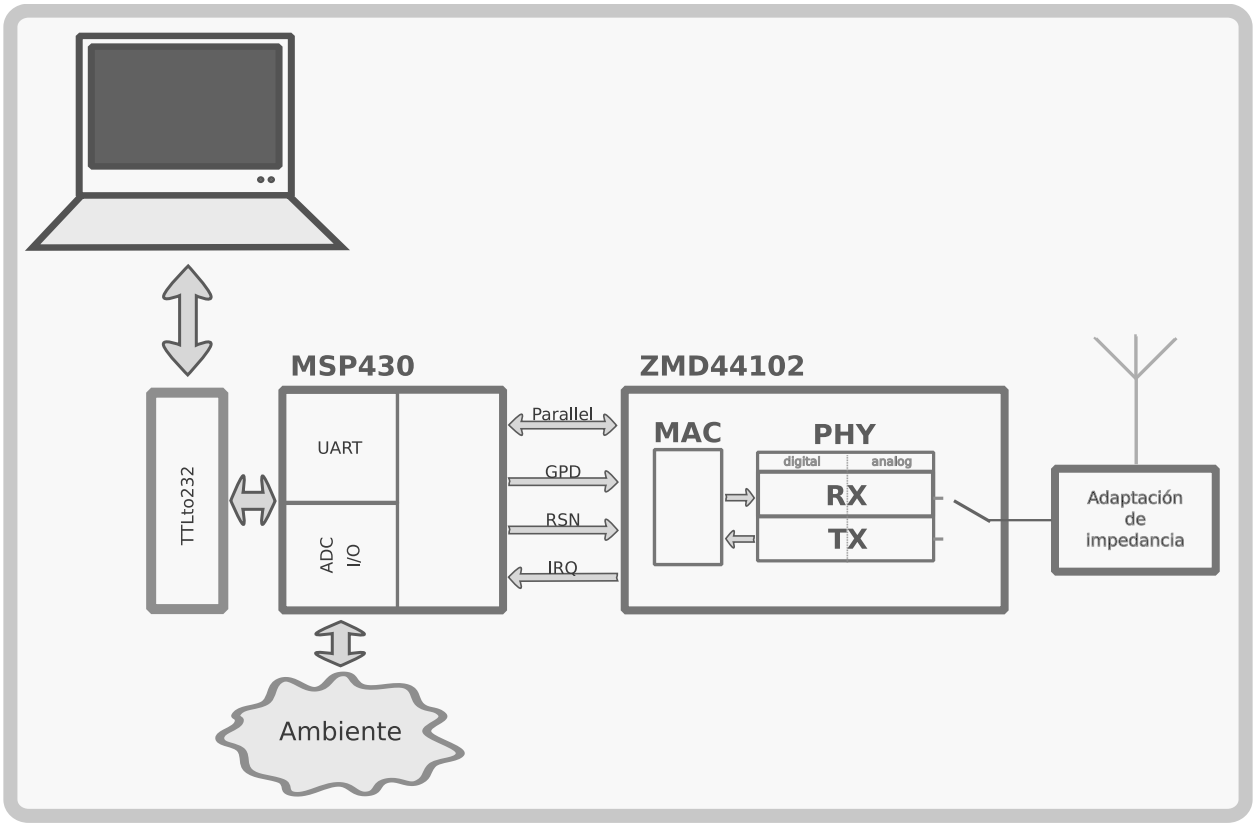

Figura 2.5: Diagrama de bloques | Diseño final

Capítulo 3

Descripción del transceiver

3.1. Descripción general

El integrado ZMD44102 es un transceiver CMOS que opera en las bandas ISM de 868.3 MHz (EU) y 902 MHz a 928 MHz (US).

Las bandas de radio frecuencia ISM (Industrial, Scientific and Medical) son bandas reservadas internacionalmente, originalmente creadas para uso no comercial en aplicaciones industriales, científicas y médicas.

En la actualidad estas bandas han sido popularizadas por su uso en comunicaciones WLAN (por ejemplo Wi-Fi) o WPAN (por ejemplo Bluetooth). Las bandas ISM fueron definidas por la ITU (Unión Internacional de Telecomunicaciones) en el artículo 5 de las Regulaciones Radio (RR), concretamente puntos 5.138 y 5.150.

El uso de estas bandas de frecuencia está abierto a todo el mundo sin necesidad de licencia. Distintos países adoptan determinadas bandas, por ejemplo en Europa se utiliza la banda 868.3 MHz mientras que en Estados Unidos y Australia se utiliza la banda de 915 MHz. Mediante las regulaciones internas de cada país, se limitan también los niveles de potencia transmitida. Este hecho fuerza a que estos tipos de comunicaciones tengan cierta tolerancia frente a errores y que utilicen mecanismos de protección contra interferencias, como técnicas de ensanchado de espectro.

El ZMD44102 es un transceiver RF de bajo consumo que utiliza voltajes de alimentación de +2.4V y +3.3V (para partes analógica y digital respectivamente). Basado en la norma IEEE 802.15.4, este transceiver alcanza tasas de transferencia de datos de hasta 40 kbits, e incorpora tecnología DSSS (Direct Sequence Spread Spectrum) para asegurar una transferencia confiable en ambientes hostiles de radio frecuencia. La norma IEEE 802.15.4 ha ganado popularidad en los últimos años debido al incremento en la utilización del protocolo de comunicaciones inalámbrico ZigBee[1].

[1] ZigBee es un protocolo de comunicaciones inalámbrico basado en el estándar para redes inalámbricas de área personal

En el Apéndice F de ésta documentación se describen las características eléctricas de operación y las características generales más destacadas del ZMD44102, así como su distribución de pines (pinout). Por información más detallada referirse a la DatasheetZmd[16].

3.2. Arquitectura interna

Como se observa en la figura 3.1, el ZMD44102 integra la capa física PHY con un soporte de hardware HW-MAC para la subcapa MAC y una interfaz para microcontrolador externo. La capa física PHY incluye la etapa analógica de RF y el procesamiento digital de señales.

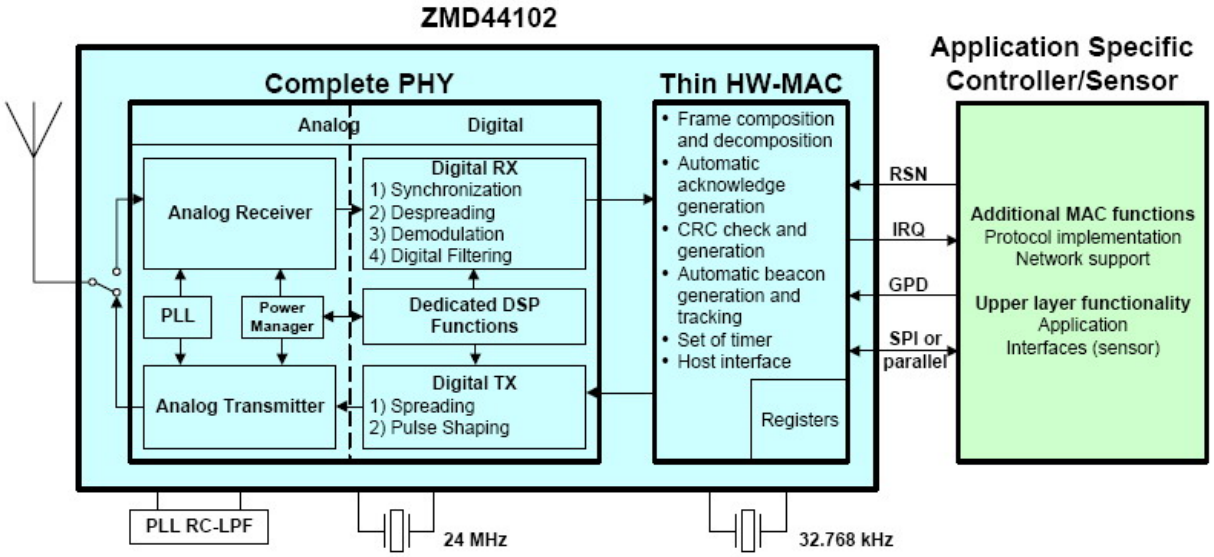

Figura 3.1: Diagrama de bloques del ZMD44102

Las siguientes secciones explican brevemente las principales características de cada bloque.

3.2.1. Integración HW-MAC

El ZMD44102 puede ser controlado a través de una interfaz serie o de una interfaz paralelo. A su vez presenta una salida para interrupciones (IRQ), una entrada de apagado general (GPD) y una entrada de reset del chip (RSN, activa en nivel bajo). El hardware MAC (HW-MAC) contiene una FIFO de transmisión de 128 bytes (TxFIFO) y una FIFO de recepción de 256 bytes (RxFIFO), maneja la composición y decomposición de las tramas así como la generación automática de reconocimientos (ACKs), la generación del CRC, y un control de temporizadores.

(WPANs) IEEE 802.15.4. Surge del fruto de una alianza, sin ánimo de lucro, de más de 100 empresas, la mayoría de ellas fabricantes de semiconductores, con el objetivo de conseguir el desarrollo e implantación de una tecnología inalámbrica de bajo costo

El ZMD44102 tiene un filtro de tramas de recepción integrado con tres niveles de filtrado:

- Nivel 1: Filtro CRC.
- Nivel 2: Filtro de tipo de trama y de dirección.
- Nivel 3: Filtro dependiente del modo de operación.

En el Nivel 1, las tramas con el checksum incorrecto son descartadas.

En el Nivel 2, la dirección y el identificador PAN de la trama son verificados de acuerdo con la sección 7.5.6.2 (Rejection and Reception) del estándar IEEE[17]. Por defecto, las tramas que tengan el tipo de trama reservado o sin definir, son rechazadas por el Nivel 2 del filtro.

En el Nivel 3, se realiza el siguiente filtrado:

- Rechazar todos las tramas que no sean del tipo acknowledge mientras se espera un acknowledge.
- Rechazar todas las tramas del tipo nonbeacon mientras se está en escaneo de fase del modo beacon track.
- Rechazar todas las tramas del tipo nonbeacon durante un active o passive scan.
- Rechazar todas las tramas del tipo non-command durante orphan scan.

Por defecto, los tres niveles de filtrado están activados, pero cada nivel puede ser desactivado por separado para propósitos de debugging.

El ZMD44102 utiliza dos relojes diferentes: uno de 24 MHz para el núcleo digital y otro reloj de tiempo real (RTC) de 32.768 kHz. El transciever tiene diferentes modos de apagado y dormido. Durante los modos Sleep y Global Power Down , el reloj de 24 MHz se apaga para reducir el consumo de energía. El RTC se mantiene encendido durante estos modos, permitiendo de esta forma no perder la sincronización de tiempo[2].

3.2.2. Capa PHY

Etapa analógica (RF)

Recepción El receptor del ZMD44102 utiliza una arquitectura de conversión directa Zero-IF (cero frecuencia intermedia), lo que resulta en la conversión digital a través de una demodulación directa desde la frecuencia de transmisión a bandabase. El camino de recepción (fig 3.2) consiste en un amplificador LNA (low noise amplifier) de 900MHz y un mixer, seguido por un filtrado pasabajos. A continuación la señal

[2] ver 3.10 por más información sobre los diferentes modos de operación del ZMD44102

pasa por múltiples amplificadores de ganancia programable (AGCs con filtros pasabajos) hasta llegar a los convertidores analógico-digital (ADC), los cuales convierten la señal a digital.

Todas las funciones restantes se llevan a cabo en el dominio digital.

En modo de operación normal, la recepción puede ser iniciada utilizando los valores por defecto de los registros. Todas las señales de control (temporizadores, apagado) son seteadas de forma automática.

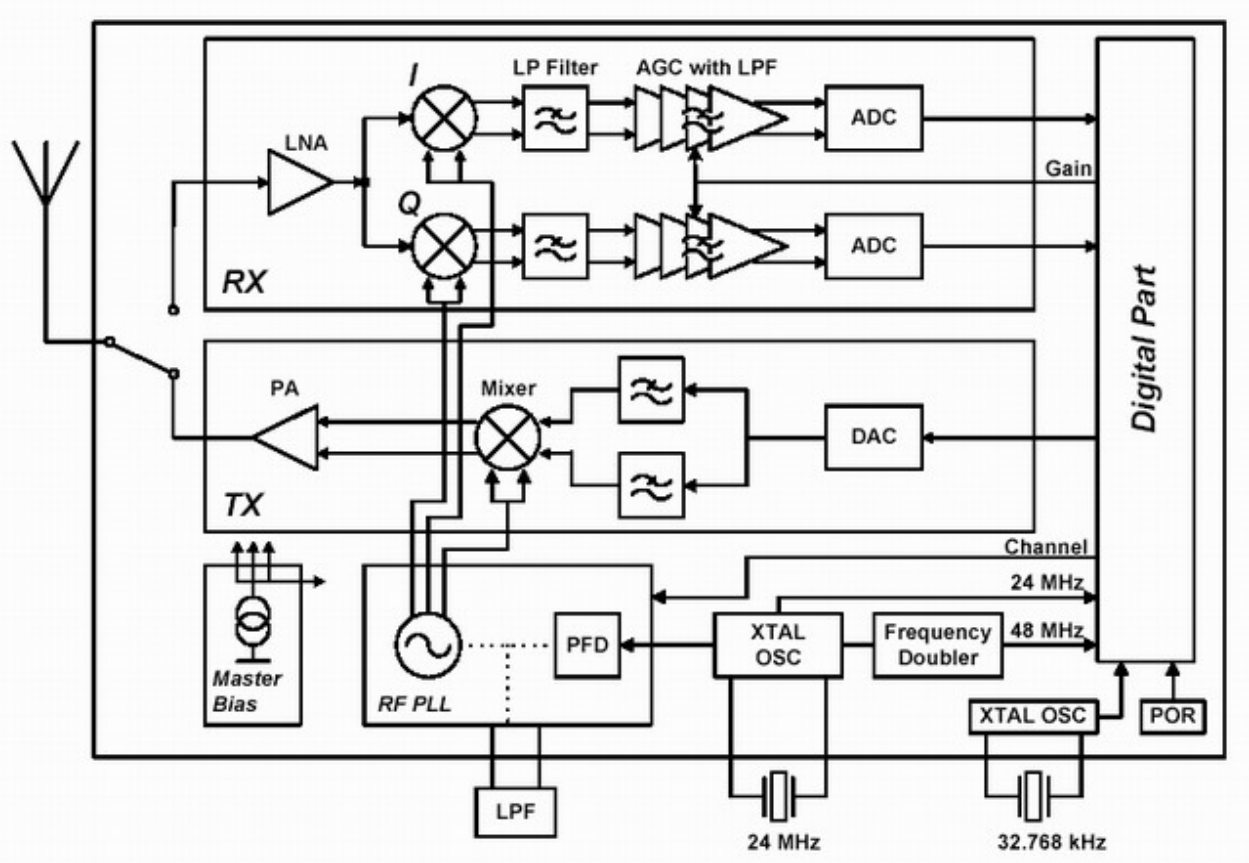

Figura 3.2: Diagrama de bloques de la capa PHY del ZMD44102

Transmisión El transmisor del ZMD44102 también utiliza arquitectura de conversión directa. El diseño es enteramente diferencial. Solamente la salida del amplificador de potencia (PA) no es diferencial. La salida del amplificador va directamente conectada a la antena, sin necesidad de red de adaptación externa.

De igual forma que para la recepción, la transmisión puede ser iniciada utilizando los valores por defecto de los registros. Todas las señales de control (temporizadores, apagado) son seteadas de forma automática. Opcionalmente, los registros por defecto del transmisor pueden ser cambiados a través del registro Transmit Mode Register (0x05). Por defecto, el PA interno transmite 0 dBm (1mW) con una impedancia de salida de 50 Ohm, sin embargo también se puede setear el registro Transmit Mode Register (0x05) y variar la potencia

de salida entre 0 dBm y -26 dBm.

Para que la entrada del receptor y la salida del transmisor sea a través del mismo pin (RFIO), el ZMD44102 cuenta con un switch integrado que desconecta los componentes respectivos dependiendo del modo seleccionado (Rx o Tx). La selección del modo puede ser seteada por el usuario desde el registro Mac Control Commands (0xA0).

Etapa digital

El bloque digital del transceiver realiza funciones de sincronización, spreading y pulse shaping (en modo transmisión), funciones de demodulación y de-spreading (en modo recepción), y filtrado digital entre otros procesamientos de señal (por más información sobre spreading o pulse shaping referirse a la sección A).

Además se realiza el control del loop AGC, con el cual se puede extender el rango dinámico de la señal controlando la ganancia del LNA y del mixer.

3.3. Osciladores de cristal

El ZMD44102 utiliza dos osciladores de cristal, uno que opera a 24 MHz y el otro a 32.768 kHz. El cristal de 32.768 kHz es utilizado como reloj en tiempo real en el modo Sleep y es utilizado para ajustes de tiempo si está operando en beacon-enabled. El cristal de 24 MHz es utilizado por la mayoría de los bloques digitales y provee la frecuencia de referencia para el PLL[3].

3.3.1. Oscilador de cristal de referencia (24 MHz)

La frecuencia de referencia de 24MHz es generada por un oscilador de Pierce simple con un resistor integrado al transceiver, un cristal externo de 24MHz y un par de condensadores (ver Figura 3.3). Esta frecuencia interviene en:

- La alimentación del reloj digital.
- Cálculos de tiempos.
- El PLL que genera la frecuencia portadora de RF.

[3] **PLL** - El lazo de seguimiento de fase, bucle de enganche de fase o PLL (phase-locked loop) es un circuito de lazo cerrado cuya señal de salida tiene una frecuencia variable según la frecuencia y la fase de la señal de referencia de entrada. La principal ventaja de los circuitos PLL es que permiten sintetizar frecuencias mayores a la frecuencia de referencia. Un circuito PLL responde a tanto la frecuencia como la fase de la señal de entrada, aumentando o disminuyendo la frecuencia de un oscilador controlado (que puede ser un VCO) hasta que se corresponde tanto en frecuencia como en fase con la señal de referencia. El oscilador controlado por tensión o VCO (voltage-controlled oscillator) es un circuito que usa amplificación, realimentación y circuitos resonantes, el cual proporciona a su salida una señal de frecuencia proporcional a la tensión de entrada.

En los modos de recepción, el integrado dobla la frecuencia de referencia para alcanzar la velocidad de procesamiento digital durante la adquisición de datos. Este oscilador está activo sólo en los modos IDLE, TX y RX. Cuando el oscilador interno está en uso se requieren C4 y C5 para el funcionamiento del cristal de resonancia. Los valores de C4 y C5 dependen del tipo de cristal utilizado.

La capacitancia total de carga se compone de los valores C4 y C5, además de los valores parásitos del PCB y de la capacidad parásita interna del ZMD44012 que es de 0.65 pF en cada pin. Para el cristal recomendado por ZMD (SMI SX5159) los valores típicos de C4 y C5 son 43 pF.

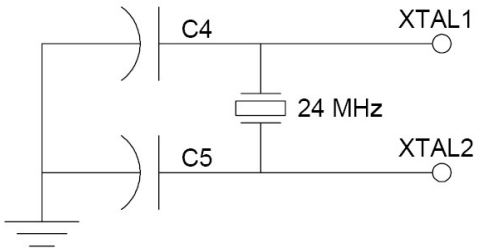

Figura 3.3: Cristal 24MHz

3.3.2. Oscilador de cristal para operaciones de bajo consumo (32.768 kHz)

El oscilador de cristal de 32.768 kHz es utilizado para operaciones de extremo bajo consumo, y siempre que el ZMD se encuentre alimentado está en funcionamiento. Existe, de todas formas, una opción en la que el usuario puede programar el cristal en modo Off y apagarlo. El oscilador provee una referencia de tiempo para el reloj de tiempo real del integrado. El cristal utiliza un oscilador de Pierce simple con un resistor integrado al transceiver.

ZMD recomienda el cristal SMI 155M327 debido a su tolerancia en frecuencia y a las especificaciones de rango de temperatura. Los valores de carga de los capacitores de carga dependen del tipo particular de cristal utilizado. Para el SMI 155M327 los valores típicos de los condensadores son de 15 pF (ver Figura 3.4).

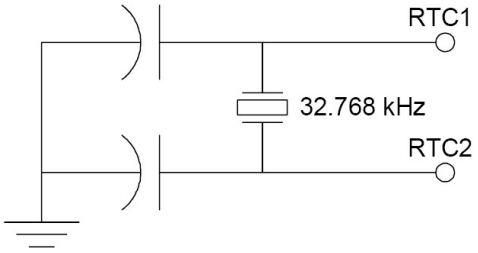

Figura 3.4: Cristal 32.768 Khz

3.4. PLL

El ZMD44102 utiliza un PLL fraccional-N. Todas las funciones están integradas en el chip, excepto el filtro de loop. El cristal de 24MHz provee la frecuencia de referencia para las bandas US y EU. En la figura 3.5 se muestra el esquemático del filtro de loop del PLL.

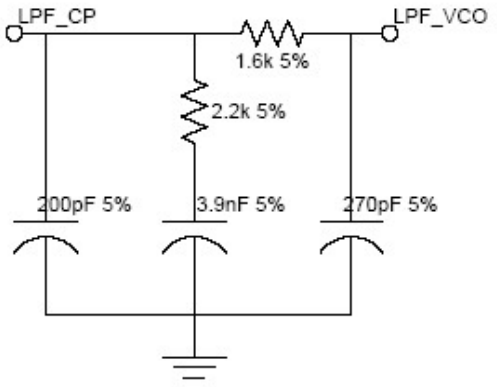

Figura 3.5: Filtro PLL

En el modo de operación normal, el usuario configura el canal de frecuencia de RF del PLL, antes de la transmisión o la recepción, escribiendo al registro PHY Current Channel (0x00). La tasa de transferencia de datos se ajusta automáticamente de acuerdo al canal seleccionado (40 kbit/s). Los números de canales y la separación en frecuencia de los mismos están definidos por el estándar IEEE 802.15.4, tal como se muestran en la figura 3.6.

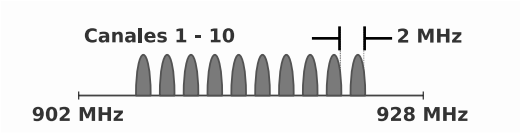

Figura 3.6: Canales

3.5. Conexión de la antena

Por defecto, el ZMD44102 utiliza para la entrada del receptor y la salida del transmisor el mismo pin (RFIO). El switch integrado desconecta los componentes respectivos dependiendo del modo seleccionado (Rx o Tx). La antena debe ser conectada vía un capacitor externo para bloquear continua. Para nuestro diseño en particular, además del bloqueo de continua se utilizan dos capacitores en paralelo y un inductor en serie que actúan como filtro pasabajos antes de la antena (ver 6.2.1).

Utilizando el pin RFO puede utilizarse un amplificador de potencia para amplificar la señal RF y aumentar el alcance de transmisión. En nuestro caso, debido a que se trata de un diseño orientado a bajo consumo y además las especificaciones de alcance de comunicación utilizando el pin RFIO supera ampliamente los objetivos del proyecto, el pin RFO se pone a tierra.

3.6. Salida de reloj

El ZMD44102 provee una salida de reloj (CLKO) que puede ser utilizado como reloj de otros dispositivos externos. El diseño planteado en este proyecto CLKO no será utilizada, por lo que el pin CLKO queda sin conexión.

3.7. Interfaces

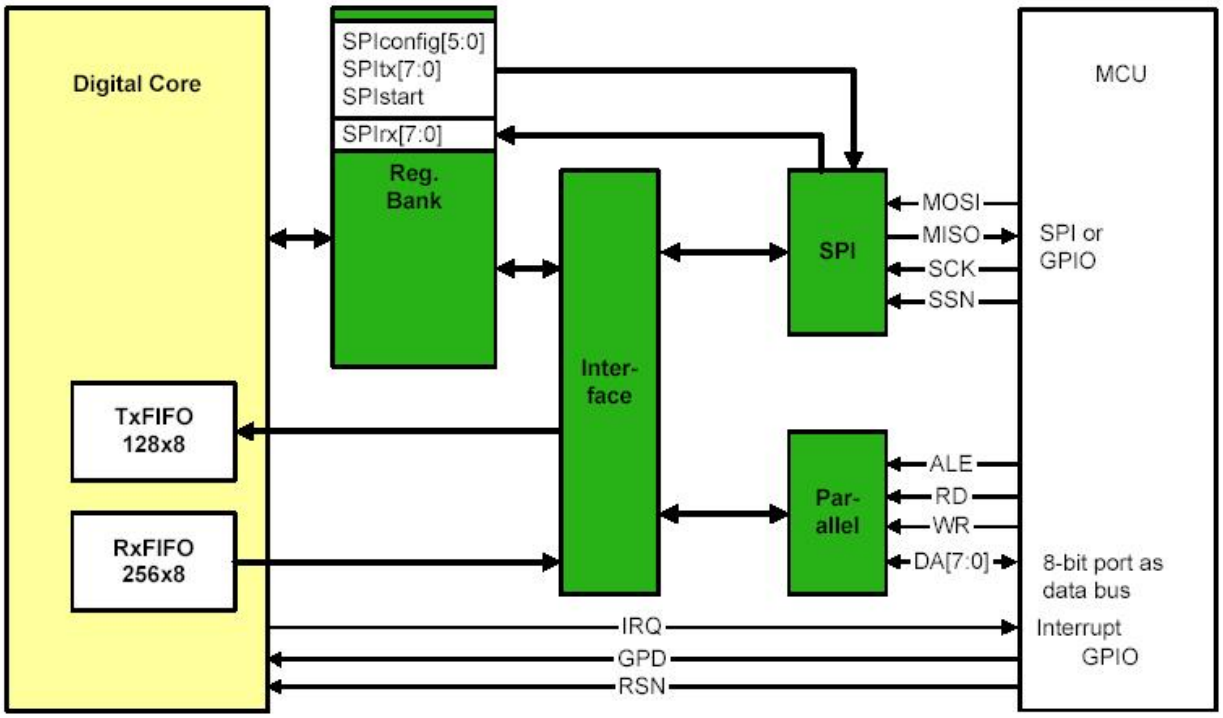

Figura 3.7: Diagrama de bloques de las interfaces

El ZMD44102 tiene dos interfaces: una interfaz serie estándar y una interfaz paralela. Ambas interfaces pueden ser utilizadas para acceder al banco de registros interno, así como a las FIFOs de transmisión (TxFIFO) y de recepción (RxFIFO). Adicionalmente tienen una salida IRQ, una entrada de apagado (GPD) y un reset (RSN), tal como se observa en la figura 3.7.

Por defecto, tanto la interfaz paralela como la serie están disponibles en modo esclavo. Para operar correctamente, la interfaz que no se utilize debe de ser deshabilitada. La interfaz paralela es deshabilitada poniendo RD, WR y ALE en alto y colocando el bus DataAddress[7:0] en estado High-Z. En nuestro caso, como estaremos utilizando la interfaz paralela, para deshabilitar la interfaz serie se debe poner SSN en alto.

Ambas interfaces pueden operar solamente si el núcleo digital del transceiver está activo.

El puerto paralelo consiste en un bus de datos bi-direccional DataAddress[7:0] y en las entradas de control read (RD), write (WR) y address latch enable (ALE). La dirección del bus DA[7:0] es controlada por la entrada RD. Si RD está en nivel alto, los pines DA[7:0] están en modo de entrada. Al subir RD se pone DA[7:0] en modo de salida luego de un tiempo de retardo (o delay) de t_{rvd} (ver tabla y diagramas de tiempo 3.8).

Parametro	**Descripción**	**Tiempo mínimo**	**Tiempo máximo**
t_{as}	Address setup time	0	
t_{ah}	Address hold time	200ns	
t_{ad}	Address to data time	0	
t_{ds}	Data setup time	0	
t_{dh}	Data hold time	0.5μs	
t_{ar}	Address to RD low time	0	
t_{zd}	High-Z to data time	0	10ns
t_{rvd}	Read low to valid data		0.5μs

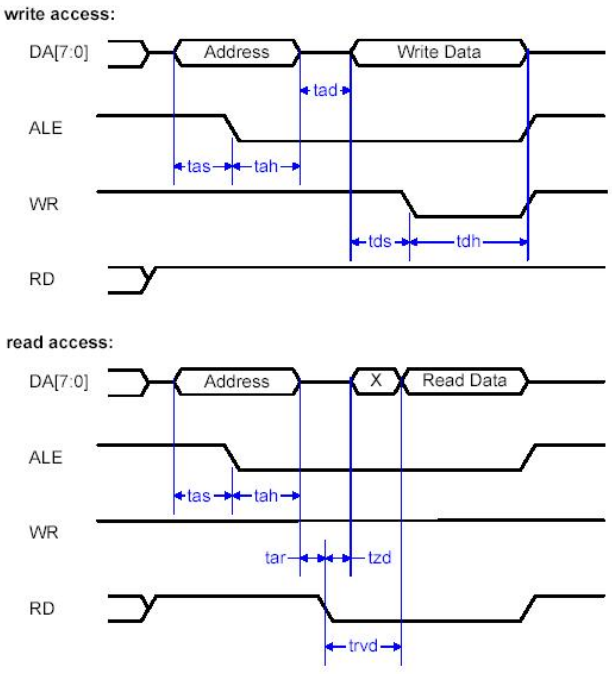

Figura 3.8: Tabla y diagramas de tiempo para escritura y lectura en modo paralelo

3.8. Encendido

La secuencia de encendido del transceiver se muestra en la figura 3.9.

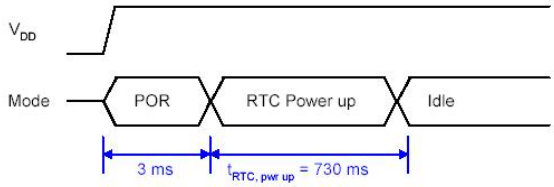

Figura 3.9: Secuencia de encendido

Un tiempo de encendido de 730 ms es necesario para estabilizar el reloj de tiempo real (RTC) de 32.768 kHz. A partir de que transcurre este tiempo, el ZMD44102 queda totalmente funcional. Los registros pueden ser accedidos luego de transcurrido POR. Para algunas funcionalidades el RTC es requerido.

3.9. Reset

El ZMD44102 cuenta con un pin de reset ($\overline{RST}$ pin 19). Este pin es activo en nivel bajo, es decir que el reset se inicia cuando baja el nivel de la señal. El modo en que se inicia el transceiver luego del reset, es controlado por el tiempo que se mantiene en nivel bajo $\overline{RST}$, en la figura 3.10 se muestra el reset simple, que lleva al transceiver al modo IDLE. En la tabla 3.1 se pueden observar los tiempos requeridos para pasar a los distintos modos[4].

Tabla 3.1: $\overline{RST}$ / modos

Modo	$\mathbf{t}_{RSN}$
Idle, Tx, or Rx	500 μs
Sleep o Global Power Down	3 ms
Off	730 ms

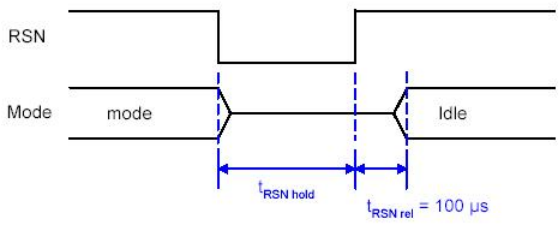

Figura 3.10: Secuencia de reset

[4]Ver 3.10 por más información sobre los diferentes modos de operación del ZMD44102

3.10. Modos de operación

A continuación se mencionan brevemente los diferentes modos de operación del ZMD44102.

3.10.1. Modos de manejo de consumo

El ZMD44102 tiene cinco modos diferentes de manejo de consumo. Estos modos pueden ser configurados por el usuario y son controlados por el microcontrolador externo. Los modos de manejo de consumo son:

- Global Power Down (GPD): El ZMD44102 puede ser forzado a entrar en este modo en cualquier momento subiendo de nivel el pin GPD. La figura 3.11 muestra un diagrama de tiempo en que se pasa de modo Idle al modo Global Power Down. En Global Power Mode el oscilador de cristal de 24MHz y la parte analógica del transceiver son apagadas, dejando solamente el oscilador de cristal de 32.768KHz en funcionamiento.

- Idle: en este modo los osciladores de cristal permanecen encendidos y la etapa analógica se apaga, dejando el transceiver en espera de entrada a cualquiera de los otros modos.

- Sleep: el modo Sleep puede ser accedido desde el modo Idle a través del registro Mac Control Commands (0xA0). A diferencia del modo Idle, solamente el oscilador de cristal de 32.768 kHz se mantiene encendido durante el modo Sleep. Una vez que se ingresa en el modo Sleep, no es posible acceder a los registros desde los puertos serie o paralelo. El ZMD44102 puede ser forzado a abandonar el modo Sleep con un pequeño pulso en el pin GPD (ver figura 3.12).

- Off: en el modo Off o apagado general, ambos relojes y osciladores de cristal son desactivados. La figura 3.13 muestra las posibles secuencias para entrar y salir del modo Off (por mayor información referirse a la página 38 de la DatasheetZmd[16]).

- Tx/Rx: En los modos Transimisión o Recepción todos los relojes y osciladores se encuentran activos, así como la etapa analógica del transceiver.

En la tabla 3.2 se indica con la letra X los bloques y/o etapas que se mantienen activas en cada modo de operación. En la última columna se indican los valores de corriente típica de consumo del transceiver en cada etapa (para una alimentación de 2.4 V).

Notas:

- El valor de corriente típica para Tx es dado para 0 dBm de potencia de transmisión.

- El valor de corriente típica para Rx es dado durante la recepción de una trama.

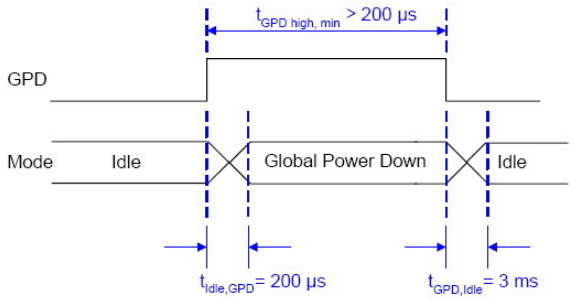

Figura 3.11: Modo Global Power Down

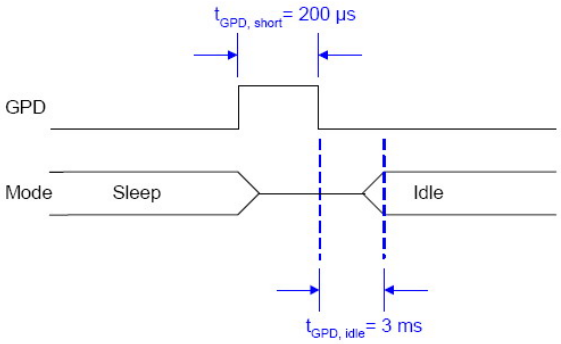

Figura 3.12: Modo Sleep

3.10.2. Modos de Tx y Rx

Modos de Transmisión

El ZMD44102 tiene cinco modos de transmisión, además de la generación automática de tramas de reconocimiento (acknowledgment). Los cinco modos se subdividen en dos modos ranurados y tres modos no ranurados.

Modos ranurados:

Tabla 3.2: Modos de operación y consumo

Modo de operación	**Oscilador 24 MHz**	**Oscilador 32.768 kHz**	**Lógica RTC**	**Contenido de registros**	**Etapa analógica**	**Corriente típica (915 MHz)**
Global Power Down		X		X		2.2μA
Idle	X	X	X	X		1.6μA
Sleep		X	X	X		2.3μA
Off						1.3μA
Tx/Rx	X	X	X	X	X	28/27μA

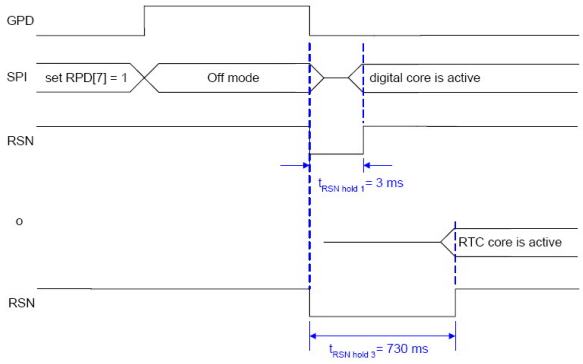

Figura 3.13: Modo Off

- GTS: Para aplicaciones de baja latencia o aplicaciones que requieren determinado ancho de banda, el coordinador de la PAN[5] puede destinar porciones del superframe (ver sección 3.10.4) a esa aplicación en particular. Estas porciones se denominan Guaranteed Time Slots (GTS).
- CSMA ranurado: Acceso múltiple con detección de portadora (ranurado).
- Transmisión directa

Modos no ranurados:

- CSMA no ranurado: Acceso múltiple con detección de portadora (no ranurado).
- Transmisión directa.

Por mayor información referirse a la sección 4.11.2 de la DatasheetZmd[16].

Modo de recepción

El transceiver puede acceder al modo Rx de varias maneras: escribiendo el comando RxOn en el registro Mac Control Commands (0xA0), a partir de la llegada de una trama de regreso de una transmisión (proveniente de los modos Tx o Generación Automática de Beacons), a partir del modo Tx para poder recibir un reconocimiento (Acknowledge), o automáticamente para resumir un ciclo de respuesta de espera de trama pausado (proveniente de los modos Seguimiento de Beacons o Generación Automática de Beacons).

Por mayor información referirse a la sección 4.11.3 de la DatasheetZmd[16].

3.10.3. Modos de Escaneo

El ZMD44102 soporta cuatro modos de búsqueda o de escaneo, listados a continuación. El modo de escaneo es seleccionado a través del registro Mac Scan Mode Configuration (0xB3).

[5] **PAN**: Personal Area Network

- ED: Durante el escaneo ED (Energy Detection), se realiza una detección de nivel de energía de forma repetitiva hasta que el tiempo del escaneo expira.
- Escaneo Activo: Permite localizar la transmisión de tramas de beacons (ver sección 3.10.4) de los coordinadores. En este modo la primera trama que se transmite es una trama del tipo "beacon request".
- Escaneo Pasivo: Similar al Escaneo Activo, excepto que la transmisión de la trama "beacon request" se saltea y se va directo a la fase de recepción/escaneo.
- Escaneo Huérfano: Permite que un dispositivo pueda reubicar a su coordinador luego de una pérdida de sincronización.

Por mayor información referirse a la sección 4.11.4 de la DatasheetZmd[16].

3.10.4. Beacons

La norma IEEE 802.15.4 define cuatro estructuras de trama:

- Trama beacon, utilizada por el coordinador para transmitir beacons.
- Trama de datos, descrita en la sección A.
- Trama de reconocimiento (acknowledgment), utilizada para confirmar correcta recepción de tramas.
- Trama de comando MAC.

En redes del tipo "beacon enabled" las tramas beacon se transmiten periódicamente por el coordinador para confirmar su presencia frente a los otros nodos de la red y enviar información de sincronización. En este tipo de red los nodos solamente necesitan estar activos mientras se está transmitiendo un beacon; el resto del tiempo los nodos pueden estar dormidos para ahorrar consumo. En las redes del tipo "non-beacon" algunos nodos están siempre activos mientras otros se mantienen dormidos la mayor parte del tiempo.

Modos de Generación Automática de Beacons

El modo de Generación Automática de Beacons es utilizado por los coordinadores para transmitir tramas beacon de forma periódica.

Por mayor información referirse a la sección 4.11.5 de la DatasheetZmd[16].

Modo de Seguimiento de Beacons

El modo de Seguimiento de Beacons es utilizado por los dispositivos de una red "beacon-enabled" para seguir a las tramas beacon del coordinador.

Por mayor información referirse a la sección 4.11.6 de la DatasheetZmd[16].

Modo de Configuración Superframe

La norma IEEE 802.15.4 permite utilizar una estructura de trama del tipo superframe. El formato de la trama superframe es definido por el coordinador de la red. La trama superframe está limitada por dos tramas beacon y tiene un largo variable $SD = 960 \cdot 2^{SO}$ (con SO entre 0 y 14).

Por mayor información referirse a la sección 4.11.7 de la DatasheetZmd[16].

Capítulo 4

Descripción del microcontrolador

4.1. Descripción general

El integrado MSP430F149 pertenece a la familia de microcontroladores de ultra-bajo consumo MSP430. La arquitectura, combinada con cinco modos de bajo consumo, es optimizada para obtener una duración prolongada de la batería en aplicaciones de sistemas de mediciones portables.

El MSP430F149 incorpora un procesador RISC de 16 bits, registros de 16 bits, y un oscilador controlado digitalmente (DCO)[1] que permite pasar de modo de bajo consumo a modo activo en menos de 6 μs. El microcontrolador trae integrado dos timers de 16 bits, un conversor analógico digital (ADC) de 12 bits, 2 interfaces universales de comunicación serie síncrona/asíncrona (USART)[2], y 48 pines de entrada y salida.

Aplicaciones típicas incluyen sistemas de sensado que capturan señales analógicas y las convierten en valores digitales, para luego procesarlas y transmitir los datos a un sistema remoto.

En el Apéndice G de ésta documentación se describen las características eléctricas de operación y las características generales más destacadas del MSP430F149, así como su distribución de pines (pinout). Por información más detallada referirse a la hoja de datos del MSP430F149[18].

[1] **DCO** - El oscilador controlado digitalmente o DCO (digitally controlled oscillator) consiste en un generador de forma de onda digital que incrementa un contador de fase de a una muestra. Esta fase es entonces buscada en una tabla de forma de onda para crear una sinusoide. Es entonces que los rangos de frecuencia que pueden ser producidos están limitados solamente por la precisión aritmética utilizada para computar la fase. Al mismo tiempo, los DCOs son ágiles en cambio de fase (y por ende frecuencia) y pueden ser modificados de forma trivial para producir salidas FM, PM o en cuadratura.

[2] **UART y USART** - La UART (universal asynchronous receiver/transmitter) es un tipo de hardware que traduce datos entre interfaces serie y paralelas. Para comunicaciones serie, la UART convierte bytes de datos en streams de bits asíncronos representados como impulsos eléctricos binarios, y viceversa. Al tratarse de una comunicación asíncrona, la UART utiliza bits de comienzo y parada para indicar el comienzo y el fin de las tramas. La USART (universal synchronous/asynchronous receiver/transmitter) implementa transimisión síncrona además de asíncrona, utilizando la información del reloj sin la necesidad de utlizar bits de comienzo y parada. Esto mejora la eficiencia de la transmisión, ya que se pueden transmitir más bits de datos en una trama.

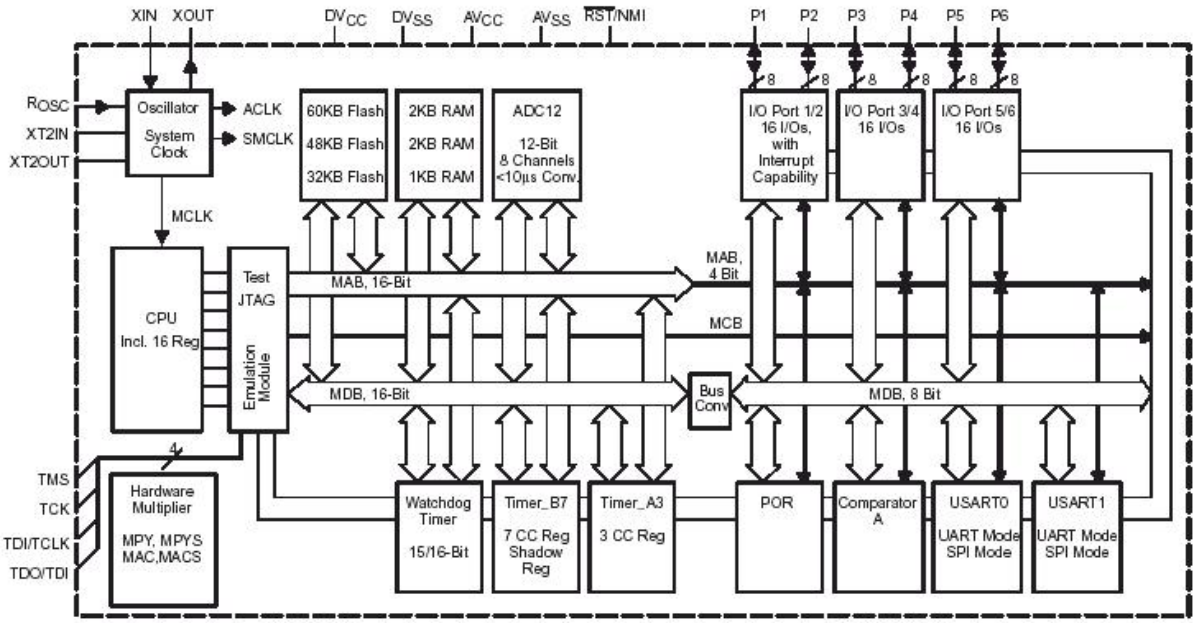

Figura 4.1: Diagrama de bloques del MSP430F149

4.2. CPU

La CPU del MSP430 tiene una arquitectura RISC de 16 bits y todas las operaciones, excluyendo las instrucciones de flujo de programa, se realizan con operaciones de registro en conjunción con siete modos de direccionamiento para operandos de origen y cuatro modos de direccionamiento para operandos de destino.

La CPU está integrada a 16 registros. El tiempo de ejecución de operación registro-a-registro es de un ciclo de reloj del procesador. Cuatro de los registros, R0 a R3, son designados como *program counter*, *stack pointer*, *status register* y *constant generator* respectivamente. Los restantes registros son registros de propósito general. Los periféricos se conectan a la CPU a través de los buses de datos, dirección y control, y pueden ser manejados con todas las instrucciones.

El set de instrucciones consiste en 51 instrucciones con tres formatos y siete modos de direccionamiento. Cada instrucción puede operar en datos de byte o de palabra.

4.3. Sistema de reloj y oscilador

El módulo de reloj de la familia MSP430x13x y MSP430x14x soporta un oscilador de cristal de 32,768 kHz, un oscilador controlado digitalmente (DCO) interno, y un oscilador de cristal de alta frecuencia. El DCO se enciende y estabiliza en menos de 6 μs.

El módulo de reloj básico provee las siguientes señales de reloj:

- ACLK: Reloj auxiliar alimentado desde un oscilador de cristal de 32,768 kHz o de alta frecuencia.
- MCLK: Reloj primario utilizado por la CPU.
- SMCLK: Reloj secundario utilizado por los módulos periféricos.

4.4. Modos de operación

El MSP430 tiene un modo activo y cinco modos de bajo consumo seleccionables por software. Un evento de interrupción puede despertar el dispositivo de cualquiera estos cinco modos de operación de bajo consumo, atender la interrupción, y luego restaurar el modo de bajo consumo al salir de la rutina de interrupción.

Los siguientes seis modos de operación pueden ser configurados por software:

- Modo Activo (AM)
  - Todos los relojes están activos
- Modo de bajo consumo 0 (LPM0)
  - La CPU se desactiva
  - ACLK y SMCLK se mantienen activas
  - MCLK se desactiva
- Modo de bajo consumo 1 (LPM1)
  - La CPU se desactiva
  - ACLK y SMCLK se mantienen activas
  - MCLK se desactiva
  - El generador de continua del DCO es desactivado, si DCO no es utilizado en modo activo
- Modo de bajo consumo 2 (LPM2)
  - La CPU se desactiva
  - MCLK y SMCLK se desactivan
  - ACLK se mantiene activa
  - El generador de continua del DCO se mantiene activo
- Modo de bajo consumo 3 (LPM3)

- La CPU se desactiva
- MCLK y SMCLK se desactivan
- ACLK se mantiene activa
- El generador de continua del DCO es desactivado

- Modo de bajo consumo 4 (LPM4)
  - La CPU se desactiva
  - MCLK y SMCLK se desactivan
  - ACLK se desactiva
  - El generador de continua del DCO es desactivado
  - DCO se detiene

4.5. Periféricos

Los periféricos se conectan a la CPU a través de los buses de datos, dirección y control, y pueden ser manejados utilizando todas las instrucciones.

4.6. Entrada/salida digital

El MSP430 tiene seis puertos de entrada/salida de 8 bits (puertos P1 a P6) con las siguientes características:

- Todos los bits de entrada/salida son programables de forma independiente.
- Cualquier tipo de combinación de entrada, salida, y condición de interrupcion es posible.
- Capacidad de selección de flanco para interrupciones de entrada en los 8 bits de los puertos P1 y P2.
- Todas las instrucciones soportan acceso de escritura/lectura a los registros de control de los puertos.

4.7. Contador watchdog

La principal funcion del contador watchdog (WDT) es realizar un reinicio controlado del sistema en el caso en que ocurra un problema de software. Si el intervalo de tiempo seleccionado expira, entonces se genera el reinicio del sistema. Si la funcion de watchdog no es utilizada en ninguna aplicación, el módulo

puede ser configurado como un contador de intervalos y puede generar interrupciones en intervalos de tiempo determinados.

4.8. Multiplicador de hardware

La operación de multiplicación es soportada por un módulo periférico dedicado, el cual realiza operaciones de bits de 16x16, 8x16 y de 8x8. El módulo soporta operaciones de multiplicación y de multiplicación-acumulación con signo y sin signo. El resultado de la operacion puede ser accedido inmediatamente luego de que los operandos son guardados en los registros de los periféricos, sin necesidad de esperar ciclos de reloj adicionales.

4.9. USART0

La familia MSP430x13x y MSP430x14x tiene un módulo periférico de transmisión y recepción síncrona/asíncrona (USART0) para comunicación serie. La USART0 soporta SPI síncrona (de 3 o 4 pines) y protocolos asíncronos de comunicación UART, por medio de canales de transmisión y recepción de doble buffer.

4.10. USART1

La familia MSP430x14x y MSP430x14x1 tiene un módulo periférico secundario de transmisión y recepción síncrona/asíncrona (USART1) para comunicación serie. La USART1 soporta SPI síncrona (de 3 o 4 pines) y protocolos asíncronos de comunicación UART, por medio de canales de transmisión y recepción de doble buffer.

La operación de USART1 es idéntica a la de USART0.

4.11. Comparator_A

La función primaria del módulo comparador A es brindar pendientes de precisión en conversiones A/D, supervisión de voltaje de batería, y monitoreo de señales analógicas externas.

4.12. ADC12

El módulo ADC12 soporta conversiones analógica-digitales de 12 bits. El módulo implementa:

- Núcleo RAS[3] de 12 bits.
- Selección de control de muestras.
- Generador de referencia.
- Almacenamiento de 16 muestras analógicas independientes sin intervención de la CPU.

4.13. Timer_A3

El timer A3 es un contador de 16 bits con tres registros de captura y comparación. El timer A3 soporta múltiples capturas-comparaciones, salidas PWM, y medición de intervalos. Además puede generar interrupciones desde el contador en condiciones de overflow y desde cada registro de captura y comparación.

4.14. Timer_B7

El MSP430x14x tiene además un contador de 16 bits con siete registros de captura y comparación. El timer B7 soporta múltiples capturas-comparaciones, salidas PWM, y medición de intervalos. Además puede generar interrupciones desde el contador en condiciones de overflow y desde cada registro de captura y comparación.

[3] **RAS** - El Registro de Aproximaciones Sucesivas o RAS convierte la señal analógica en bits de salida, dentro de la arquitectura de conversor A/D que utiliza en este microcontrolador.

Capítulo 5

Antena y red de adaptación

5.1. Antena

En sistemas inalámbricos orientados a bajo consumo, un diseño óptimo de antena es fundamental para obtener un rango de transmisión de datos aceptable. En nuestro caso particular, como uno de los requerimientos del sistema, el transmisor debería ser capaz de transmitir al menos -3dBm de potencia, con un alcance no menor de 10 metros de distancia.

En la actualidad existe una gran gama de diseños de antenas que transmiten en el rango de frecuencia de los 915MHz. Este valor de frecuencia permite lograr diseños de antena muy compactos. Como nuestra aplicación en particular refiere a un enlace RF de corta distancia de una placa portátil y de pequeño tamaño, nuestro diseño se orienta a antenas hechas en PCB.

Dentro de las antenas hechas para PCB, se destacan las antenas en espiral, los dipolos y monopolos doblados, y las antenas de loop, entre otras. ZMD ha experimentado con varios diseños de antena y recomienda el monopolo doblado de la Figura 5.2.

Este tipo de antenas al igual que las antenas en espiral, no requieren un plano de tierra (incluso deben

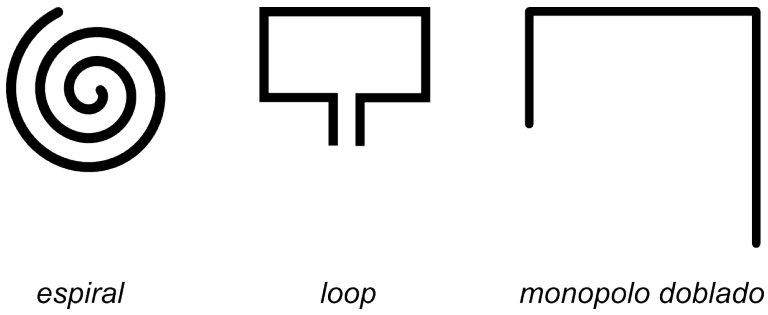

Figura 5.1: Algunos tipos de antena de PCB

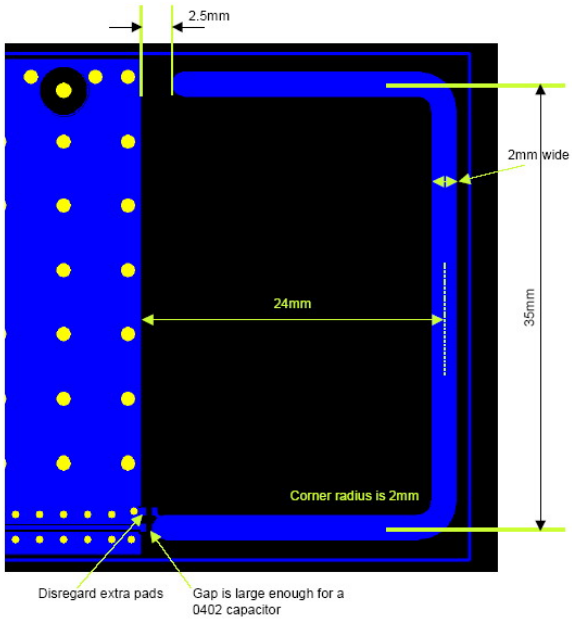

Figura 5.2: Monopolo doblado propuesto por ZMD

colocarse alejadas de los componentes y posibles planos de tierra). La principal desventaja es la fácil desintonización por la cercanía de objetos o la manipulación humana. Las antenas de loop poseen mayor inmunidad al contacto de las manos lo cual hace que sean unas de las más utilizadas en aplicaciones similares a este proyecto. Las antenas de loop también presentan independencia respecto a los planos de tierra, sin embargo son antenas de baja ganancia. La ganancia de la antena depende principalmente de las dimensiones de la misma: cuanto menor es el área del loop menor es su ganancia. Esto introduce un compromiso en el momento de reducir el tamaño de loop en busca de reducir el área del circuito total.

Las antenas de loop pueden tener variadas formas: rectangular, cuadrada, circular, triangular, romboide; y en varios casos puede presentar más de un loop. Estas antenas, a diferencia del monopolo doblado y de las antenas en espiral, tienen su entrada tal que un extremo del loop va conectado a tierra y el otro extremo se conecta a la línea de transmisión/recepción vía un capacitor de sintonización. Como las antenas de loop presentan impedancia inductiva, se utilizan uno o varios capacitores para sintonizar la antena. Los capacitores pueden ubicarse en diferentes lugares del loop dependiendo del diseño de la antena.

Las antenas construidas para realizar las pruebas de transmisión de la placa fueron dos:

- Antena de loop basada en las recomendaciones de diseño de Microchip[29].
- Monopolo doblado basado en la recomendación de diseño de ZMD[19].

Ambas antenas fueron construidas en Eagle como placas independientes, de forma que quedaran separadas de la placa que contiene al transceiver. De esta manera se facilitan las pruebas tanto de potencia como de las

redes de adaptación. En cada una de las antenas se dejó previsto un conector SMA para poder acoplarlas al conector de la placa principal, así como una red de adaptación PI para adaptar las impedancias características de las antenas (de ser necesario) con la línea de 50 Ohm proveniente de la placa principal.

5.1.1. Monopolo doblado

ZMD sugiere tener las siguientes consideraciones de diseño:

- Evitar partes metálicas en ninguna de las capas del área de la antena.
- Para evitar corrosión, el cobre de la antena debe ser bañado con una muy delgada capa de oro o niquel.
- Utilizar un capacitor en serie para la sintonización fina de la antena. ZMD utilizó en su diseño un capacitor de 5.6pF, pero para cada diseño su valor puede variar en pequeña proporción.
- La antena debe ser alimentada por una línea de impedancia característica de 50 Ohm.

Todas las consideraciones fueron tomadas en cuenta excepto la segunda, debido a temas de costos (una vez que se elija una antena para el diseño integrado es altamente recomendable aplicar algunos de estos tratamientos para evitar la corrosión y prolongar la vida útil de la antena). Para la sintonización de la antena se utilizaron dos capacitores en paralelo, cada uno de ellos de valor 2.7pF.

Los monopolos tienen máxima radiación en el plano normal al eje de la antena y mínima radiación en la dirección del eje, por lo tanto el monopolo debe colocarse en posición vertical para que sea omnidireccional. En el monopolo doblado se curva la pista de la antena y disminuye la ganancia en el plano de máxima radiación. Sin embargo, se registra una radiación más uniforme en el plano de mínima radiación (plano horizontal al PCB), mejorando el rendimiento en el caso en que no sea posible colocar el PCB en posición vertical.

5.1.2. Modelado y diseño de la antena de loop

El modelo teórico de la antena se basó en las recomendaciones de Microchip para su producto RFPIC[29], las cuales se resumen a continuación.

En la figura 5.3 se muestra el circuito equivalente para una antena de loop pequeña. El circuito consiste en una resistencia de radicación R_{rad} en serie con una resistencia de pérdidas R_{loss} y una inductancia L. R_{rad} representa la energía RF irradiada por la antena mientras que R_{loss} modela las pérdidas.

Asumiendo un flujo de corriente I constante a través del loop, la potencia total entregada a la antena se puede expresar entonces como:

$$P_{total} = I^2 \cdot (R_{total}) = I^2 \cdot (R_{rad} + R_{loss}) \tag{5.1}$$

La impedancia total de la antena $Z_{total} = R_{total} + jwL$ se puede calcular a partir de las siguientes expresiones (obtenidas de las referencias [30] y [31]):

$$R_{rad} = 31.171 \frac{A^2}{\lambda^4} \tag{5.2}$$

$$R_{loss} = \frac{1}{2a} \cdot \sqrt{\frac{\pi \cdot f \cdot \mu}{\sigma}} \tag{5.3}$$

$$L = \frac{\mu}{2\pi} \cdot l \cdot ln(\frac{8 \cdot A}{l \cdot a}) \tag{5.4}$$

en donde

- A representa el área del loop en metros cuadrados
- λ representa la longitud de onda a la frecuencia de radiación en metros
- a representa el ancho de la pista de cobre en metros
- f representa la frecuencia de radiación en Hertz
- μ es $4 \cdot \pi \cdot 10^{-7}$
- σ es la conductividad del cobre por metro
- w es $2 \cdot \pi \cdot f$

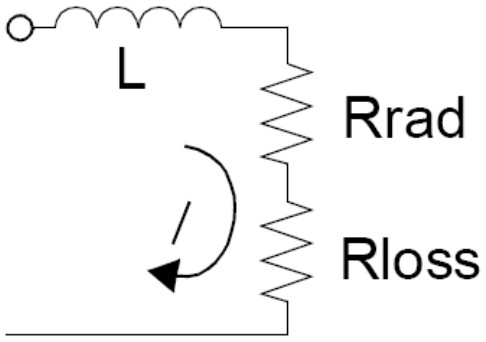

Figura 5.3: Circuito equivalente para una antena de loop pequeña

Tabla 5.1: Dimensiones antena de loop

Magnitud	**Valor**
Largo de loop grande	$20mm$
Ancho de loop grande	$10mm$
Área del loop grande	$2cm^2$
Ancho de pista	$1.2mm$
Ancho de loop chico	$2.4mm$
Largo de loop chico	$4.9mm$
Área de loop chico	$0.12cm^2$

- l es el perímetro del loop en metros

Para el caso de nuestra antena, se consideró un loop de 20 x 10 mm con un ancho de pista de 1,2 mm, obteniéndose una valor de impedancia total $Z_{total} = 0.3 + j212.7$ ($R_{total} = 0.3Ohm$, $L = 37nH$).

Para poder aumentar la impedancia obtenida, la recomendación propone agregar un segundo loop pequeño y un condensador. El acoplamiento magnético entre el loop grande y el loop pequeño actúa de forma similar a lo que sucede con los bobinados primario y secundario en un transformador, aumentando así la impedancia de la antena.

Considerando un loop pequeño de 2.4 x 4.9 cm de dimensión, se obtiene una impedancia vista de 500 Ohm, la cual puede ser fácilmente adaptada a 50 Ohm a través de una red de adaptación.

En la tabla 5.1 se resumen las dimensiones finales de nuestra antena de loop.

5.2. Red de adaptación

Para adaptar la antena de loop (impedancia 500 Ohm) a la impedancia de salida del tranceiver (50 Ohm) se diseñaron 3 posibles redes de adaptación del tipo PI (ver figura 5.4), basándose en los cálculos de adaptación de impedancia del Capítulo 4 de la referencia [32].

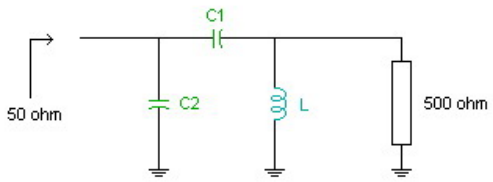

Figura 5.4: Red de adaptación PI

Las tablas 5.2, 5.3 y 5.4 muestran los valores calculados, junto con los valores reales estándar utilizados

Tabla 5.2: Red de adaptación #1

Elemento	**Valores calculados**	**Valores reales**
L	$22nH$	$22nH$
C1	$1.8pF$	$1.8pF$
C2	$2.8pF$	$2.7pF$

Tabla 5.3: Red de adaptación #2

Elemento	**Valores calculados**	**Valores reales**
L	$18nH$	$18nH$
C1	$2.3pF$	$2.4pF$
C2	$4.2pF$	$3.9pF$

para cada red de adaptación.

Tabla 5.4: Red de adaptación #3

Elemento	**Valores calculados**	**Valores reales**
L	$10nH$	$10nH$
C1	$4.4pF$	$4.3pF$
C2	$8.8pF$	$9.1pF$

A continuación se muestran los caminos de adaptación en el Diagrama de Smith para las posibles redes de adaptación, tanto para los valores calculados como para los valores reales utilizados. En los diagramas con los valores reales, a modo de dar una referencia, se muestra el lugar geométrico de los puntos con pérdidas de retorno de -20dBm (pequeño círculo con centro en el punto medio del diagrama). La herramienta de simulación utilizada fue el programa MIMP[33] (Motorola Impedance Matching Program). Esta interfaz gráfica permite evaluar la performance de una red de adaptación de impedancias a partir de los resultados de cálculos previamente hechos (así como también a partir de la observación en el diagrama de Smith).

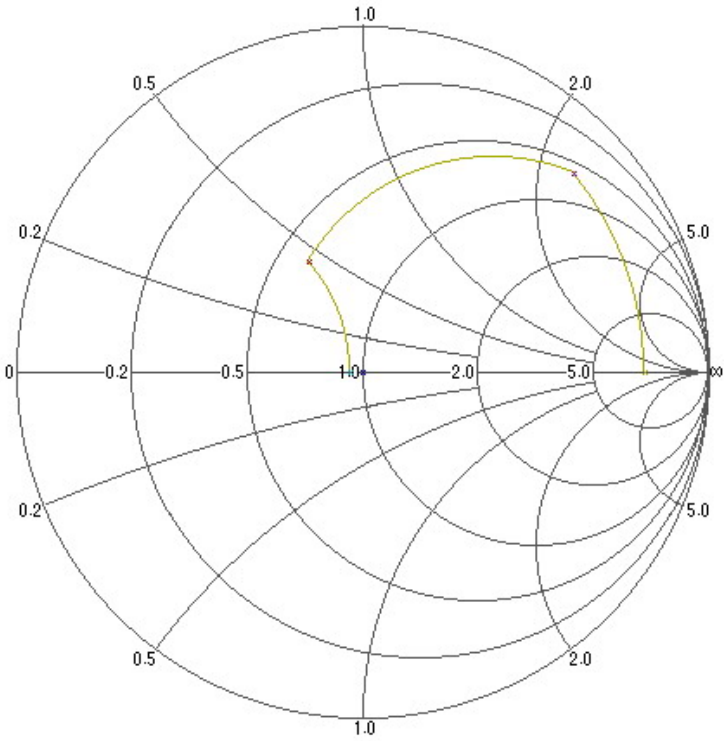

Figura 5.5: Red de adaptación #1: valores calculados

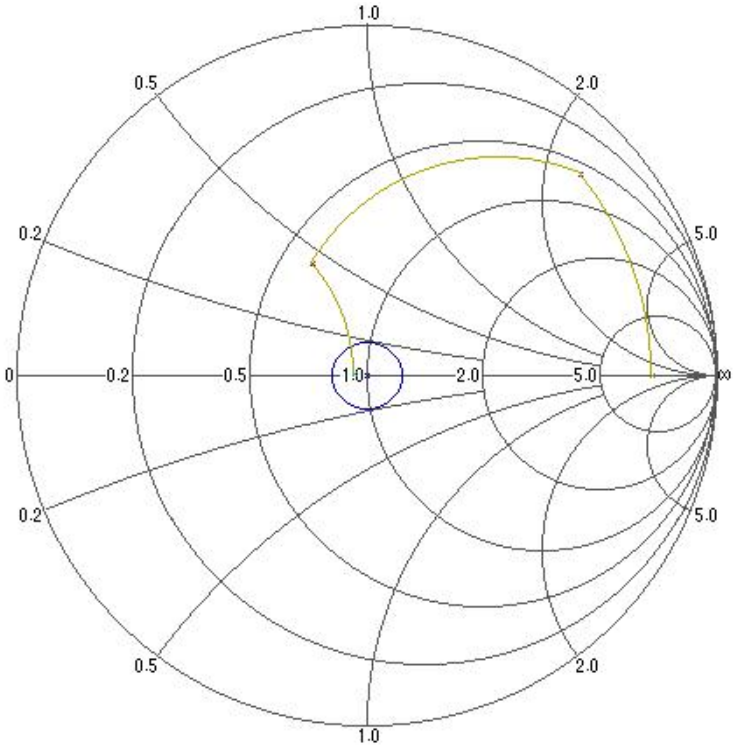

Figura 5.6: Red de adaptación #1: valores reales

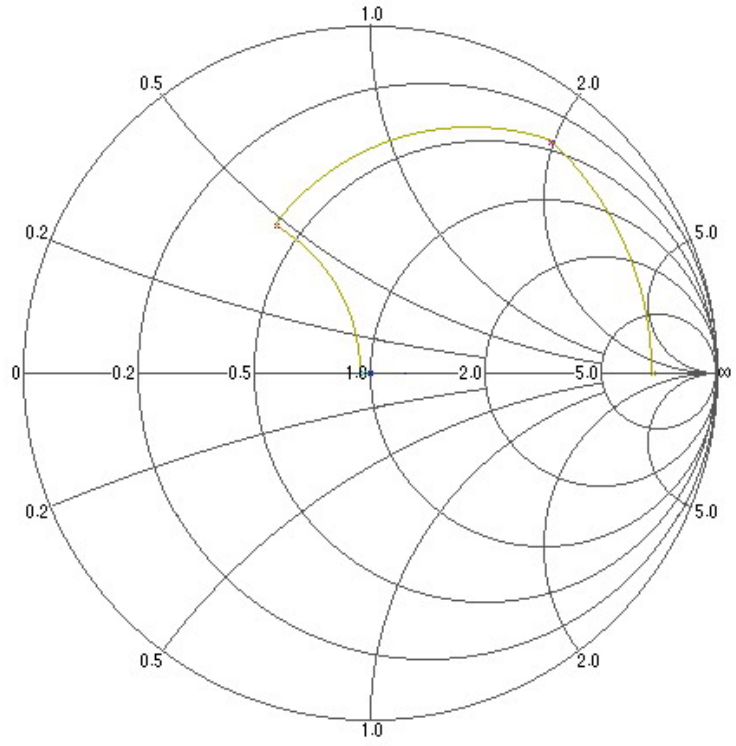

Figura 5.7: Red de adaptación #2: valores calculados

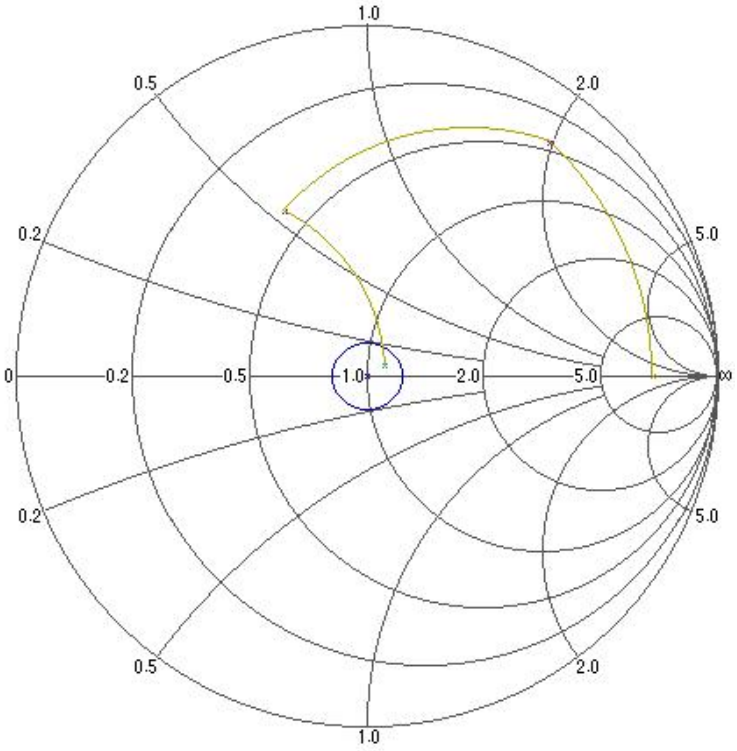

Figura 5.8: Red de adaptación #2: valores reales

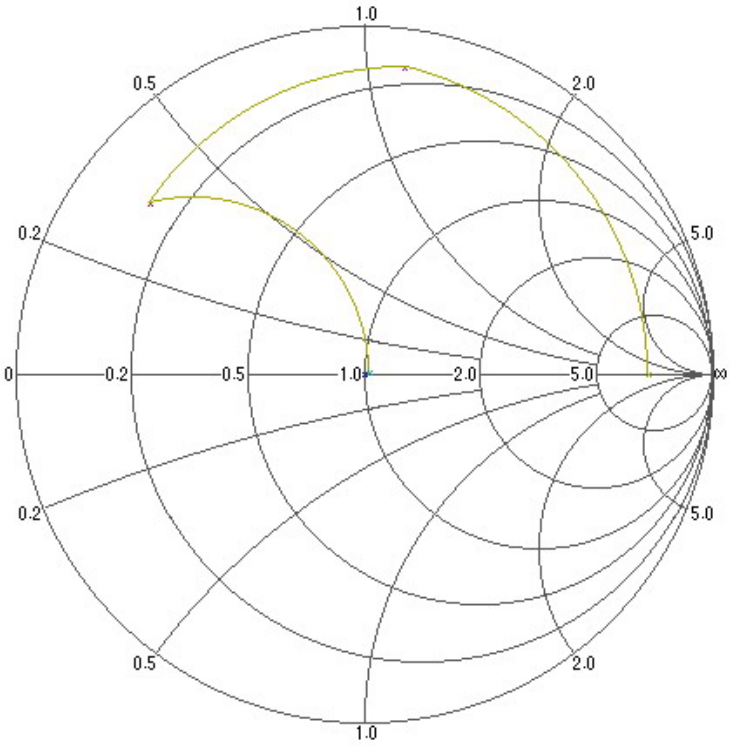

Figura 5.9: Red de adaptación #3: valores calculados

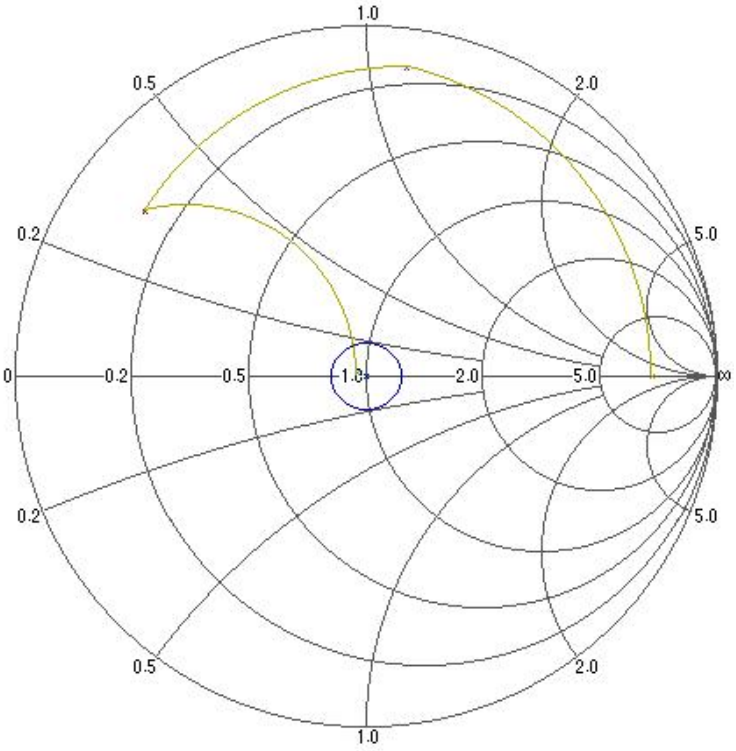

Figura 5.10: Red de adaptación #3: valores reales

Capítulo 6

Diseño y elaboración del circuito

6.1. Introducción

En el presente capítulo se describen las decisiones de diseño tomadas, analizando la realización del esquemático, los criterios de enrutamiento tomados y el posicionamiento de los componentes. Finalmente se describe la metodología utilizada en la elaboración del circuito impreso.

6.2. Diseño del circuito

6.2.1. Transceiver

Como se describió anteriormente el ZMD44102 utiliza dos relojes y un PLL (secciones 3.3 y 3.4 respectivamente), requiriendo ambos de componentes externos. Además se requieren capacitores como filtros en la alimentación del transceiver y un filtro pasa bajo en la salida de radio frecuencia. En la figura 6.2 se muestra el esquemático del prototipo.

Modo de comunicación del transceiver

El transceiver prevé 2 modos de comunicación con el microcontrolador, modo paralelo y modo serie. Se decidió usar el transceiver en modo paralelo, ya que en este modo se utilizan menos ciclos de reloj en la transferencia de datos. Al utilizar menos ciclos de reloj se puede lograr un menor consumo, siendo esto uno de nuestros requerimientos principales.

Para usar el modo paralelo, se utilizó la siguiente configuración de pines:

- SSN - activo por nivel bajo: se dejó alto para deshabilitar el modo serie

- MOSI SPI - master out, slave in: no conectado
- MISO S0PI - master in, slave out: no conectado
- SCK SPI - serial clock: no conectado

Para usar el modo de comunicación paralelo se conectaron entre el microcontrolador y el transceiver los siguientes pines:

- ALE - Address latch enable
- RD - Read data
- WR - Write data
- DATA[0..8]

Salida RF

La señal RF a la salida del transceiver (pin RFIO) es filtrada mediante un capacitor en paralelo que bloquea continua y un filtro armónico pasabajo (filtro Butterworth de 3er orden) que elimina las frecuencias armónicas no deseadas. Un conector SMA se coloca al final de la línea para conectar la antena. En la figura 6.1 se muestra el camino en el diagrama de Smith del filtro armónico utilizado. Como se puede apreciar en el diagrama, no existen cambios significativos en la impedancia vista a la salida del filtro respecto a la de entrada (el punto de llegada del camino es muy cercano al punto de salida de 50 ohms). Igualmente se dejó previsto otro camino auxiliar con posibilidad de soldar dos redes de adaptación PI, permitiendo de esta manera poder realizar cualquier tipo de ajuste fino o inclusive permitir el conexionado de antenas de impedancia vista distinta a 50 ohms.

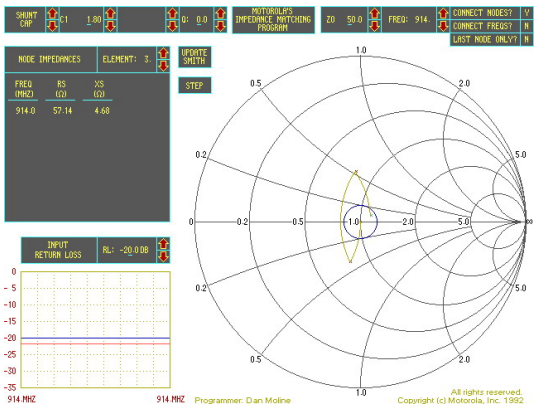

Figura 6.1: Filtro RF

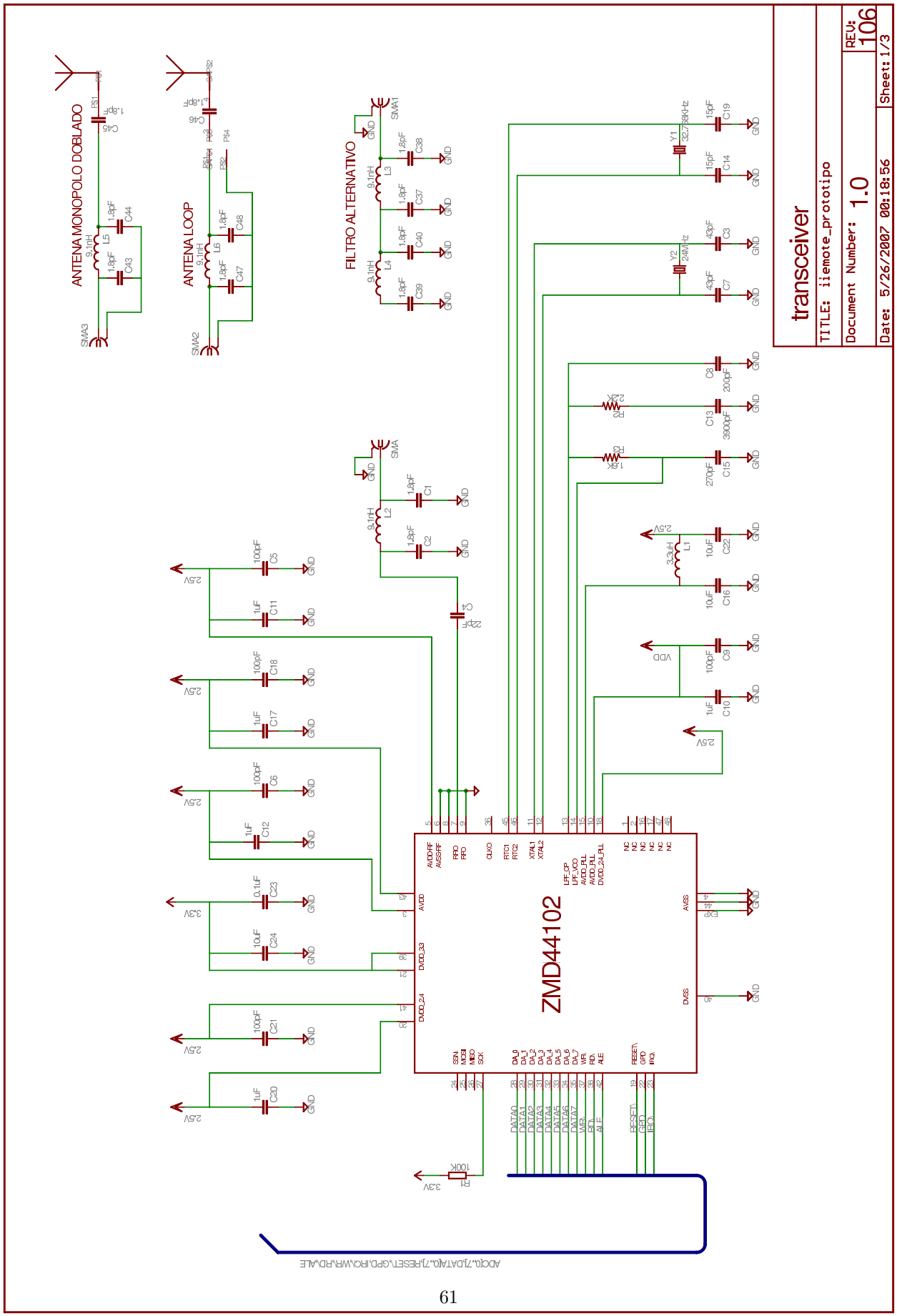

Figura 6.2: Esquemático Transceiver

6.2.2. Microcontrolador

Variación de la frecuencía con la temperatura

La frecuencia de trabajo esta dada por el "Digitally-controlled oscillator" (DCO), el cual tiene un coeficiente de variación de la frecuencia de salida muy alto respecto a la temperatura. Este coeficiente se puede reducir aproximadamente a un 0.1 %/°C, utilizando una resistencia externa R_{OSC} de 100k.

En la figura 6.3 se muestra la variación de DCO con respecto a la temperatura para los dos casos.

Siendo que hay que alcanzar velocidades de transmisión de 40 kbits por segundo, la transmisión por RS232 tiene que poder superar este requerimiento en velocidad, para así poder despachar una ráfaga larga de datos a un PC. Esto nos obliga a utilizar una frecuencia superior a la provista por el reloj de 32.768 kHz para poder mantener los tiempos del puerto UART, por lo cual necesitamos la frecuencia f_{DCO}. Es necesario que esta frecuencia sea estable y conocida, por lo tanto se tomó la decisión de emplear una resistencia externa R_{OSC} de 100K.

De esta manera se logra tener una frecuencia independiente de la temperatura, y a su vez el uso de un R_{OSC} externo nos permite llegar a frecuencias mayores.

Puertos

Se dejaron dos puertos libres del controlador para poder interactuar con el exterior (puerto 5 y puerto 6). Estos puertos sirven para reconocer estados del microcontrolador o para conectar periféricos por el puerto de ADC. A través del puerto ADC, un gran número de aplicaciones pueden ser llevadas a cabo. Por ejemplo, el microcontolador puede adquirir y procesar información de un sensor o sistema de sensado que mida información de variables externas. Para la programación del microcontrolador se utilizó la interfaz JTAG

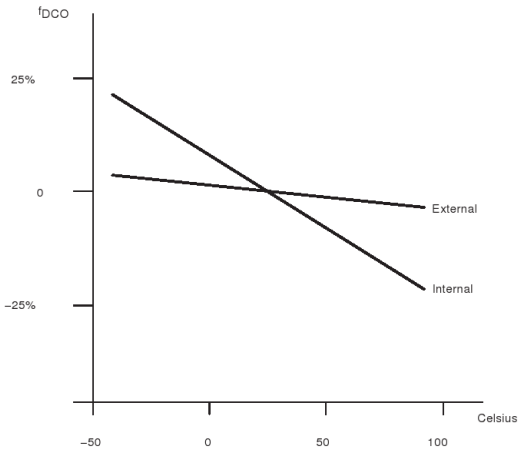

Figura 6.3: f_{DCO} Vs Temperatura

(ver apéndice C), para lo cual se dejó provisto un conector externo con seis pines conectado a TDO, TDI, TMS, TCK, VCC, y VSS.

6.2.3. Alimentación

La alimentación del circuito esta dada por una batería externa que es conectada al pin VIN+ el cual alimenta a los dos reguladores de voltaje, tal como se muestra en la figura 6.5. En los circuitos de cada regulador se usaron los filtros indicados por Texas Instruments en sus respectivas hojas de datos.

6.2.4. Adaptador de niveles RS232

Para el caso del transceiver RS232 se eligió el MAX3221 de Texas Instruments porque además de adaptar los niveles de voltaje necesarios para nuestra aplicación, posee un modo dormido y un modo autodormido. El modo autodormido lo que hace es pasar al chip a modo dormido cuando no detecta ninguna conexión. En la figura 6.4 se muestra el esquemático usado. Este circuito es el que recomienda Texas Instruments para la utilización del MAX3221. Se dejó un conector DB9 hembra para conectarse una PC.

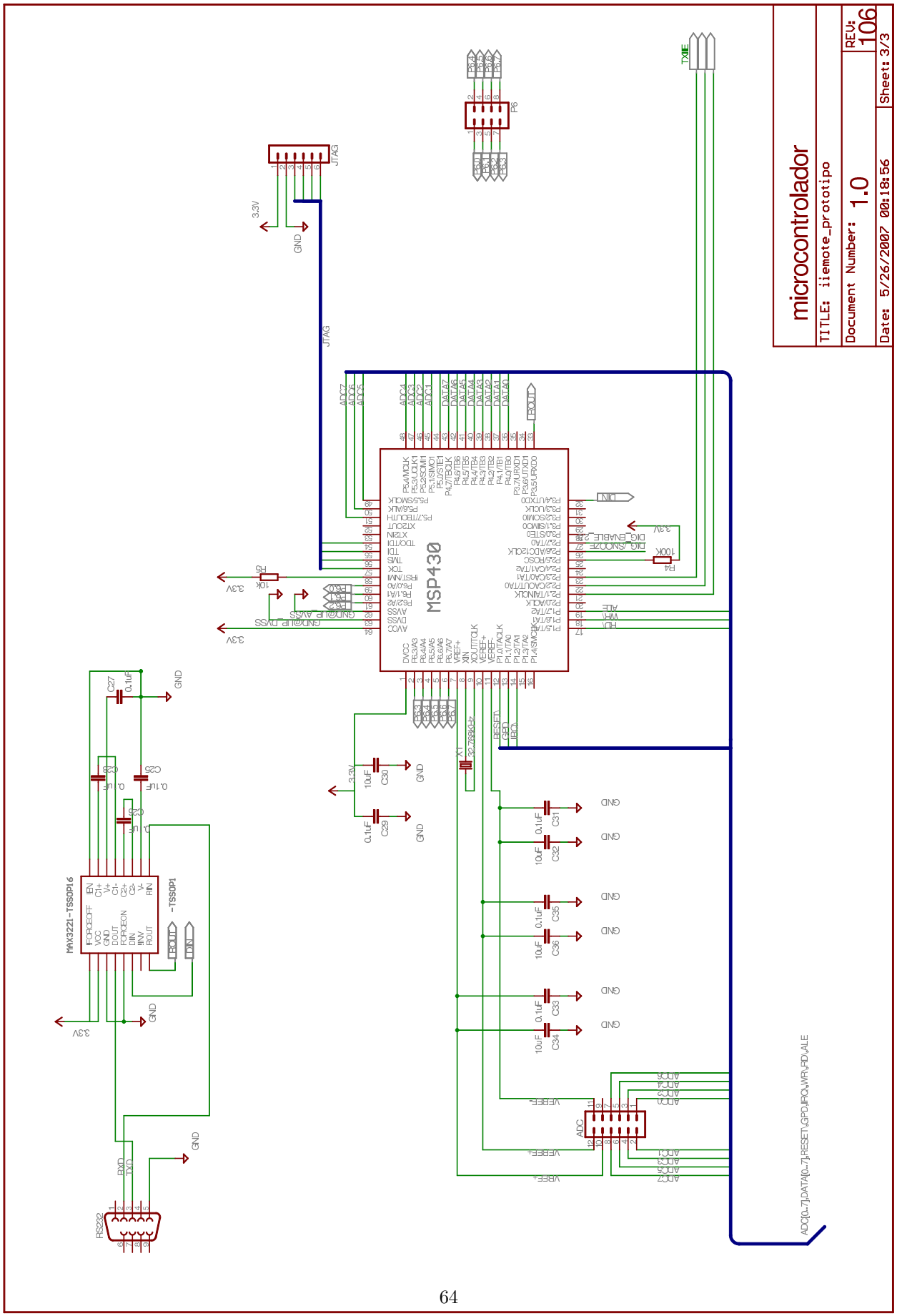

Figura 6.4: Esquemático Microcontrolador

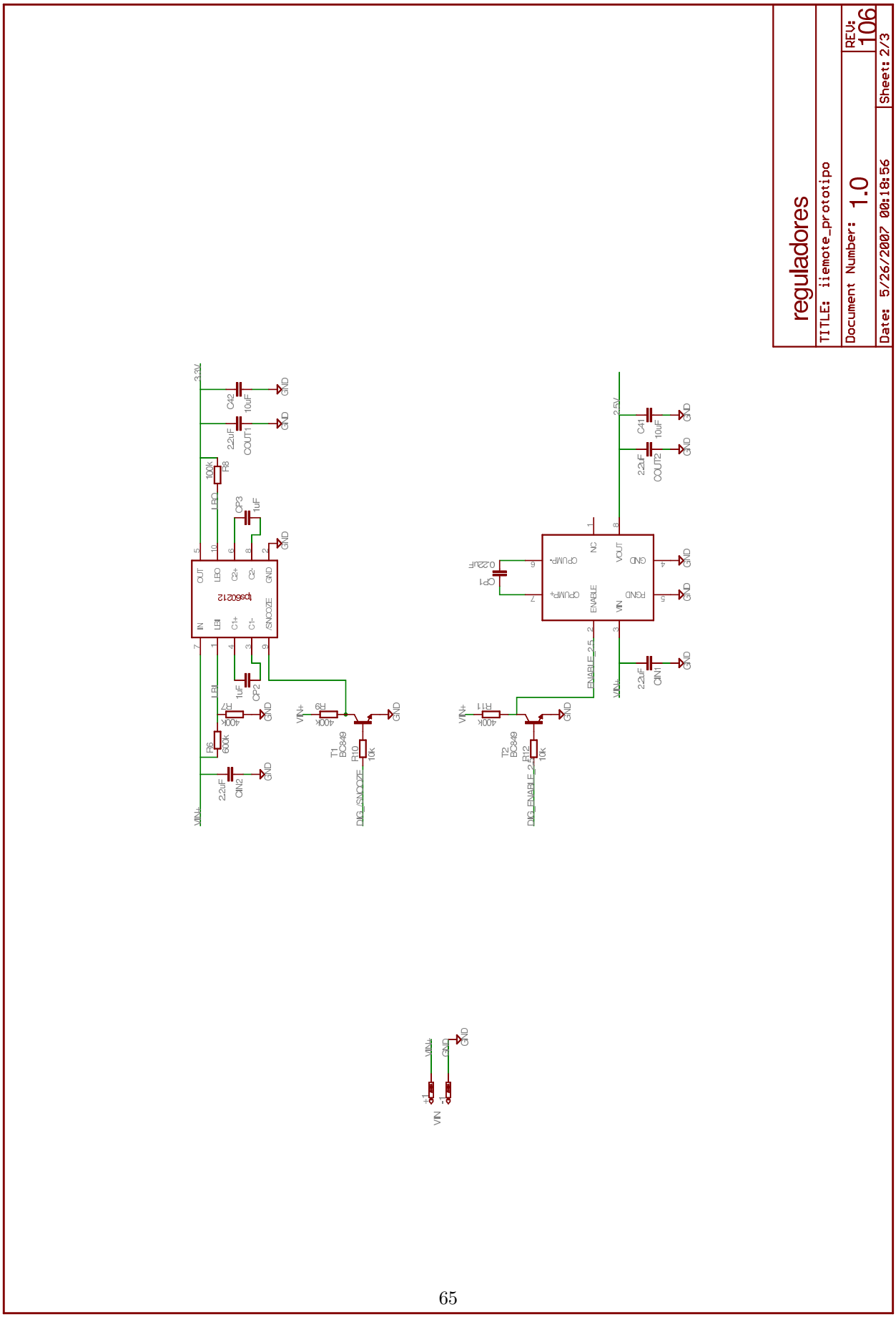

Figura 6.5: Esquemático Alimentación

6.3. Diseño del PCB

Una vez llegado el momento de evaluar la construcción del PCB, las posibilidades fueron hacer la placa en Uruguay o hacer la placa en el extranjero. La ventaja principal de hacer la placa en el extranjero era que se podría implementar un diseño de varias capas, mientras que en el caso de fabricar en Uruguay se podía lograr a los sumo dos capas y sin vias metalizadas. La principal ventaja de la realización de la placa en plaza es la fácil y rápida accesibilidad, además de un costo menor de fabricación.

Una de las variables de mayor peso al momento de tomar la decisión, fue sin duda alguna el nivel de calidad en la fabricación de las máscaras y de las vias. No se tenía seguridad de que en Uruguay se pudiera conseguir una buena calidad de máscara para la placa. La restricción en el tamaño de la placa fue otro de nuestros requerimientos principales, para lo cual se estimaron y compararon los tamaños para un diseño en dos capas y para un diseño en cuatro capas.

La decisión final fue construir el PCB en el exterior para asegurar una calidad de placa superior (confiando en que los tiempos de espera no se extendieran demasiado, comprometiendo los tiempos del $\mathbf{I_{IE}M_{OTE}}$).

Una vez tomada esta decisión, se nos recomendó la empresa PCBExpress para la construcción, ya que los tutores tenían experiencia previa con la empresa con resultados satisfactorios.

Características de la fabricación:

- 4 capas.
- Agujeros metalizados.
- Máscara antisoldante LPI (Liquid Photo Imageable) con la que se logran líneas de una precisión de 8 a 5 milésimas de pulgada (mil).
- Máscara silkscreen (leyendas en lado de componentes).
- Ancho mínimo de pistas: .007 (7 mil).
- Cobre de 1 onza.
- Espesor de dieléctrico (FR4):
  - entre capas 1-2 y 3-4: .012”
  - entre capas 2-3: .028”
- Constantes dieléctricas del FR4:
  - entre capas 1-2 y 3-4: 4.6

- entre capas 2-3: 4.7

- Espesor de cobre:
  - capas 1 y 4: .002”-.0025”
  - capas 2 y 3: .0012”-.0013”

6.3.1. Posicionamiento de componentes

Los componentes elegidos fueron en su gran mayoría de montaje superficial y para cada valor requerido se buscó el encapsulado más pequeño posible. De esta forma la mayoría de los capacitores son 0402 excepto por los capacitores de 10uF que son 0603, ya que no se encontraron encapsulados más pequeños para dicha capacitancia. Las resistencias y los inductores son todos 0603. Vale aclarar que todos los integrados y uno de los osciladores (el de 24MHz) también son de montaje superficial (ver figura 6.6). Para el posicionamiento de los componentes en la placa, primeramente se fueron ubicando los distintos bloques funcionales identificados (ver 6.2) junto con sus respectivos componentes. Estos bloques son cuatro: bloque del transceiver, bloque del microcontrolador, bloque de alimentación, y bloque adaptador de niveles RS232. Una vez distribuidos cada uno de los bloques en el “board”, se reubicaron algunos componentes para lograr un mejor aprovechamiento de la placa. Para algunos sectores críticos del circuito, ciertas consideraciones respecto al ruteo de pistas y ubicación de componentes fueron tomadas en cuenta. En sectores como la salida RF o la alimentación de los integrados es necesario reducir las distancias de las pistas lo más posible para reducir al máximo los niveles de ruido.

Por lo tanto:

- Los capacitores conectados directamente a las alimentaciones de todos los integrados fueron ubicados los más cerca posible de los pines de entrada de los mismos, para reducir las distancias de las pistas que entran en los integrados y por ende reducir los niveles de ruido.
- En la Salida RF (pin RFIO) se acortó el largo de la pista que une el ZMD44102 con el conector SMA.
- Las pistas de entrada a los pines de la interfaz JTAG se acortaron en largo y fueron simplificadas en forma (evitando curvas innecesarias y esquinas abruptas).
- Los componentes externos del PLL se posicionaron cerca del transceiver y fue rodeado por relleno de tierra para aislar las señales del oscilador de 24 MHz y de la alimentación del filtro VCO.

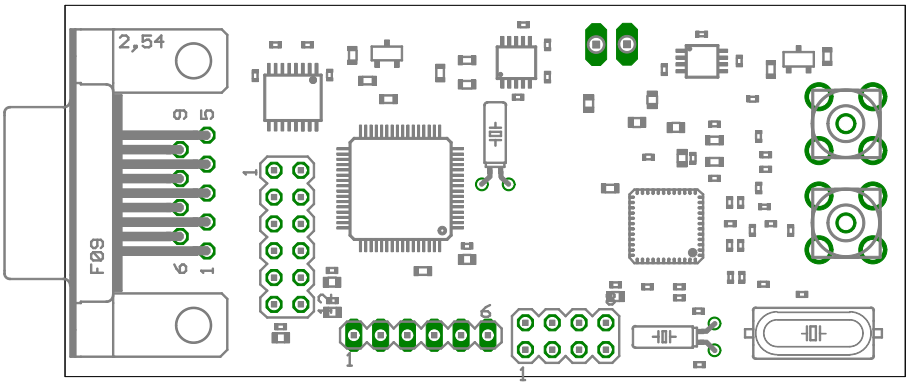

Figura 6.6: Posición de los componentes en la placa

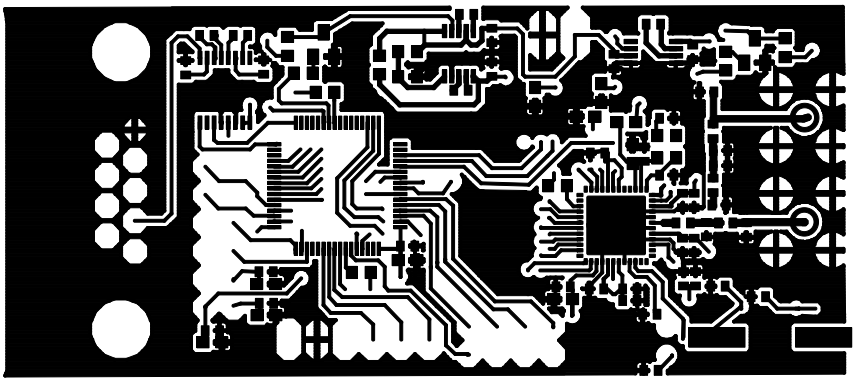

Figura 6.7: Capa superior del PCB

6.3.2. Ruteo

El ruteo se realizó en cuatro capas: se utilizaron dos capas para las señales, una capa dedicada a al plano de tierra y otra dedicada a las alimentaciones. Para el ruteo de señal se utilizaron las capas superior (figura 6.7) e inferior (ver figura B.7 del apéndice B) del PCB. De esta manera se logra tener fácil acceso a las pistas de señal, brindando mejores posibilidades de prueba y también posibilitando la corrección de errores. El plano de tierra es una capa de cobre que se encuentra conectada al voltaje de referencia, como se muestra en la figura B.8 del apéndice B. El plano de alimentación es una capa como la de tierra pero separado en dos sectores; uno referenciado al 3.3V y otro sector referenciado a 2.5V (figura B.9 del apéndice B). La alimentación se podría haber hecho en estrella pero al contar con un plano dedicado para la alimentación no fue necesario hacerlo.

6.4. Montaje del sistema

En un inicio se pensó en hacer el montaje del sistema soldando los componentes en horno, pero para el soldado a horno es altamente recomendable contar con plantilla o “stencil”. El “stencil” es una placa

de acero inoxidable que se coloca sobre la placa previamente a introducirla en el horno. El "stencil" trae perforaciones en los lugares exactos donde es necesario colocar el estaño que fija los componentes a las pistas del PCB. Al pintar el "stencil" con pasta de soldar, se retira el "stencil" y se colocan los componentes en su posición para luego llevar la placa a horno y fundir la pasta de cada componente. Al no contar con "stencil", se decidió probar soldar cada componente a mano. Se utilizaron distintas técnicas de soldado manual y se fueron mejorando estas técnicas con la práctica. A continuación se describen dos técnicas finales empleadas, con las cuales se obtuvieron los mejores resultados. En la figura 6.8 se puede ver el prototipo final luego del montaje.

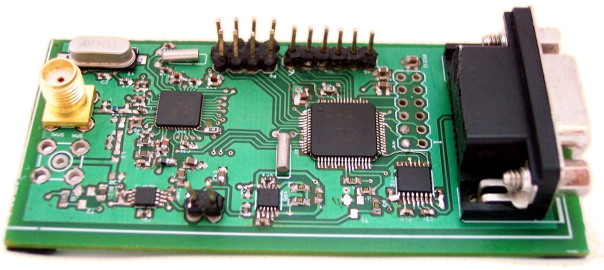

Figura 6.8: Prototipo $\mathbf{I_{IE}M_{OTE}}$

6.4.1. Técnica de soldado a mano

Los siguientes componentes de montaje superficial fueron soldados con la técnica de soldado a mano:

- Condensadores.
- Inductores.
- Resistencias.
- Cristal de 32.768 kHz.
- Reguladores de voltaje TPS.
- Transceiver MAX3221.

Los materiales utilizados fueron los siguientes:

- Soldador de punta mediana (1 o 2 mm).
- Estaño fino (0.5 mm).
- Flux líquido.

- Pincel.
- Lupa.
- Pinza chata.

Procedimiento:

- Una vez que se tiene el soldador caliente, aplicar el flux con un pincel sobre los pads a los que se quiere soldar el componente.
- Estañar los pads asegurándose que el estaño quede extendido de la manera más uniforme posible sobre los pads.
- Colocar el componente sobre los pads y calentar con el soldador uno de los pads mientras se sujeta el componente desde el otro extremo con la pinza chata.
- Una vez que el componente queda fijo a uno de los pads, se vuelve a tomar el estaño y con el soldador se van calentando y estañando el resto de los pads hasta que el componente queda fijo en su totalidad.

6.4.2. Técnica de soldado a mano con pistola de aire caliente

Para soldar el transceiver ZMD44102 y el microcontrolador MSP430F149 se utilizó otra técnica distinta a la mencionada anteriormente, en la cual se incorpora una pistola de aire caliente en el proceso de montaje. Debido a que con esta técnica el PCB es sometido al calor directo de la pistola de aire caliente, siempre es conveniente entonces empezar por soldar estos circuitos más complicados primero, y luego soldar el resto de los componentes con la técnica de soldado a mano.

Los materiales utilizados en ésta técnica fueron los siguientes:

- Soldador de punta mediana (1 o 2 mm).
- Estaño fino (0.5 mm).
- Flux líquido.
- Pincel.
- Lupa.
- Pinza chata.
- Pistola de aire caliente.

- Papel de aluminio.

Procedimiento:

- Una vez que se tiene el soldador caliente, aplicar el flux con un pincel sobre los pads a los que se quiere soldar el componente.
- Estañar los pads asegurándose que el estaño quede extendido de la manera más uniforme posible sobre los pads[1].
- Cubrir con papel de aluminio el resto del PCB dejando visible solamente la zona donde se va a colocar el componente (zona de montaje). Esto se debe hacer para proteger el resto del PCB del calor al que se va a someter la zona de montaje.
- Encender la pistola de aire caliente y calentar la zona de montaje hasta que el estaño de los pads se derrita.
- Retirar un poco la pistola de aire caliente y colocar rápidamente con la pinza chata el componente a montar sobre los pads.
- Acercar nuevamente la pistola de aire caliente para mantener el estaño derretido sobre los pads y con el uso de la lupa alinear los pines del componentes sobre los pads (puede ser conveniente protegerse la mano del calor con un guante de cuero).
- Luego de que el componente se encuentre perfectamente alineado en su lugar, retirar del todo la pistola de aire caliente y con la pinza chata presionar gentilmente el componente sobre el PCB mientras se seca el estaño.

[1] En el caso que se tenga un numero grande de pads para estañar, es conveniente no aplicar el flux en todos los pads de una sola vez ya que éste se evapora rápidamente. En éstos casos, lo mejor es ir aplicando el flux y luego estañando los pads en varias etapas.

Capítulo 7

Software de prueba

7.1. Introducción

Previo a la construcción del primer $\mathbf{I_{IE}M_{OTE}}$, mientras los tiempos de fabricación e importación del PCB corrían, se diseño y construyo un programador JTAG paralelo (ver el apéndice C) basado en el diseño de Olimex[34] y se desarrollaron diversas rutinas en C que permitieron probar el hardware en una forma incremental e independiente.

Como se explicó en el capítulo 4, se utilizaron las herramientas de desarrollo de GNU. Particularmente, todo el código C presentado en este documento fue compilado con el compilador CompiladorGCC[1] (Gnu C Compiler portado para el procesador MSP430).

En las rutinas desarrolladas, se buscó separar el código de acceso al hardware en librerías que luego pudieran ser reutilizadas (UART, transciever), construyendo así una pequeña capa de abstracción del acceso al hardware (HAL).

En este capítulo se presentan las rutinas desde un punto de vista funcional y las partes más importantes del código de las mismas. El código completo de todas ellas se encuentra en el apéndice E.

7.2. Software de pruebas de I/O | leds

La primera rutina desarrollada es minimalista. Su único objetivo es alternar en forma secuencial el estado de todos los pines de los puertos 5 y 6 del microcontrolador. Se elaboró una pequeña placa auxiliar, que conecta en los pines de estos puertos LEDs, como se muestra en la figura 7.1.

Esta rutina se utilizó para tener la primera “señal de vida” del microcontrolador, con ella nos aseguramos que el microcontrolador funciona correctamente y también sus puertos de entrada y salida. Esto es

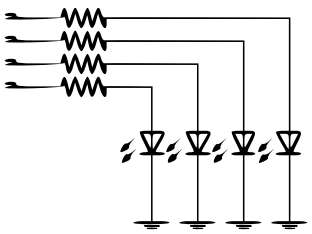

Figura 7.1: Placa auxiliar de leds

muy importante pues nos permite confiar en los puertos como señales de depuración para las rutinas más complejas.

Código de prueba de puertos:

```
for(;;) {                          /* bucle infinito */
  volatile int a,b,c;
  for(a=0;a<10;a++){
    for(b=0;b<20;b++){
      for(c=0;c<2000;c++){
        P6OUT ^= 0x44;             /* alterno estado de los bits 2 y 6 */
        P5OUT ^= 0x44;
      };
      P6OUT ^= 0x22;               /* alterno estado de los bits 1 y 5 */
      P5OUT ^= 0x22;
    };
    P6OUT ^= 0x11;                 /* alterno estado de los bits 0 y 4 */
    P5OUT ^= 0x11;
  };

}
```

7.3. Software de prueba de uart

Se desarrolló una pequeña biblioteca de acceso al puerto serie sin utilizar buffer y una pequeña aplicación de prueba. La rutina *uarttest* imprime un texto en la pantalla del PC solicitando que se ingrese un caracter (“Favor toca una tecla:”), y queda a la espera del mismo. Cuando recibe un caracter, hace alternar el estado

de “los leds” (cambiando el estado de las señales del puerto 2 y 6), envía un mensaje reportando el caracter recibido y solicitando que se ingrese otro. El PC se encuentra conectado a la UART a través de su puerto COM.

A continuación se puede ver un ejemplo de una sección de esta pequeña aplicación:

```
Favor toca una tecla:
Algo me dice que tocaste la "J", toca otra:
Algo me dice que tocaste la "F", toca otra:
Algo me dice que tocaste la "R", toca otra:
```

Esta rutina permitió validar el funcionamiento del hardware y depurar la biblioteca desarrollada. La interfaz de la biblioteca (*uart.h*) es presentada en la sección 7.5.

7.4. Software de prueba del transceiver

ZMD distribuye un código de referencia (ZMD44102SK BCS v1.0) junto a toda la documentación de su diseño tipo. El mismo está desarrollado para la placa ZMD44102StarterKit, una plataforma de prueba del transceiver fabricada por ZMD, basada en el el procesador C8051F12x. El código esta desarrollado en C, en particular para el compilador Keil C v7.

Nosotros contamos con un Starter Kit, pero no con el kit de desarrollo para el microprocesador C8051F12x, por lo cual no fue posible utilizar este software en la plataforma del Starter Kit. Igualmente, el Starter Kit tiene una aplicación pre-cargada que permite leer y escribir los registros del ZMD, utilizando una interfaz gráfica provista por el kit. La comunicación entre el GUI y el Starter Kit es vía USB (USBcomm), y utiliza su propio protocolo no documentado.

El software de referencia BCS, permite un acceso directo al transceiver, permitiendo leer y escribir directamente los registros del mismo desde un PC a través de una consola serie (ej: Realterm[35]). También tiene una serie de funciones para configurar el transceiver en algún modo pre-establecido.

Aprovechando que el fuente C de esta aplicación está disponible junto a la documentación del Starter Kit de ZMD, se centraron los esfuerzos en migrarlo al **IEMOTE** y al compilador mspgcc. A su vez este software se expandió y mejoró para que sea más funcional al proyecto.

A continuación se listan las principales tareas involucradas en la migración del código:

Macros Definición de macros para compatibilidad con el compilador Keil.

Puertos Se redefinieron todos los macros de acceso a puertos para hacerlos compatibles a nuestra plataforma.

Tiempos La aplicación utiliza funciones de espera, calibradas para esperar N μs o N ms, dichas funciones fueron recalibradas para la frecuencia de trabajo del $\mathbf{I_{IE}M_{OTE}}$.

UART Se implementó la biblioteca *uart.h* para el $\mathbf{I_{IE}M_{OTE}}$ y se adaptó el código para que la utilice.

Comunicación El Starter Kit utiliza la comunicación **serie** entre el transceiver y el microcontrolador, el $\mathbf{I_{IE}M_{OTE}}$ se comunica en forma **paralela**, por lo cual fue necesario desarrollar desde cero la biblioteca de acceso al transceiver.

SysInit La inicialización del sistema fue reescrita, para inicializar correctamente nuestro microcontrolador.

atoh Función de conversión de hexadecimal escapado a byte. Nuestro compilador no cuenta con esta facilidad, por lo cual la implementamos.

Funciones Desarrollamos nuevas funciones para facilitar las pruebas de la plataforma.

A continuación se listan las funciones actualmente accesibles desde consola:

```
 --- BCS | Version IEEMote ---
(0)  check LEDs                (1)  ZMD44102 reset
(2)  GPD enable                (3)  GPD disable
(4)  write to ZMD44102 reg.    (5)  read from ZMD44102 reg.
(s)  set phyPIB values         (g)  get phyPIB values
(I)  Init ZMD44102 (MTP)       (i)  get stored addr entr.
(S)  FrameSniffer              (D)  Go to Sleep
(R)  Hello World RX (SK)       (H)  Hello World TX (SK)
(t)  transmit unslotted mode   (r)  receive unslotted mode
(o)  UnslottDataPoll_C         (p)  UnslottDataPoll_D
(c)  passive scan              (e)  energy detection scan
(v)  active scan
(b)  auto beacon transmit      (a)  beacon tracking
(m)  BcTrafficCoord_d2c (C)    (n)  BcTrafficDev_d2c (D)
(G)  BcCoord_d2c_gts (C)       (T)  BcDev_d2c_Gts (D)
(X)  RF Tx test pn-seq         (x)  RF Tx test cont. carr.
(Y)  Rx test alcance           (U)  Tx test alcance
```

La funciones "`write to ZMD44102`", y "`read from ZMD44102`", que permiten la escritura y lectura directa de los registros del ZMD44102, son fundamentales en la prueba del transceiver. "`FrameSniffer`" permite visualizar todos las tramas que se reciben en un canal y están bien delimitadas. Las rutinas "`Hello World ...`" permiten probar la comunicación entre dos $\mathbf{I_{IE}M_{OTE}}$, o entre un $\mathbf{I_{IE}M_{OTE}}$ y la placa Starter Kit. Por último, las funciones "`RF Tx test...`" configuran el ZMD44102 para que genere un espectro conocido a su salida, esto es muy útil pues permite ver cuán bien se comportan los filtros, y si hay algún corrimiento

de frecuencia de una forma fácil. Como se verá más adelante, estas rutinas fueron fundamentales para las pruebas iniciales del $\mathbf{I_{IE}M_{OTE}}$.

Como se visualiza en la lista anterior, son muchas las funciones, algunas de estas funciones sólo se portaron a la plataforma, otras fueron portadas y mejoradas, y otras desarrolladas exclusivamente para las pruebas del $\mathbf{I_{IE}M_{OTE}}$.

En las partes que siguen, se describirá la arquitectura general del software, y se presentarán algunas de las funciones implementadas.

7.4.1. Arquitectura del BCS

La figura 7.2 muestra la estructura general de la aplicación.

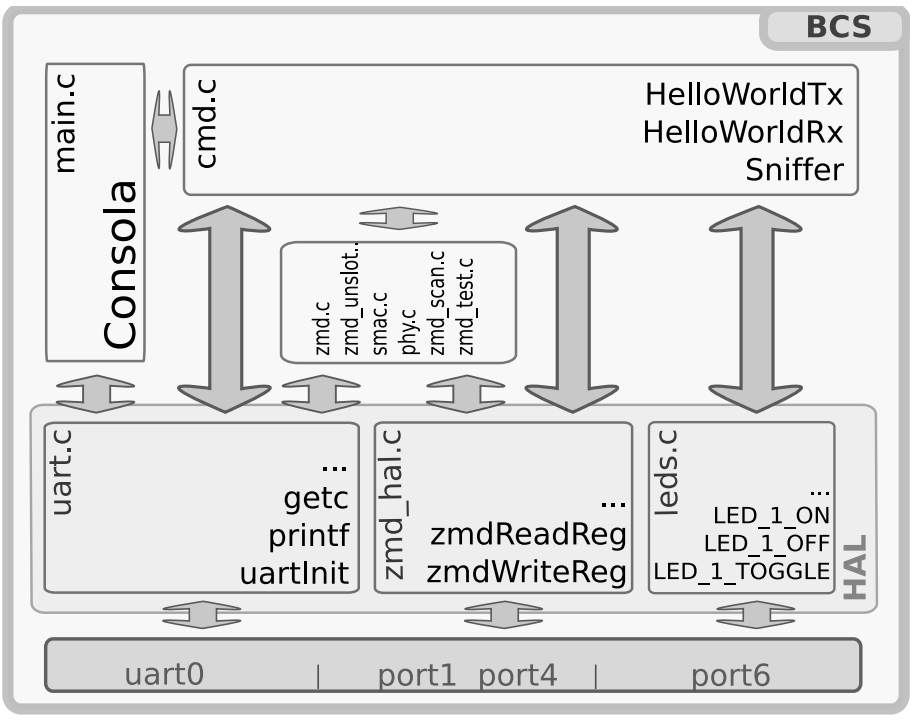

Figura 7.2: Arquitectura del software BCS

Es importante visualizar la separación en capas de la misma, se busca crear una pequeña capa de abstracción del acceso al hardware (HAL), de forma de facilitar el desarrollo de otras aplicaciones.
Todas las funciones que implementan los comandos están en *cmd.h*, y éstas utilizan recursos auxiliares de las siguientes bibliotecas:

leds.h funciones de acceso al puerto 2.

phy.h funciones de capa física del ZMD44102.

smac.h estructuras de capa MAC de la norma IEEE 802.15.4.

uart.h funciones de acceso y inicialización del UART 0.

zmd_beacon_mode.h funciones para configurar el ZMD44102 en los distintos modos de "beacon".

zmd.h funciones generales de ZMD44102.

zmd_hal.h funciones de acceso y inicialización del transceiver.

zmd_scan.h funciones de búsqueda.

zmd_unslotted.h Transmisión de datos sin "slots".

En la sección 7.5 se presentan las principales bibliotecas de la capa HAL.

7.4.2. Rutina Hola Mundo

Como se vió, el Starter Kit de ZMD viene pre-cargado con un software muy básico. Éste permite la ejecución de "scripts" (batch de escrituras a registros del ZMD44102), los cuales se cargan utilizando una interfaz gráfica propia del Kit. Buscando aprovechar esto, desarrollamos las rutinas "pares" a dos de los "scripts" que soporta el Starter Kit, de forma de poder probar el **$I_{IE}M_{OTE}$** contra una plataforma que ya está probada.

Las rutinas "pares" que implementamos permiten poner un transceiver en modo recepción, y configurar otro para transmitir una trama básica con requerimiento de acknowledge. Luego de ejecutado el proceso, leyendo los registros de estado del transceiver podemos saber si se transmitió correctamente, si se recibió correctamente y si el acknowledge adecuado fue recibido por el transmisor. Estos sencillos "script" nos permitieron validar el funcionamiento del **$I_{IE}M_{OTE}$** , como se verá en el capítulo 8.
A continuación se presenta una versión comentada de la rutina TxHello:

```
void cmdTxHello (void)
{
  zmdReset(HOLD_TIME_1);
  zmdWriteReg(MTP_CTRL, 0x09);              /* comienza programación: ver zmd datasheet
p.118 */
  while (zmdReadReg(MTP_CTRL) != 0x0C);
  zmdWriteReg(0x37, 0x00);

  zmdWriteReg(0x80,0x48); /* escribe 0x48 (ascii H) en la FIFO de transmisión */
```

```
  zmdWriteReg(0x80,0x65); /* escribe 0x65 (ascii e) en la FIFO de transmisión */
  zmdWriteReg(0x80,0x6C); /* escribe 0x6C (ascii l)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x6C); /* escribe 0x6C (ascii l)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x6F); /* escribe 0x6F (ascii o)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x20); /* escribe 0x20 (ascii SPC)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x77); /* escribe 0x77 (ascii w)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x6F); /* escribe 0x6F (ascii o)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x72); /* escribe 0x72 (ascii r)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x6C); /* escribe 0x6C (ascii l)en la FIFO de transmisión */
  zmdWriteReg(0x80,0x64); /* escribe 0x64 (ascii d)en la FIFO de transmisión */
  zmdWriteReg(0x60, 0x0B); /* escribe largo de trama = 11 */
  zmdWriteReg(0x62, 0xC0); /* escribe 0xC0 en el byte mas alto de la trama de control (bits[15:8])
*/
  zmdWriteReg(0x61, 0x21); /* escribe 0x21 en el byte mas bajo de la trama de control (bits[7:0])
*/
  zmdWriteReg(0x00, 0x01); /* define transmisión en el canal 1 (906 MHZ) */
  zmdWriteReg(0x71, 0x12); /* le asigna 0x12 al campo de identificación del encabezado de
fuente MAC para una trama de transmisión */
  zmdWriteReg(0x70, 0x34); /* le asigna 0x34 al campo de identificación del encabezado de
fuente MAC para una trama de transmisión */
  zmdWriteReg(0xA0, 0x03); /* se inicia modo transmisión*/
}
```

A continuación se presenta una versión comentada de la rutina RxHello:

```
 void cmdRxHello (void)
{
  zmdReset(HOLD_TIME_1);                    /* Reset del ZMD44102 */
  zmdWriteReg(MTP_CTRL, 0x09);              /* se inicia programación */
  while (zmdReadReg(MTP_CTRL) != 0x0C);     /* zmd datasheet p.118 */
```

```
    zmdWriteReg(0x37, 0x00);                    /* se termina programación */

    zmdWriteReg(0x00, 0x01);                    /* seteamos canal 01 (906 Mhz) */
    zmdWriteReg(0x08, 0x62);
    zmdWriteReg(0xB6, 0x01);
    zmdWriteReg(0xA0, 0x04);                    /* se inicia modo recepcion */
}
```

Una vez ejecutada la rutina rxHello podemos verificar el estado del transceiver leyendo el registro macRxStatus (0xA4), como se muestra en la tabla 7.1. Si todo funciona correctamente, es de esperar que al iniciar la rutina el estado sea Acquire (0x01). Luego de ser ejecutado TxHello en la placa transmisora, el estado de la placa receptora tendría que ser DataAck (0x80), indicando que se recibió una trama con pedido de acknowledge.

Visto desde la placa transmisora, al ejecutar txHello, si el paquete fue transmitido y se recibió el acknowledge correspondiente, el valor del registro macRxStatus (0xA3) es Success (0x80), como se muestra en la tabla 7.2.

Tabla 7.1: macRxStatus — 0xA4

Valor	**Estado**	**Descripción**
0x00	Off	Rx off
0x01	Acquire	Acquisition is ongoing
0x02	RxActive	Frame reception is ongoing
0x10	Ack*	Acknowledge frame received
0x14	AckTimeOut*	Wait for acknowledge timed out
0x18	AckFramePend*	Ack received with frame pending bit set
0x20	PollNoData*	No frame received within T_MaxFrameResponse after an acknowledge with the frame pending bit set
0x24	PollCAPend*	No pending data received within the current CAP. WaitForFrame process will be summed in the next CAP.
0x80	Data*	Frame without the acknowledge requested bit set received
0xC0	DataAck*	Frame with the acknowledge requested bit set received

Tabla 7.2: macTxStatus — 0xA3

Valor	**Estado**	**Descripción**
0x00	Off	Tx off (Reset default)
0x01	GTS	GTS transmission
0x02	SltCSMA	Slotted CSMA transmission
0x03	SltDir	Slotted direct transmission
0x04	UnSltCSMA	Unslotted transmission
0x05	UnSltDir	Unslotted direct transmission
0x06	Ack	Automatic acknowledge transmission
0x07	Fast	Transmission quickly following an Ack
0x08	Success*	Transmission completed successfully
0x10	CAfail_Chbusy*	Transmission failed due to channel access failure because of a busy channel
0x20	CAfail_CAPfail*	Transmission failed due to channel access failure because of CAP end
0x40	GTSfail*	GTS transmission failed due to unavailable slot

7.5. Bibliotecas de abstracción de hardware — HAL

7.5.1. Biblioteca uart.h

Para abstraer el acceso al hardware, y buscando la mayor flexibilidad posible, se desarrolló una biblioteca para el puerto UART0 del msp430[1].

La implementación prevé el funcionamiento del puerto serie configurado como "8N1 a 9600 baud" (palabra de 8 bit, 1 bit de parada y sin paridad), la velocidad esta fijada por el cristal de 32.768kHz por lo cual es independiente de la configuración de la frecuencia del sistema (MCLK). Igualmente se pueden configurar distintas velocidades en tiempo de compilación.

A continuación se presenta *uart.h*, con comentarios resumidos (versión completa en apéndice E):

```
#ifndef UART_INCL
#define UART_INCL

#include <io.h>
#define gets uartGets
#define getchar uartRx
#define puts uartTxString
#define putc uartTx
#define getc uartRx
#define printf(format, ...) uprintf(pchar1, format, ## __VA_ARGS__)

int pchar1(int);           /* envia caracter - compatible con uprintf */
void uartInit(void);       /* inicia Uart0 en 9600 8N1 */
void uartTxString(char *);                          /* transmite un string */
unsigned int uartGets(char *s, unsigned int n);  /* recibe linea o n bytes */
unsigned char atoh(char * p_escape_hex); /* convierte hex escapado en byte */

static inline void uartTx(char c)      /* copia 1 caracter a buffer de salida */
{
  while(!(U0IFG & UTXIFG0));
```

[1] La biblioteca de acceso al puerto serie se basa en un código de ejemplo, elaborado por Peter Baxendale, el mismo fue corregido y adaptado a nuestras necesidades.

```
  U0TXBUF = c;
}
static inline unsigned int uartRx(void) /* lee caracter de buffer de entrada */
{
  while(!(U0IFG & URXIFG0));
  return U0RXBUF;
}

static inline unsigned int inkey(void)  /* hay algo en el buffer de entrada? */
{
  return (U0IFG & URXIFG0);
}

#endif
/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

La implementación tiene la ventaja de brindar las facilidades de las funciones *printf* para dar formato al texto de salida, logrando que sea muy sencillo la utilización del puerto serie.

7.5.2. ZMD Hardware Abstraction Layer

Se desarrolló una biblioteca de abstracción de hardware para el transceiver (el software de ZMD utilizaba esta biblioteca para acceder al hardware). El **IEMOTE** utiliza comunicación paralela entre el transceiver y el microcontrolador, por lo cual fue necesario el desarrollo de las rutinas de acceso manteniendo la interfaz definida en el BCS.

En esta biblioteca se encuentran las funciones básicas de comunicación con el ZMD, la implementación sigue los requerimientos de tiempo expresados por el diagrama de tiempos de la figura 3.8 en la página 36.

A continuación se presenta la interfaz (*zmd_hal.h*), con comentarios resumidos (versión completa en apéndice E):

```
#ifndef _ZMD_HAL_H_/* Prevengo doble inclusión */
#define _ZMD_HAL_H_

#include ''globals.h''
```

```
#define HOLD_TIME_1  1  // 500 us
#define HOLD_TIME_2  2  // 3 ms
#define HOLD_TIME_3  3  // 730 ms

#define zmdWriteTxFifo(length, data_buffer) zmdWriteBlock(TX_FIFO, length, data_buffer)

void zmdGpd (BOOL value);                                  /* Global Power Down */
void zmdReset (UINT8 hold_time);                           /* Reset ZMD */
void zmdInitHal (void);                                    /* Inicializa puertos */
void zmdWriteReg (UINT8 reg_addr, UINT8 Dataout);          /* Escribe a un registro */
UINT8 zmdReadReg (UINT8 reg_addr);                         /* Lee un registro */
void zmdWriteBlock (UINT8 reg_addr, UINT8 length, UBYTE* pData);
                    /* Copia un bloque de datos desde la memoria a un registro */

void zmdReadBlock (UINT8 reg_addr, UINT8 length, UBYTE* pData);
                    /* Le un registro N veces y lo copia a un bloque de memoria */

void wait_ms (UINT16 ms);                                  /* Espera milisegundos */
void wait_us (UINT16 us);                                  /* Espera microsegundos */

#endif
/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

Capítulo 8

Pruebas y ajustes

8.1. Antenas

8.1.1. Primeras pruebas de caracterización de las antenas

Previamente a probar rutinas de recepción y transmisión con cada una de las antenas diseñadas, se utilizó una antena patrón como generadora de ruido y se fueron probando cada una de las antenas diseñadas como receptoras. La antena patrón utilizada fue una antena bicónica la cual fue conectada a la salida del generador de ondas del analizador de espectro (tracking generator output) generando una salida de referencia de 92.0 dBμV. Una vez que la antena patrón estuviera emitiendo señal en "todo" el espectro se fueron colocando en la entrada del analizador cada una de las antenas diseñadas y se registraron los valores de potencia recibidos. La antena patrón se colocó, mediante un cable extenso, a 2 metros de distancia de el punto de medición en donde se colocaron cada una de las antenas a probar. Todas las antenas probadas fueron colocadas en posición horizontal y conectadas a la entrada del analizador de espectro en posición horizontal.

La tabla 8.1 muestra los resultados obtenidos en cada medición. Éstos solamente tienen valor relativo entre ellos y sirven como comparación entre las distintas antenas.

Como se observa claramente, las antenas que entregaron la mayor cantidad de potencia fueron el monopolo

Tabla 8.1: Potencia de recepción en dBμV

Antena	**902 MHz**	**915 MHz**	**928 MHz**
Monopolo doblado	77.7	71.3	63.5
Antena de loop (red de adaptación #1)	59.0	53.0	61.8
Antena de loop (red de adaptación #2)	65.0	59.1	54.5
Antena de loop (red de adaptación #3)	56.3	57.8	59.0
Monopolo (Starter Kit)	72.0	66.5	66.5

y el monopolo doblado (por más información sobre el diseño de éstas antenas referirse al Capítulo 5).

8.2. Ajustes del programador JTAG

Se desarrolló un programador JTAG para el microcontrolador MSP430, orientado a cumplir los requerimientos del software msp430-jtag del kit de desarrollo de GNU (mspgcc). Este programador de 4 hilos, permite programar la EEPROM del microcontrolador, así como depurar en línea la ejecución del software en él haciendo uso de la herramienta gdb. En el apéndice C se presenta el esquemático y layout del programador desarrollado.

8.2.1. Primeras programaciones del $\mathbf{I_{IE}M_{OTE}}$

Si bien finalmente se logro programar exitosamente el microcontrolador, las primeras pruebas fueron dominadas por *"lo esotérico*[1] *"*. Sigue una pequeña lista de eventos, ordenados por su ocurrencia en el tiempo:

- Se programa la primera rutina de prueba (testleds.c) con éxito.
- Se programa la aplicación BCS en forma exitosa.
- El microcontrolador deja de responder a cualquier "excitación externa".
- Se fabrica otro $\mathbf{I_{IE}M_{OTE}}$ y se programa la aplicación BCS en forma exitosa.
- El microcontrolador de la segunda placa deja de responder a eventos externos.
- Se analiza el programador, se teme por un problema de alimentación, y se desconfía del buen funcionamiento del buffer del programador, se realizan cambios en el hardware sin obtener éxito alguno.
- Se identifica en foros de internet personas con similares problemas, pero ninguna respuesta.
- Por accidente se intenta programar sin alimentar la placa, falla la programación pero la EEPROM del microcontrolador es borrada.
- Se conecta la alimentación y se programa el chip, previamente borrado, de forma exitosa.
- Se intenta repetidas veces borrar la EEPROM sin alimentación, y algunas veces funciona (1 de cada 30).
- Luego de muchas pruebas y lectura de foros en Internet, se descubrió que si se descargaban los condensadores de alimentación del $\mathbf{I_{IE}M_{OTE}}$, previo a conectar el programador para borrar sin alimentación, *casi siempre* funcionaba.

[1] Esotérico: Dicho de una cosa: Que es impenetrable o de difícil acceso para la mente.[36]

- Sigue la secuencia de grabación/borrado funcionando aleatoriamente y haciendo perder mucho tiempo; en el transcurso de esto, el software crecía y se depuraba.
- A partir de un momento, fue imposible re-borrar la EEPROM.

Luego de muchas horas de pruebas con otros programadores y muchas horas en internet intentando encontrar el causal de tal *despropósito*, se logra conseguir un osciloscopio digital para intentar ver las señales del *misterioso* JTAG.

8.2.2. Ruido

Una vez todo conectado, se "capturó" señal por señal del JTAG (TDI, TMS, TCK y TDO) en el osciloscopio, durante una secuencia de programación. En una primera instancia se percibieron unos picos muy raros en la señal de TDI, lo cual no parecían propios de la comunicación. Se indagaron muchas referencias al respecto, en particular, un articulo muy interesante titulado "JTAG Cable Considerations[37]", hizo desaparecer lo *esotérico*. El ruido que se notaba en TDI tenía correlación con la señal TCK, y la señal TDI generaba también ruido en TCK. JTAG sincroniza la comunicación con la señal TCK, el ruido generado en TCK puede confundir a el *target* (microcontrolador), y llevarlo a saltar al próximo evento en un momento incorrecto. Una vez identificado el problema, se buscó eliminar las causas, se utilizó cable blindado para TCK, y se puso un cable más corto. Luego de todo esto, se constató en el osciloscopio que el ruido había desaparecido.

8.2.3. Buffer

Se solucionaron los problemas de ruido, pero el programador seguía sin responder correctamente.

Como se muestra en el apéndice C, el programador es básicamente un buffer, utilizado para reconstruir la señal sin ruidos, y una serie de resistencias que permiten ajustar los niveles de tensión.

Analizando las capturas de las señales, se identificó que la señal TMS no tenía los niveles de voltaje debidos. Luego de analizar el problema, se concluyó que el buffer del programador tenía una salida "rota", probablemente víctima de las numerosas pruebas. Se intentó comprar un remplazo en plaza (Uruguay), y no fue posible, por lo cual simplemente se remplazo el buffer por una resistencia para esta señal y la programación comenzó a funcionar en forma correcta.

8.2.4. Conclusión

Finalmente se lograron solucionar todos los inconvenientes del programador, pero la "experiencia" con JTAG no pasó desapercibida. Lejos de ser algo sencillo, requirió de una gran inversión de tiempos. Igualmente,

en los foros de internet sobre el MSP430, se puede ver que es un problema común, el que mucha gente enfrenta y suele no poder resolver, esperamos que nuestra experiencia sea útil para otros. Con tal motivo enviamos una nota resumiendo esto a los diversos foros, la cual también publicamos en http://compendio.com/iiemote/jtag.html. Como saldo:

- 3 programadores de JTAG destruidos.
- 140 horas hombre invertidas/destruidas.
- 314 historias dignas de ser contadas.

8.3. $\mathbf{I_{IE}M_{OTE}}$

Una vez soldados todos los componentes del $\mathbf{I_{IE}M_{OTE}}$, era necesario comenzar a probar los distintos sub-sistemas del mismo. Para esto se construyeron las rutinas de software que se presentaron en el capítulo 7. En esta sección se registran todos los pasos que se siguieron en las pruebas de los distintos sub-sistemas y se comentan los ajustes que fueron necesarios en el transcurso de este proceso.

8.3.1. Encendido del prototipo

Inconvenientes encontrados en el PCB

Recién terminado de soldar, se probaron todas las alimentaciones en busca de cortocircuitos, u otros errores. En estas pruebas se encontró un error en la alimentación de $2.5V$. En el ruteo, una pista se corrió bajo uno de los condensadores de alimentación del ZMD44102 (condensador C20), creando un cortocircuito entre $2.5V$ y tierra. Probablemente la pista se movió por error, durante el proceso de generación de los gerber, puesto que antes de esto se habían corrido todos los test de DRC[2] y ERC[3], y todo estaba correcto. Para solucionar este problema, se corto la pista con una trincheta.

Luego de solucionado el primer traspié del encendido, se alimento la placa y se verificó la correcta alimentación de todos los integrados. En este punto se detectó un problema en el plano de tierra superior; por error se cambió el parámetro de "isolation" de este plano en la generación del gerber (o sea, posterior los test DRC/ERC). Esto provocó que algunas tierras quedaran desconectadas, para lo cual se pusieron pequeños hilos de cobre conectando las tierras y solucionando el problema. Los componentes y pines que quedaron separados del plano de tierra superior fueron:

- Pines 62 y 63 del MSP430F149.

[2]DRC: Design Rule Check
[3]ERC: Electrical Rule Check

- Condensadores C16 (0603) y C15 (0402).
- Pin 1 del MAX3221.

A continuación se mencionan los restantes imprevistos encontrados en el prototipo:

- Se conectó la resistencia R1 en el pin SSK en vez del pin SSN del ZMD44102, para lo cual se debió cortar la pista de entrada a SSK y mediante un alambre de cobre soldar R1 a el pin SSN.
- El footprint del regulador TPS60212 diseñado en Eagle tiene una separación de pines incorrecta, con lo cual se tuvieron que separar, utilizando la pinza chata, las patas del regulador previamente a su soldado.
- Los voltajes de referencia para el ADC (pines VREF+, VeREF+, VeREF- del MSP430F149) fueron ruteados junto a los pines de salida del puerto 5, es más practico que se encuentren junto a los pines de salida del puerto 6.
- Los pines Rx y Tx del conector hembra RS232 estaban cambiados, con lo cual hubo que cruzar los pines del conector macho utilizado.

Ya solucionados todos los inconvenientes de alimentación, se prosiguió cargando las distintas aplicaciones para probar los distintos sub-sistemas. Si bien estos inconvenientes retrasaron la etapa de soldado, es importante mencionar que ninguno de ellos tuvieron repercusiones graves en el ensamblado general o el funcionamiento del prototipo de la placa.

Prueba de UART

Habiendo probado la rutina *testleds.c* previamente, y habiendo constatado que los puertos P5 y P6 del microcontrolador funcionaba correctamente, se cargó la rutina *uartest.c*. La primera versión de esta rutina tenía unos errores en la inicialización del puerto, por lo cual fue necesario depurarla utilizando los LEDs. Si bien la depuración con LEDs es tediosa, los errores eran menores y fueron solucionados rápidamente.

En esta etapa no se identificó ningún problema de diseño, si hubo algunos inconvenientes con el soldado, en dos de los tres $\mathbf{I_{IE}M_{OTE}}$ armados fue necesario re-soldar las patitas de rx/tx entre el microcontrolador y el MAX3221.

Encendido del transceiver

Para esta etapas se utilizó la aplicación BCS descrita anteriormente. Lo primero que se probó fue la lectura y la escritura de los registros del ZMD44102, en esta etapa se encontró y solucionó un error en zmdReadReg, que provocaba que se leyeran los registros en desorden.

Luego de haber validado la comunicación entre microcontrolador y transceiver, se intento comunicar el **primer** $\mathbf{I_{IE}M_{OTE}}$ (el único completamente soldado en este momento) con el Starter Kit de ZMD, haciendo uso de las rutinas TxHelloWorld y RxHelloWorld. Las pruebas fueron infructuosas, no logrando comunicación alguna.

No habiendo logrado la comunicación en RF, se intentó observar con un analizador de espectro transmitía al $\mathbf{I_{IE}M_{OTE}}$. Lo único que constatamos en esta instancia, es que ese **primer** $\mathbf{I_{IE}M_{OTE}}$ era incapaz de generar nada en RF, ni un tímido ruido. Para descartar motivos, se constato utilizando un osciloscopio, que los 2 cristales del transceiver estaban encendidos. Luego se intentó verificar todas las soldaduras de este sector del circuito, sin encontrar ninguna falla evidente.

No habiendo logrado avance alguno con ese **primer** $\mathbf{I_{IE}M_{OTE}}$, se decidió terminar de soldar otro (el **segundo** $\mathbf{I_{IE}M_{OTE}}$), pero cambiando el método de soldado del transceiver. El primero fue soldado utilizando sólo un soldador de punto, para esta segunda placa, se decidió utilizar una pistola de aire caliente para el soldado del transceiver (ambos métodos son descritos en el capítulo 6). Una vez soldada la **segunda** placa, se reintentó la comunicación con el Starter Kit, utilizando las rutinas HelloWorld y para sorpresa de algunos, la comunicación fue todo un éxito.

Comprobada la comunicación con el Starter Kit, relevamos con el analizador de espectro el comportamiento de este $\mathbf{I_{IE}M_{OTE}}$ (**segundo**) en frecuencia. La primera prueba realizada en el laboratorio fue la observación en el espectro de frecuencia de la señal emitida por la rutina de prueba del ZMD44102 pn-Code Sequence (la misma es accedida desde la aplicación BCS). Ésta genera un patrón de referencia en toda la banda de interés, con la cual se pudo constatar un espectro centrado para cada uno de los canales de la banda de frecuencia de trabajo, validando el diseño y elaboración. En la figura 8.1 se observa el espectro emitido por el ZMD44102 durante esta prueba. La antena utilizada en este caso fue el monopolo doblado.

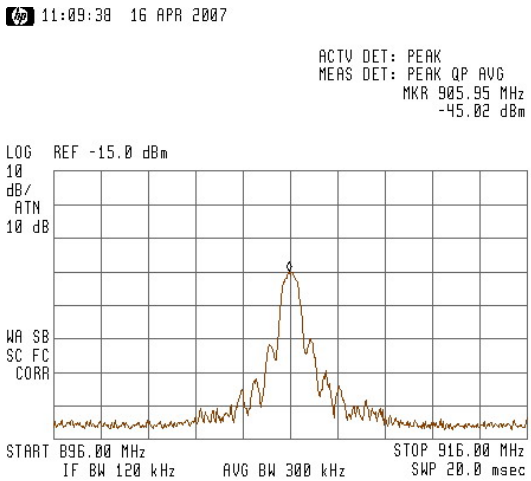

Figura 8.1: Señal emitida por la rutina pn-Code Sequence para el canal 1 (906 MHz)

Ya comprobado el buen funcionamiento, restaba tener otro **$I_{IE}M_{OTE}$** funcionando para poder probar el sistema de forma independiente. Se soldó una **tercera** placa, utilizando el método de la pistola de aire caliente. Y se intentó re-acomodar el transceiver de la **primera** placa soldada calentando la zona con la pistola de aire caliente, una vez fundido el estaño utilizando la pinza de precisión se ajusto la posición del transceiver. Terminadas ambas placas, se prosiguió haciendo las pruebas con las rutinas Tx/RxHelloWorld, y ninguna de las placas funcionó correctamente. Visto que no funcionaron correctamente, nuevamente se observo en el analizador de espectro la salida de ambas placas corriendo la rutina de prueba (pn-code sequence signal).

En la nueva placa, el **tercer** **$I_{IE}M_{OTE}$** , la misma señal mostrada en la figura 8.1 se desplazaba en frecuencia, en vez de quedar centrada en un canal, se desplazaba hacia la derecha de forma continua. Esto hizo pensar en algún error en los componentes externos del PLL, por lo cual se probaron una a una todas las soldaduras de éstos, encontrando una resistencia mal soldada. Se re-soldó la resistencia y se observo nuevamente el resultado de la rutina en el osciloscopio, y el espectro fue el esperado. Se verificó el buen funcionamiento, comunicando el **segundo** **$I_{IE}M_{OTE}$** con éste.

En el **primer** **$I_{IE}M_{OTE}$** , el espectro estaba siempre centrado en un canal más bajo al debido, estando siempre fijo por los 840MHz, también se desconfió de los componentes del PLL, pero no se encontró ningún error.

8.3.2. Caracterización del $I_{IE}M_{OTE}$

Ya contando con 2 placas funcionando, se realizaron diferentes ensayos para adquirir medidas de los parámetros de interés para caracterizar al **$I_{IE}M_{OTE}$** .

Medidas de consumo

Utilizando la aplicación BCS, se midió el consumo en los modos mostrados en la tabla 8.2.

Tabla 8.2: Modos de operación

	Microcontrolador	**Transceiver RF**	**Reg** @2.5V	**Reg** @3.3V	**Transceiver RS232**
Idle	Encendido	Encendido	Encendido	Encendido	Encendido (*)
Recibiendo	Encendido	RX	Encendido	Encendido	Dormido (*)
Enviando@0dBm	Encendido	TX^{0dBm}	Encendido	Encendido	Encendido (*)
Enviando@-16dBm	Encendido	TX^{-16dBm}	Encendido	Encendido	Encendido (*)
Enviando@-20dBm	Encendido	TX^{-20dBm}	Encendido	Encendido	Encendido (*)
Enviando@-26dBm	Encendido	TX^{-26dBm}	Encendido	Encendido	Encendido (*)
Idle dormido	Encendido	Apagado	Apagado	Dormido	Dormido (*)
Dormido	Dormido (LPM3)	Apagado	Apagado	Dormido	Dormido (*)

(*) El Transceiver RS232, en forma automática selecciona su modo de consumo, mientras detecta una señal RS232 conectada en su entrada se mantiene encendido y se duerme ante la ausencia de la misma. Para estas medidas se forzó su estado, desconectado y conectando una señal RS232.
(**LPM3**) En el modo LPM3, el microcontrolador apaga todos sus periféricos, los relojes MCLK, SMCLK, y el oscilador DCO están apagados, mientras el reloj ACLK continua operando.

Fue utilizado el modo LPM3 para el estado dormido del microcontrolador, pues permite mantener Timers

corriendo sin utilizar el procesador. De esta forma el timer puede despertar al microcontrolador, para que éste despierte al sistema entero.

Estas medidas se realizaron utilizando un osciloscopio como se muestra en la figura 8.2. Con el osciloscopio medimos la caída de tensión en una resistencia conocida, colocada en serie en la alimentación de la placa. Para suavizar los picos de consumo, y facilitar la medir de la corriente media, se utilizo un condensador en paralelo.

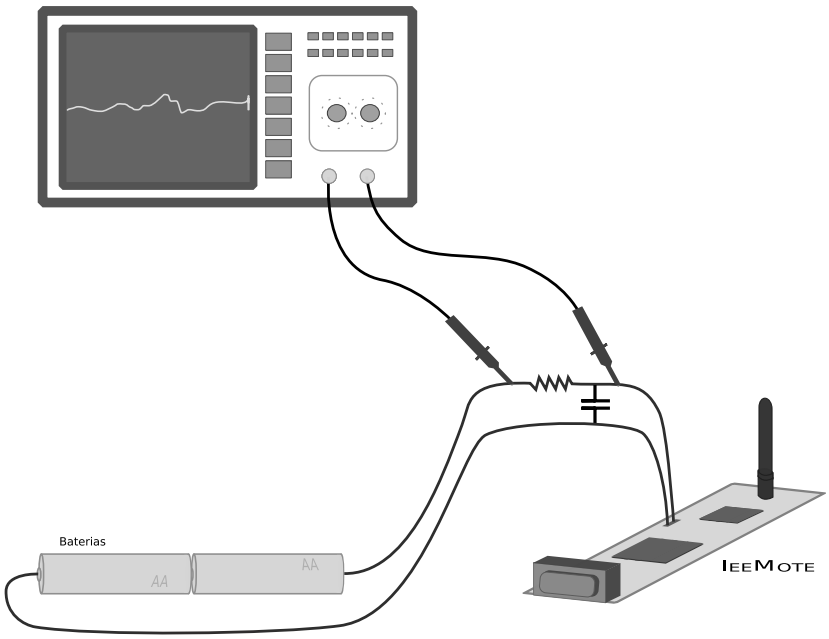

Figura 8.2: Medidas de consumo con osciloscopio

En las tablas 8.3 y 8.4 se muestran los resultados obtenidos a distinta frecuencia de operación del microcontrolador y con el driver RS-232 encendido y apagado.

Tabla 8.3: Medidas de consumo a 2.4MHz

Modo	Consumo con RS-232	Consumo sin RS-232
Idle	15.1 mA	8.3 mA
Enviando @0dBm	57.8 mA	51.5 mA
Enviando @-16 dBm	55.7 mA	48.3 mA
Enviando @-20 dBm	54.7 mA	47.4 mA
Enviando @-26 dbm	53.6 mA	46.4 mA
Recibiendo	67.4 mA	60.1 mA
Idle dormido	12.7 mA	5.5 mA
Dormido	-	17 uA

Tabla 8.4: Medidas de consumo a 770kHz

Modo	Consumo con RS-232	Consumo sin RS-232
Idle	12.0 mA	4.8 mA
Enviando @0dBm	55.8 mA	49.1 mA
Enviando @-16 dBm	51.3 mA	44.1 mA
Enviando @-20 dBm	50.4 mA	42.5 mA
Enviando @-26 dbm	49.6 mA	41.9 mA
Recibiendo	63.8 mA	56.7 mA
Idle dormido	9.2 mA	2.0 mA
Dormido	-	17 uA

Medidas de frecuencia del microcontrolador

La rutina test led de la aplicación BCS, al iniciar alterna 20 veces el pin P6.0. Utilizando un osciloscopio se capturó esta señal y se observó una frecuencia igual al doble de la frecuencia del reloj de trabajo, ya que el cambio de estado implica un ciclo de reloj. La aplicación inicializa el microcontrolador en su máxima frecuencia, configurando los osciladores internos para esto. De este ensayo se concluye que la máxima frecuencia a la cual puede operar el microcontrolador del **IIEMOTE** es $f_{MCLK} = 2.38$ MHz. La mínima frecuencia se alcanza cuando se configura el sistema para utilizar como reloj principal (MCLK) la señal generada por el cristal de 32.768 kHz.

Medidas de alcance de la comunicación

Se realizaron diversos ensayos para determinar los máximos alcances con las distintas antenas contruidas, y transmitiendo a diferentes potencias. También se realizaron pruebas para ambientes interiores y exteriores.

Como se presentó en el capítulo 7, se realizaron 2 rutinas para este ensayo, las mismas están accesibles desde la consola de la aplicación BCS.

Una rutina se encarga de enviar una ráfaga de 10000 paquetes de 10 byte separados cada 50 ms, la ráfaga se dispara con un pulsador conectado al puerto P6 y reporta el estado de la transmisión utilizando 1 LED conectado al mismo puerto. Una vez ejecutada la aplicación, se puede controlar el envió de las ráfagas sin depender de un ordenador, esto facilita mucho las pruebas en campo. El paquete enviado se presenta en la tabla 8.5, básicamente es una cadena de 10 bytes generada aleatoriamente.

Tabla 8.5: Paquete enviado

octeto	0	1	2	3	4	5	6	7	8	9
dato	0x32	0x05	0x33	0xA3	0xFF	0x1F	0xD1	0xE0	0x05	0x91

La otra rutina recibe los paquetes y mantiene tres contadores numero_de_paquetes (recibidos), paquetes_erróneos, bits_erróneos. Cada vez que recibe un paquete lo compara, y actualiza los contadores. Luego, para cada paquete recibido imprime una lista mostrando los siguientes datos:

- Leyenda: texto ingresado por el usuario al iniciar la rutina de no más de 10 caracteres, se utiliza para "marcar" las medidas para luego poder identificarlos (ej: "Monopolo10").
- numero_de_paquete: contador de paquetes recibidos hasta el momento.
- paquetes_erróneos: contador de paquetes.
- bits_erróneos: contador de bits recibidos incorrectamente.
- RSSI: Received Signal Strength Indication del enlace.
- LQI16: Link Quality Indicator de 16 bits.
- LQI8: Link Quality Indicator de 8 bits, compatible con la definición de la norma.

Todos estos datos se imprimen en una sola linea, separados por ";", logrando que sean fácilmente importados a una hoja de calculo u otra aplicación.

En la tabla 8.6 se presentan los resultados para distintas distancias y máxima potencia. En las tablas 8.7 y 8.8 se presentan los resultados transmitiendo a una distancia de 10 metros y utilizando distintas potencias, exterior e interior respectivamente. Todos los ensayos se realizaron utilizando la antena monopolo doblado, por ser la que mejor se comporta en nuestra frecuencia de interés.

Se hace especial énfasis en los ensayos a 10 metros, ya que uno de los requerimientos a cumplir implica enviar 10 bytes por segundo a 10 metros de distancia.

Tabla 8.6: Ensayos de alcance | Máxima potencia | Exterior

Distancia	**PER**
5	$0,67\,\%$
10	$0,25\,\%$
15	$0,23\,\%$
20	$0,27\,\%$
25	$0,72\,\%$
30	$0,48\,\%$
35	$0,63\,\%$
40	$0,67\,\%$
45	$3,55\,\%$
50	$14,07\,\%$
55	$16,37\,\%$

Tabla 8.7: Ensayos de alcance | 10 metros | Exterior

Potencia	**PER**
$0dBm$	$0,25\,\%$
$-16dBm$	$0,42\,\%$
$-20dBm$	$3,12\,\%$
$-26dBm$	$22,32\,\%$

Tabla 8.8: Ensayos de alcance | 10 metros | Interior

Potencia	**PER**
$0dBm$	$0,27\,\%$
$-16dBm$	$0,55\,\%$
$-20dBm$	$3,22\,\%$
$-26dBm$	$51,95\,\%$

Resumiendo, el $\mathbf{I_{IE}M_{OTE}}$ es capaz de transmitir a una distancia de 40 metros cumpliendo con el requerimiento de $PER < 1\,\%$, en un ambiente exterior. También es capaz de transmitir a 10 metros de distancia utilizando una potencia de transmisión de -16 dBm, y cumpliendo el requerimiento de $PER < 1\,\%$ en un ambiente interior y en uno exterior.

Características del $\mathbf{I_{IE}M_{OTE}}$

Luego de análisis, los ensayos y medidas se concluyen las características mostradas en la tabla 8.9.

Tabla 8.9: Caracteristicas del $\mathbf{I_{IE}M_{OTE}}$

			mínimo	típico	máximo	Unidad
Alimentación	**Vin**		1,8	-	3,6	V
Consumo	**I**	**Idle**		12.0		mA
		RX		63.8		
		TX@0dBm		55.8		
		TX@-16dBm		51.3		
		TX@-20dBm		50.4		
		TX@-24dBm		49.6		
		Dormido		0.017		
Bandas de transmisión	f_c			898/906 − 926		MHz
Frecuencia del microcontrolador	$MCLK$		32.768	-	2380	kHz
Tiempo dormido/encendido	t_{wakeup}		-	-	20	μs
Temperatura de trabajo (*)	$T_{trabajo}$		−20	25	60	°C
Alcance (@0dBm)	d	$PER < 1\,\%$			40	metro
		$PER < 4\,\%$			45	

(*) Los límites de temperatura de operación son teóricos y vienen dados por las características de todos los componentes utilizados, pero nunca se realizaron ensayos que los validen.

8.3.3. Casos de estudio | Transmisión de 10 bytes

Visto que uno de los requerimientos para el $\mathbf{I_{IE}M_{OTE}}$, es poder trasmitir a 10 metros, 10 bytes por segundo durante 1 año con una batería de $1AH$, se analiza este caso. Tomemos como caso de estudio la Transmisión de 1 paquete de 10bytes cada 1 segundo utilizando la menor potencia posible para cumplir con un $PER < 1\,\%$ ($-16dBm$) y alimentando el sistema con una batería de una capacidad de $1AH$. En este análisis, el microcontrolador siempre operará a $770kHz$ y el driver de RS-232 siempre se encontrará dormido.

El sistema se comunica a 40 kbps, por lo cual para transmitir 10 bytes se requieren:

$$tiempo\_tx = \frac{10bytes}{\frac{40kbps}{8\frac{bit}{bytes}}} = 2ms \tag{8.1}$$

Entonces, suponiendo una sincronización correcta, el sistema estará $2ms$ transmitiendo. Además se invierte $2ms$ entre el despertar del sistema, la adquisición del dato a transmitir y en dormir el sistema. Los restantes $995ms$ estará en modo sleep ($1s - 5ms = 995ms$). En la tabla 8.10 se muestra el ciclo que se repetirá todos los segundos.

Ahora, también contemplamos la posibilidad de tener un paquete perdido (o con error) y tener que retransmitir el paquete. Esto implica tener una secuencia de confirmación y retransmisión, entonces luego de enviar el paquete esperamos $1ms$ a la espera de la confirmación en modo recibiendo. Luego el rate de error de paquete es de $1\,\%$, por lo cual podemos estimar que en media son necesarios $2ms + 1\,\% = 2.02ms$

Tabla 8.10: Ciclo de recepción

Tiempo	Modo	Consumo
$2ms$	Despertando/Adquiriendo	$2.0mA$
$2ms$	Transmitiendo	$44.1mA$
$1ms$	Encendido/durmiendo	$2.0mA$
$995ms$	**dormido**	$17.0\mu A$

para transmitir un paquete, y en media el tiempo que estamos esperando por un ACK es $1ms + 1\,\% = 1.01$. Resumiendo, transmitiendo con posibilidad de recuperarse de paquetes perdidos, el sistema estará repitiendo cada 1 minutos el ciclo que se muestra en la tabla 8.11.

Tabla 8.11: Ciclo de recepción con ACK

Tiempo	Modo	Consumo
$2ms$	Despertando/Adquiriendo	$2.0mA$
$2.02ms$	Transmitiendo	$44.1mA$
$1.01ms$	Recibiendo ACK	$56.7mA$
$1ms$	Encendido/durmiendo	$2.0mA$
$993.97ms$	**dormido**	$17\mu A$

El consumo medio de la transmisión sin recuperación de perdida:

$$I^{media}_{sin\_ack} = \frac{2ms * 2mA + 2ms * 44.1mA + 1ms * 2mA + 995ms * 17\mu A}{1s} = 111.115\mu A \tag{8.2}$$

El consumo media de la transmisión con recuperación de perdida:

$$I^{media}_{con\_ack} = \frac{2ms * 2mA + 2.02ms * 44.1mA + 1.01ms * 56.7mA + 1ms * 2mA + 993.97ms * 17\mu A}{1s}$$
$$= 169.246\mu A \tag{8.3}$$

La vida útil de una batería de $1AH$ se puede calcular como:

$$t_{sin\_ack} = \frac{1AH}{I^{media}_{sin\_ack}} \approx 9000\ horas \approx 1\ año\ y\ 10\ días \tag{8.4}$$

$$t_{con\_ack} = \frac{1AH}{I^{media}_{con\_ack}} \approx 5909\ horas \approx 8\ meses,\ 2\ días\ y\ 5\ horas \tag{8.5}$$

Concluyendo, podemos estimar que la duración máxima de la carga de la batería, con el microcontrolador trabajando a $770kHz$, es de 9000 horas sin recuperación de paquetes perdidos. Por lo cual el **$\mathbf{I_{IE}M_{OTE}}$** puede funcionar de forma autónoma, transmitiendo 10 bytes cada 1s por 1 año y 10 días.

Capítulo 9

Diseño final

9.1. Introducción

En este capítulo se presenta un diseño final para la plataforma, incorporando las correcciones a los errores detectados en el prototipo y diversas mejoras.

9.2. Modificaciones y Mejoras del PCB

En el capítulo anterior (sección 8.3.1) se describen en detalle los inconvenientes encontrados en el prototipo durante la etapa de soldado de la placa.

Para el diseño final de la placa todos estos inconvenientes fueron solucionados tanto en el esquemático como en el "board" de Eagle. Además de estas correcciones, se introdujeron algunas mejoras para optimizar el diseño de la placa.

A continuación se mencionan las modificaciones y mejoras hechas para el diseño final de la placa:

- Se corrigió la pista cortocircuitada bajo el condensador C20.
- Los siguientes componentes y pines fueron conectados al plano de tierra superior:
  - Pines 62 y 63 del MSP430F149.
  - Condensadores C16 (0603) y C15 (0402).
  - Pin 1 del MAX3221.
- Se agregó máscara antisoldante en los footprints del MSP430F149, reguladores de voltaje (TPS60210 y REG711) y MAX3221.

- Se movió la pista de la resistencia R1 al pin SSK del ZMD44102 hacia el pin SSN.
- Se corrigió el footprint del regulador TPS60212 diseñado en Eagle.
- Se cambió el ruteo de los voltajes de referencia para el ADC (pines VREF+, VeREF+, VeREF- del MSP430F149) hacia los pines de salida del puerto 6.
- Se cambió el pin Rx por el pin Tx en el conector DB9 de la entrada RS232 a la placa.
- Se conectó el pin LBO del TPS60210 al puerto 2.4 del MSP430F149.
- Se agregaron señales de alimentación y tierra a los conectores de los puertos 6 (puerto ADC) y puerto 5 (entradas/salidas).
- Se incorporó relleno de tierra en la capa de señal inferior.
- Se quitó el camino alternativo de red de adaptación (capa de señal superior).
- Se deja accesible el segundo puerto uart del microcontrolador desde conectores de expansión.
- Se dejan accesibles 2 pines de interrupción al microcontrolador (P2.0-P2.1) desde conectores de expansión
- Visto los problemas encontrados con el ruido y JTAG, se tomó mayor precaución en el ruteo de estas señales. En particular se separaron las señales en el ruteo para lograr que siempre exista relleno de tierra entre ellas.

9.3. Esquemático

A continuación se presentan los esquemáticos del diseño final.

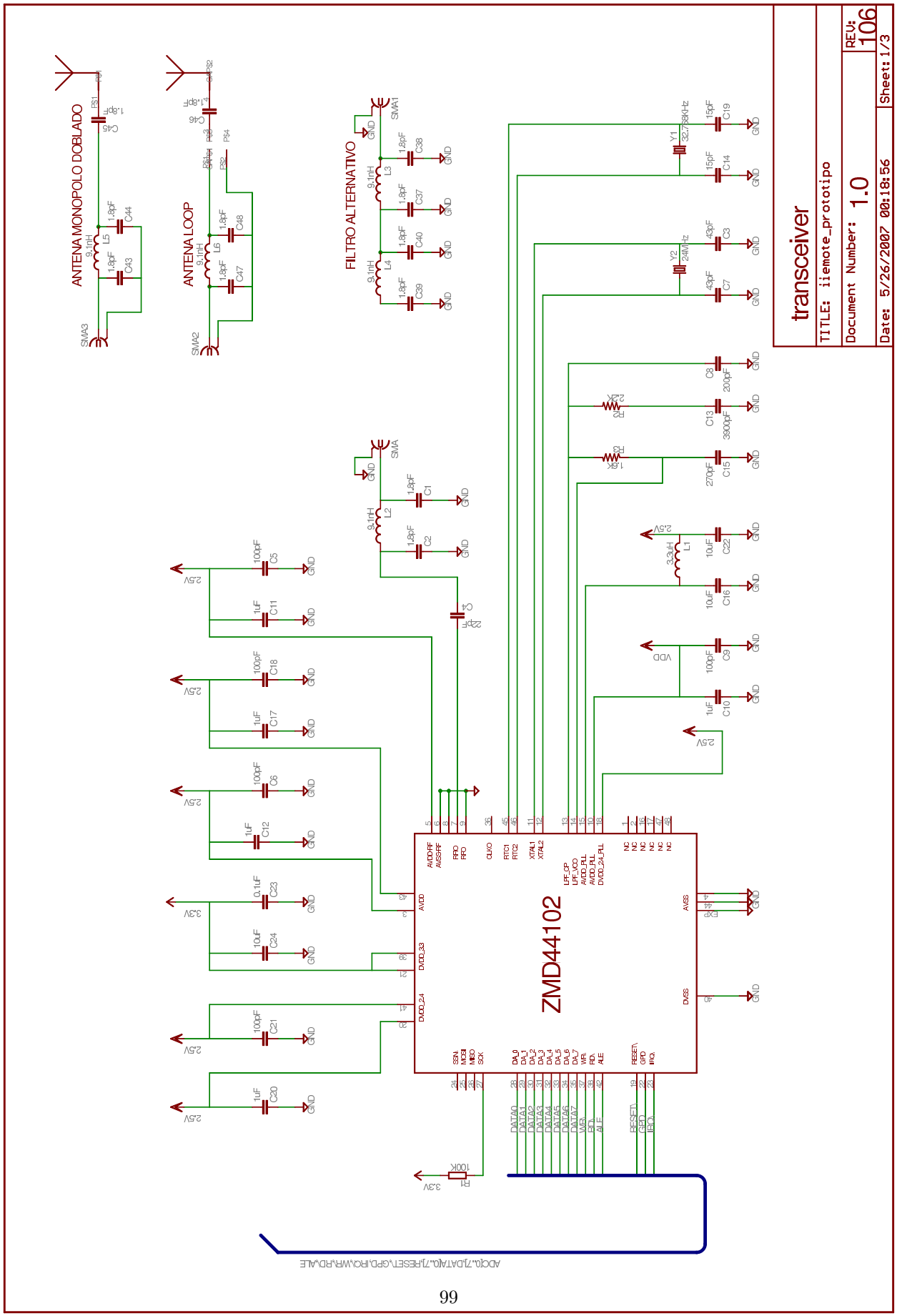

Figura 9.1: Esquemático Transceiver

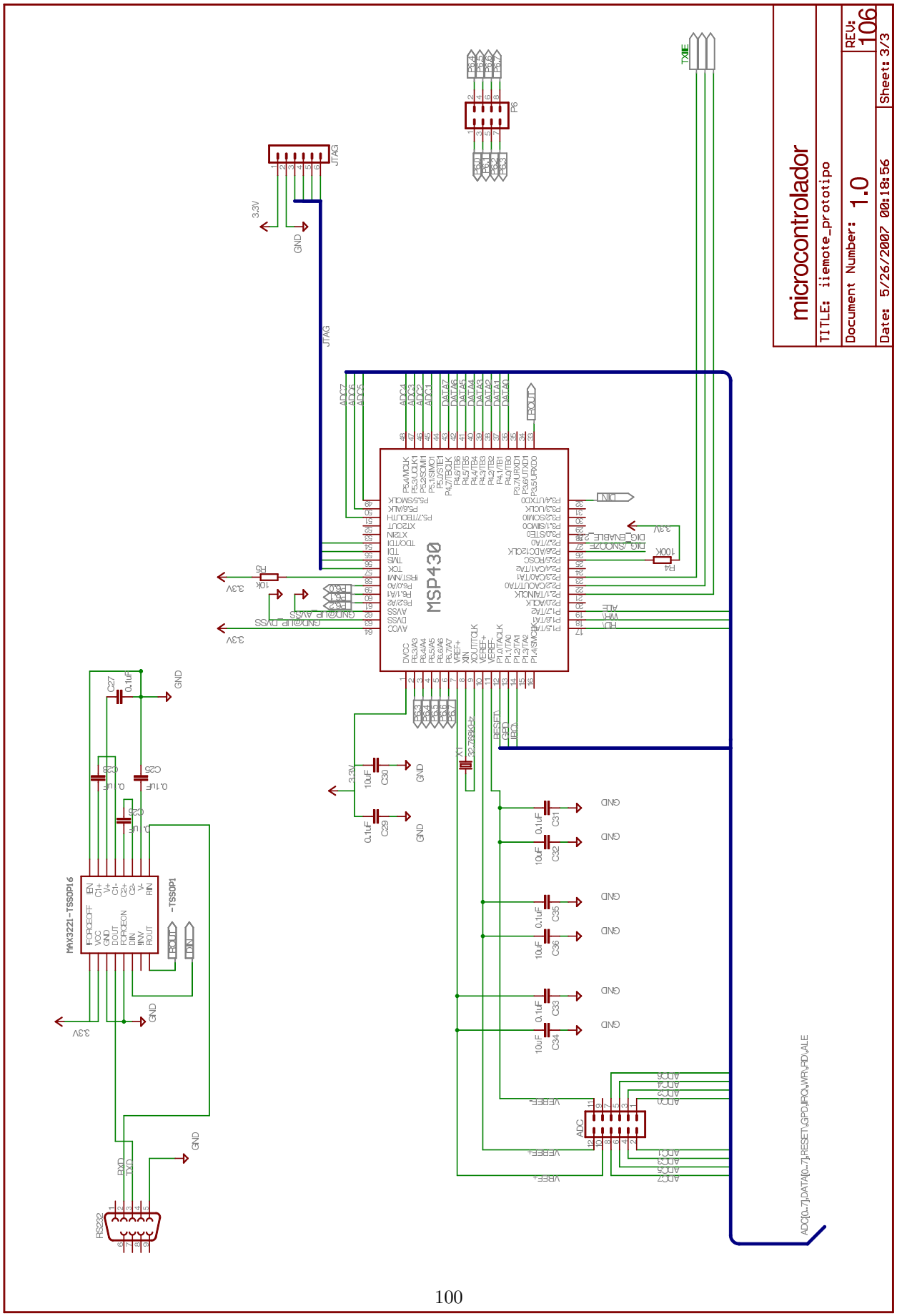

Figura 9.2: Esquemático Microcontrolador

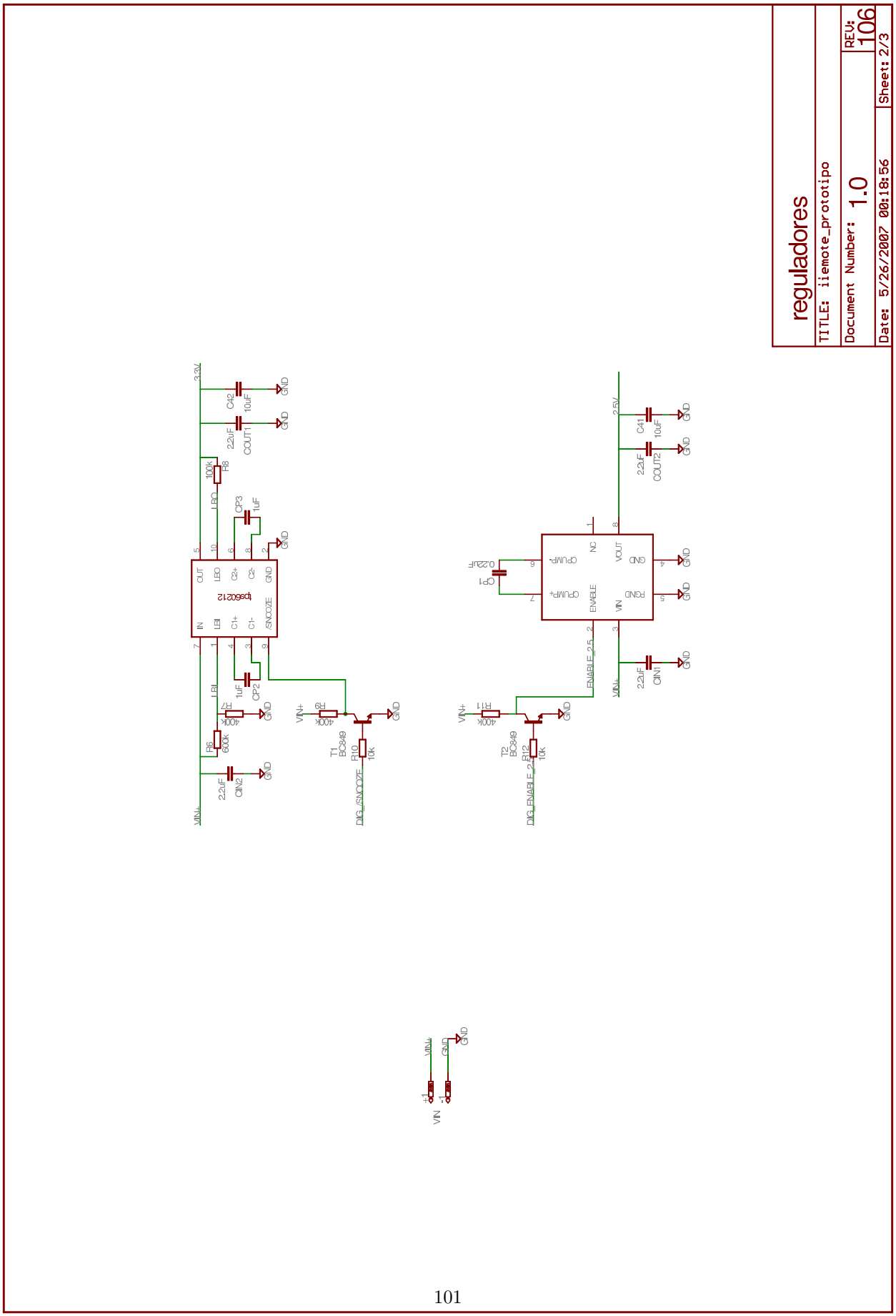

Figura 9.3: Esquemático Alimentación

9.4. PCB

A continuación se presentan la posición de los componentes y la capa superior del PCB para el diseño final. Las figuras de todas las capas del diseño final se encuentran en el apéndice B).

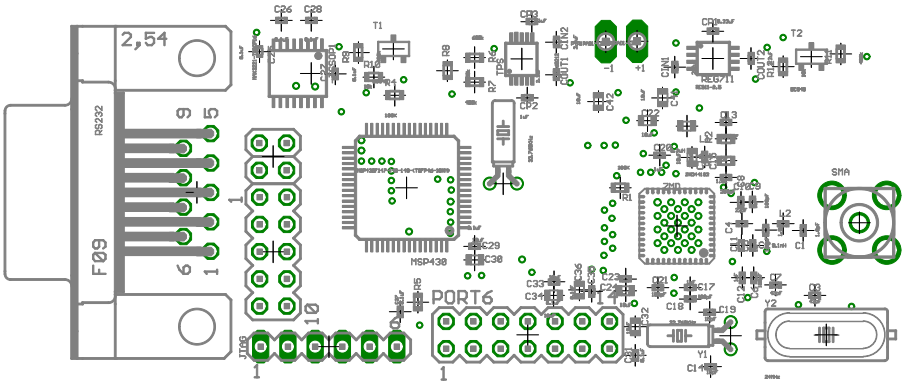

Figura 9.4: Posición de los componentes en la placa - diseño final

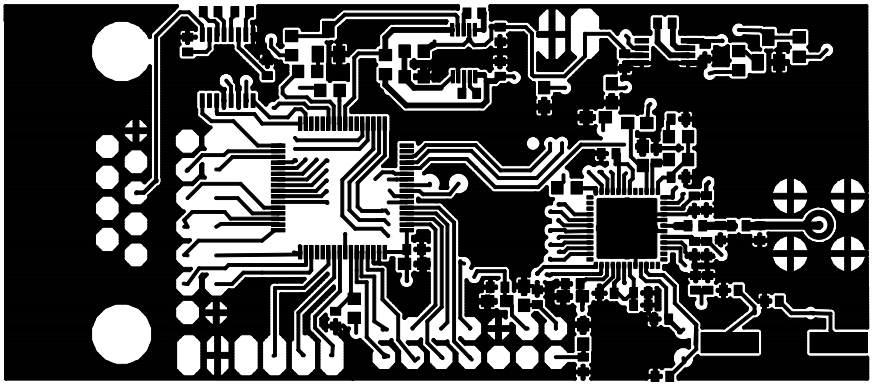

Figura 9.5: Capa superior del PCB - diseño final

Capítulo 10

Evaluación del proyecto

10.1. Introducción

En este capítulo se realiza el análisis de costos del **IIEMOTE** , se evalúa la plataforma de hardware diseñada y una evaluación final del proyecto en general.

10.2. Análisis de costos

En este análisis se estudiaron los costos de la fabricación de un **IIEMOTE** , así como los costos de la fabricación de un lote de 20.

10.2.1. Costos por unidad

En la tabla 10.1 se presentan los precios de los distintos componentes que componen el sistema, incluyendo la fabricación de la placa del **IIEMOTE** (PCB).

Todos los precios son en dólares americanos, no incluyen impuestos ni costos de envío.

Para el prototipo, los circuitos impresos fueron fabricados por PCBExpress, consta de 4 capas, mascara de anti-soldado y pistas de "leyendas". El costo de la fabricación del PCB fue de 250 U$S la unidad.

Visto que para la fabricación de estas unidades se compran poca cantidad de componentes, es necesario recurrir a distribuidores minoristas, lo que aumenta sensiblemente los costos. A modo de comparación, en la tabla 10.2 se muestran los precios de los componentes de Texas Instrument comprados en grandes lotes directo del fabricante. Se puede apreciar que si se fabricara en grandes cantidades, el preció podría reducirse de una forma notoria.

Tabla 10.1: Costos de componentes por unidad y en dólares

Componente	**Descripción**	**Fabricante**	**Proveedor**	**Costo**
ZMD44102	Transciver	ZMD	Future-Electronics	8.55
MSP430F149IPM	Microcontrolador	Texas Instruments	Digi-Key	10.08
TPS60212DGS	Regulador de 3.3V	Texas Instruments	Digi-Key	2.38
REG711EA-2.5	Regulador de 2.5V	Texas Instruments	Digi-Key	2.25
MAX3221CDBR	SR232	Texas Instruments	Digi-Key	1.38
L1	Inductor 3.3uH	TDK Corporation	Digi-Key	0.41
L2	Inductor 9.1nH	Murata	Digi-Key	0.43
Y1	Cristal 32.768KHz	Citizen	Digi-Key	0.70
X1	Cristal 32.768KHz	Citizen	Digi-Key	0.70
Y2	Cristal 24.000MHz	Fox Electronics	Digi-Key	0.91
Circuito impreso	PCB de 4 capas	PCBExpress	PCBExpress	250
CONN SMA PCB	Conector SMA	Amphenol-RF	Digi-Key	3.23
Componentes	Capacitores y resistencias	Varios	Digi-Key	30.19
Conectores			Eneka	4
TOTAL				**315.21**

Tabla 10.2: Precios Texas Instruments

Componente	**Descripción**	**Cantidad mínima**	**Costo**	**Total**
MSP430F149	Microcontrolador	160	6.05	968
TPS60212DGS	Regulador de 3.3V	80	0.95	70
REG711EA-2.5	Regulador de 2.5V	80	0.99	79.2
MAX3221CDBR	SR232	250	0.88	220

10.2.2. Costos 20 unidades

Tabla 10.3: Costos de componentes en dólares

Componente	Fabricante	Vendedor	Cantidad	Costo	Total
ZMD44102	ZMD	Future-Electronics	20	8.55	171
MSP430F149IPM	Texas Instruments	Digi-Key	20	10.08	201.6
TPS60212DGS	Texas Instruments	Digi-Key	25	1.9	47.5
REG711EA-2.5	Texas Instruments	Digi-Key	20	2.25	45
MAX3221CDBR	Texas Instruments	Digi-Key	25	1.10	27.5
L1	TDK Corporation	Digi-Key	20	0.34	6.8
L2	Murata	Digi-Key	20	0.36	7.2
Y1, X1	Citizen	Digi-Key	40	0.60	24
Y2	Fox Electronics	Digi-Key	20	0.787	15.7
Circuito Impreso	PCBExpress	PCBExpress	20	375	375
CONN SMA PCB	Amphenol-RF	Digi-Key	20	3.23	66.6
Componentes	Varios	Digi-Key	20	30.19	603.8
Conectores	Eneka	Eneka	20	4	80
TOTAL					**1672**

Al aumentar la cantidad de unidades a 20, los costos de los componentes bajan como se muestra en la tabla 10.3. El costo que baja más significativamente es el del circuito impreso (PCB) ya que el ingreso a fabrica tiene un costo fijo grande. Encargar 20 placas en el extranjero tiene un costo de 375 U$S, mientras que encargar 2 placas vale 250 U$S.

El costo de comprar los componentes para la fabricación de 20 $\mathbf{I_{IE}M_{OTE}}$ es de 1672 U$S en total, lo cual da 84 U$S por $\mathbf{I_{IE}M_{OTE}}$. El precio bajó sensiblemente de 315 U$S (1) a 84 U$S (20). Es importante destacar que estos costos no incluyen costos de diseño, ni la mano de obra necesaria para el soldado y las pruebas de cada unidad.

10.3. Evaluación general del $\mathbf{I_{IE}M_{OTE}}$

El hardware desarrollado tiene múltiples aplicaciones y se llegó a un diseño flexible capaz de adaptarse a un gran número de casos de uso. Sin duda el MSP430 es un muy buen microcontrolador y logra cumplir las expectativas de versatilidad y robustez.

El ZMD44102 demostró ser una muy buena alternativa. La sub-capa MAC que integra hace que el uso del mismo sea muy sencillo, posibilitando el control del mismo desde rutinas de software simples y por lo tanto aptas para bajo consumo.

Al iniciar el análisis, se manejó como criterio que el microcontrolador contara con una versión de TinyOS portada, en esa instancia se pensaba en que esto pudiera darle viabilidad al proyecto, logrando que el

IEMOTE mismo fuera "seductor" para otros grupos de investigación, ya que al momento no se cuenta con hardware compatible con la norma IEEE 802.15.4 en la banda 868/915 MHz que sea también compatible con TinyOS. Se inició la migración de TinyOS v2 a esta plataforma, pero lamentablemente, en el transcurso de este proceso ZMD saco del mercado el transceiver, comunicando lo siguiente en su web:

> *"ZMD recientemente anunció un acuerdo de desarrollo con Renesas Technology of Japan. Bajo este acuerdo ZMD colaborará con Renesas en el desarrollo de la próxima generación de soluciones 802.15.4 y ZigBee a 900MHz. El diseño juntará la tecnología de ZMD en RF 900MHz con la solución de microcontroladores líderes en la industria de Renesas. Como resultado de este anuncio el dispositivo de 900MHz RF ZMD44102 no va a tener más soporte como un producto de producción."*[1]

Luego de esto se intentó contactar a ZMD, se enviaron e-mails sin obtener ninguna respuesta. Esta situación terminó desmotivando el desarrollo de un driver (Radio Stack) para TinyOS del ZMD44102, quedando inconclusa esta tarea. Sin duda esta noticia tira por tierra las grandes ventajas apreciadas en este transceiver.

Si bien se alcanzaron los objetivos de compatibilidad con la norma, y de versatilidad de la plataforma, los resultados obtenidos en el consumo del sistema se alejan de las previsiones teóricas que se realizaron en la etapa de análisis. Esto principalmente al no tomar en cuenta la eficiencia de los reguladores de voltaje. Igualmente el consumo obtenido es aceptable visto la versatilidad del sistema y el alcance logrado.

10.4. Evaluación general del proyecto

Como proyecto de fin de carrera, fue muy enriquecedor y se abordaron un gran número de materias dentro de la ingeniería eléctrica, permitiendo tener contacto con el diseño, la planificación y la solución de problemas en áreas como las Telecomunicaciones, Radio Frecuencia, Electrónica y el desarrollo de software embebido.

En muchos casos se notó la falta del instrumental e infraestructura adecuada, y se invirtió mucho tiempo en la solución "artesanal" de problemas "*ya resueltos a nivel industrial*".

Es importante destacar que el costo de sistema es bajo y competitivo en el mercado, aunque al ZMD haber retirado el transceiver, no es claro el futuro del diseño.

[1] Versión original en ingles: "*ZMD have recently announced a joint development agreement with Renesas Technology of Japan. Under this agreement ZMD will collaborate with Renesas on the development of next generation 900MHz 802.15.4 and ZigBee solutions. These designs will bring together ZMD's 900Mhz RF technology and the industry leading microcontroller solutions of Renesas. As a result of this announcement the ZMD 44102 900Mhz RF device will no longer be supported in the market as a full production item.*"

Apéndice A

Reseña de la norma (IEEE 802.15.4)

A.1. Medio físico

La norma prevé la comunicación en varias bandas de frecuencia, para adaptarse de forma óptima a las distintas aplicaciones, respetando la normativas de todos los países. En la tabla A.1 se presentan las bandas en que la norma opera, resumiendo también algunas características de la modulación.

Tabla A.1: Bandas de frecuencia y Tasas de transmisión

PHY (MHz)	**Banda de frecuencia (MHz)**	**Parámetros del Spreading**		**Parámetros de Datos**		
		Chip rate $\left(\frac{kchip}{s}\right)$	**Modulación**	**Tasa** $\left(\frac{kbit}{s}\right)$	**Tasa** $\left(\frac{ksímbolo}{s}\right)$	**Símbolos**
868/915	868-868.6	300	BPSK	20	20	Binario
	902-928	600	BPSK	40	40	Binario
2450	2400–2483.5	2000	O-QPSK	250	62.5	16-ary Orthogonal

En nuestra banda de interés (868/915 MHz), se asignan 10 canales de frecuencia, centrados en $f_c = 906 + 2(k-1)MHz$, con $k = 1, 2, ..., 10$. Se deben emplear DSSS (Discrete Sequence Spread Spectrum) con BPSK (Binary Phase Shift Keying) para la modulación de los chips, y Differential Encoding para la codificación de los datos. La tasa de datos será de 40 kb/s.

A.1.1. Modulación y spreading — 868/915 MHz

En la figura A.1 se resume el esquema de modulación propuesto por la norma 802.15.4 como referencia para las bandas 868/915 MHz. Cada bit de un paquete tiene que ser procesado con "differential encoding". La correspondencia o mapeo bit-to-chip y las funciones de modulación son ejecutadas para cada octeto, procesándose siempre primero el bit menos significativo (LSB).

Figura A.1: Modulación

Differential encoding

Differential encoding es la suma en módulo dos (XOR) de los bits de datos que se quieren codificar (datos crudos) con los datos previamente codificados. Esta operación es llevada a cabo por el transmisor y se puede describir en la siguiente ecuación:

$$E_n = R_n \oplus E_{n-1} \tag{A.1}$$

Siendo R_n es el bit a codificar, E_{n-1} es el bit previamente codificado, y E_n es el bit resultante de la codificación. Por cada paquete transmitido, R_1 es el primer bit codificado y E_0 se asume con valor cero. Análogamente, el proceso de decodificación realizado por el receptor se puede describir en la siguiente ecuación:

$$R_n = E_n \oplus E_{n-1} \tag{A.2}$$

Por cada paquete recibido, E_1 es el primer bit en ser decodificado y E_0 se asume con valor cero.

A.1.2. Mapeo bit-to-chip

Cada bit es mapeado en una secuencia PN[1] de 15-chip acorde a la tabla A.2

Tabla A.2: Mapeo símbolo a chip

Input bits	**Chip values** ($c_0 c_1 \dots c_{14}$)
0	111101011001000
1	000010100110111

A.1.3. Modulación BPSK

La secuencia de chips es modulada en la portadora usando BPSK. En la la banda de 868 MHz se transmite a una tasa de 300 kchip/s y para la banda de 915MHz se transmite a una tasa de 600 kchip/s.

[1] Pseudo-random Noise sequence: secuencia generada para ser estadísticamente aleatoria

La forma del pulso (Pulse-shape) utilizada para representar cada chip de banda base es el coseno elevado o "raised cosine" (roll-off factor=1) y está descrito por la siguiente ecuación:

$$p(t) = \frac{sen(\pi t/T_C)}{\pi t/T_C} \cdot \frac{cos(\pi t/T_C)}{1 - 4t^2/T_C^2} \tag{A.3}$$

Se envía primero el chip menos significativo (c_0) y por último el chip más significativo (c_{14}).

A.2. Especificaciones de la banda de radio

Se deben cumplir todas las regulaciones de las regiones, y además, todo equipo operando en las bandas 868/915/MHz deberá cumplir los requerimientos que se detallan a continuación.

A.2.1. Máscara en la densidad espectral de potencia de transmisión

El espectro transmitido deberá estar fuera de los límites dados por la tabla A.3.

Tabla A.3: Mascara de transmisión

Frecuencia	Límite relativo	Límite absoluto
$\|f - f_c\| < 1.2MHz$	$-20dB$	$-20dBm$

Tanto para el límite relativo como para el absoluto, la potencia espectral promedio debe ser medida con una resolución de ancho de banda de 100 kHz. Para el límite relativo, el nivel de referencia debe ser la mayor potencia espectral promedio medida a ± 600 kHz de la frecuencia portadora f_c.

A.2.2. Sensibilidad del receptor

El receptor deberá ser capaz de alcanzar una sensibilidad de -92dBm o mejor. Las condiciones son tales que el PER (Packet Error Rate) sea menor a 1 %, y que la potencia sea medida en las terminales de la antena, que la interferencia no esté presente y que el largo del paquete a transmitir por el medio físico, sea igual a 20 octetos.

A.2.3. Tiempos de retorno

El tiempo de retorno TX-RX es el lapso comprendido desde el momento que se deja de transmitir y se comienza a recibir. Este debe ser medido desde el momento que se terminó de enviar el ultimo símbolo transmitido hasta el momento en que el receptor esté listo para recibir el próximo paquete PHY.

El tiempo de retorno RX-TX es el lapso comprendido desde el momento que se deja de recibir y se comienza a transmitir. Este debe ser medido desde el momento que se termino de recibir el ultimo símbolo del ultimo paquete recibido hasta el momento en que el transmisor comienza a transmitir el *acknowledgment* resultante.

Tanto el tiempo de retorno TX-RX como el RX-TX deben ser menores a $aTurnaroundTime = 12$ períodos de símbolo (transmitiendo a una tasa de 40 ksímbolos/s este tiempo equivale a unos $293\mu s$ aproximadamente).

A.2.4. EVM

La precisión en la modulación de un transmisor IEEE 802.15.4 queda determinada midiendo el EVM (Error-Vector Magnitude). Un transmisor IEEE 802.15.4 debe tener valores EVM menores a un 35 % para una medición de 1000 chips.

A.2.5. Tolerancia de la frecuencia central de transmisión

La tolerancia de la frecuencia central de transmisión f_c debe ser $\pm$ $40ppm$ como máximo.

A.2.6. Potencia de transmisión

El transmisor deberá ser capaz de transmitir al menos -3dBm.[1]

A.2.7. Nivel máximo de la señal deseada a la entrada del receptor

El máximo nivel de entrada en el receptor es el máximo nivel de potencia de la señal deseada (en dBm) presente a la entrada del receptor para la cual el criterio de tasa de error es alcanzado. El receptor deberá tener un nivel de señal máxima deseada mayor o igual a -$20dBm$ en la entrada.

A.3. Physic Protocol Data Unit (PPDU)

Los paquetes de datos de capa física son acorde a la tabla A.4, y esta compuesta por los siguientes componentes:

- *SHR* - Encabezado de sincronización (Synchronization Header), que permite que el receptor se sincronice y fije la ruta del stream de bits.
- *PHR* - Encabezado de capa física (PHY Header), que contiene información sobre el largo de los datos.

[1] El transmisor diseñado por el IIE transmitirá como máximo 2dBm

- *PSDU* - Datos (PHY Service Data Unit) o carga (payload) de largo variable, que lleva la trama MPDU (MAC protocol data unit) de la subcapa MAC.

Tabla A.4: Formato de los paquetes de capa física

32 bits	**8 bits**	**7 bits**	**1 bit**	**largo variable**
Preamble	SFD	Largo de trama	Reservado	PSDU
SHR		PHR		PSDU
Header				Payload

A.3.1. Preamble

El campo Preamble (preámbulo) es utilizado por el transceiver para obtener sincronización de los chips y símbolos cuando ingresa un mensaje. El preámbulo está compuesto por 32 ceros binarios.

A.3.2. SFD

El campo SFD (start-of-frame delimiter) indica el fin de la sincronización y el comienzo del encabezado de los datos. La secuencia de bits debe ser la siguiente:

bit 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	bit 7
1	1	1	0	0	1	0	1

A.3.3. Largo de trama

El largo de trama es un campo que especifica la cantidad de octetos que contienen los datos. Es un valor entre 0 y *aMaxPHYPacketSize*. La siguiente tabla resumen los distintos tipos de datos para diferentes valores de largo de trama:

Valores de largo de trama	**Payload**
0-4	Reservado
5	MPDU (Acknowledgment)
6-7	Reservado
8 a *aMaxPHYPacketSize*	MPDU

A.3.4. PSDU

El campo PSDU es de largo variable y contiene los datos del paquete de la capa PHY. Para todos los valores de largo de trama 5 o mayores a 7 octetos, el PSDU contiene la trama MPDU de la subcapa MAC.

Apéndice B

PCB

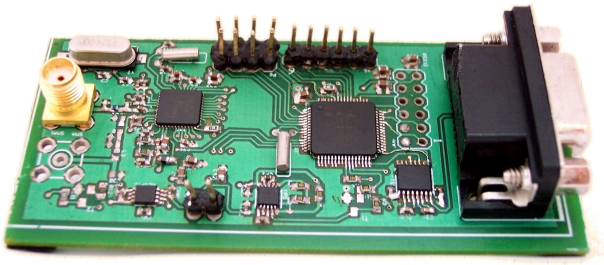

Figura B.1: Prototipo $\mathbf{I_{IE}M_{OTE}}$

B.1. Capas del prototipo

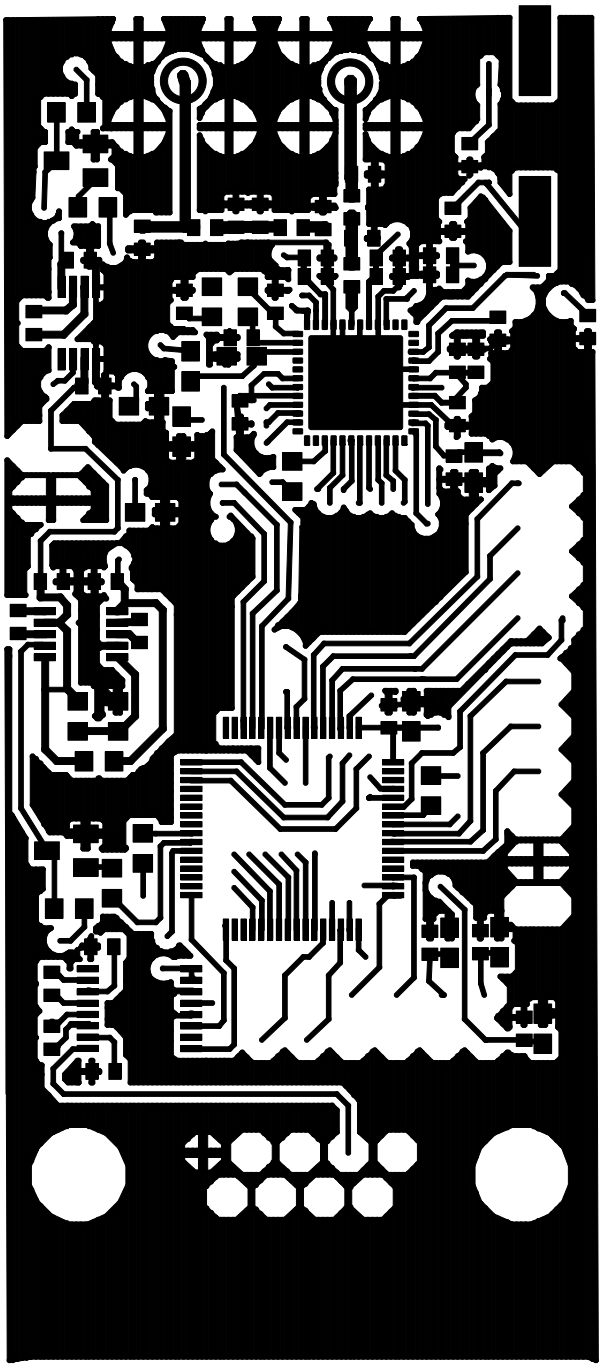

Figura B.2: Capa superior del PCB

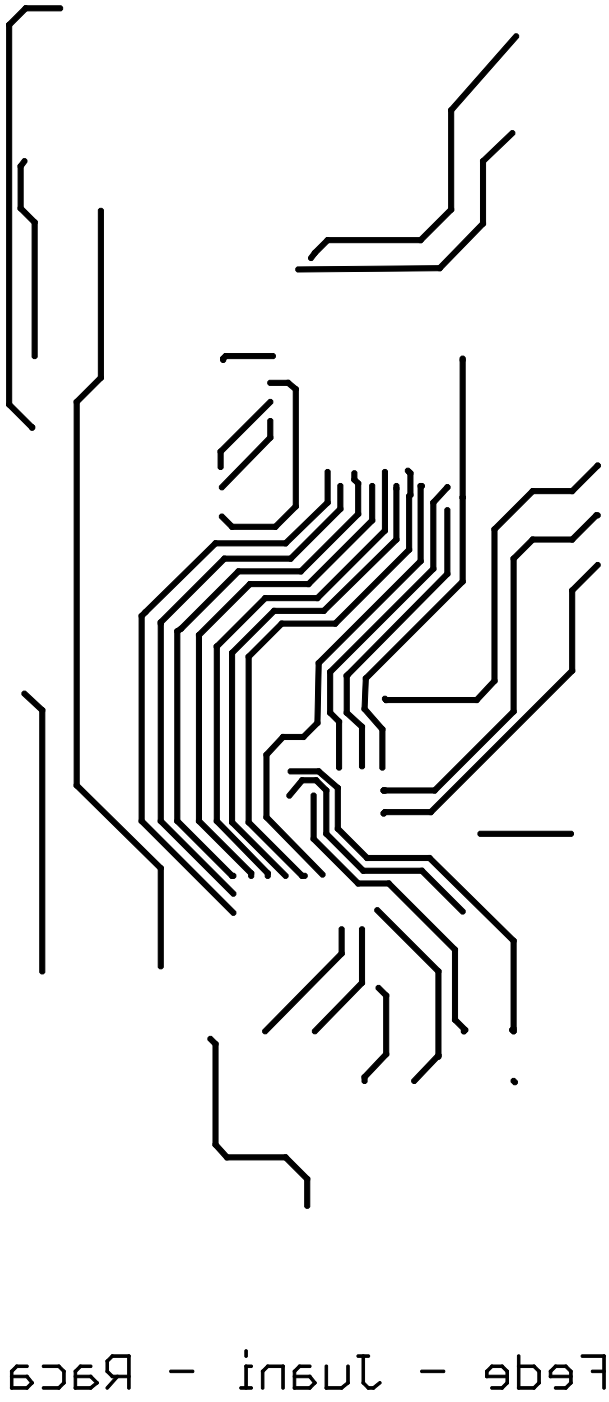

Figura B.3: Capa inferior del PCB

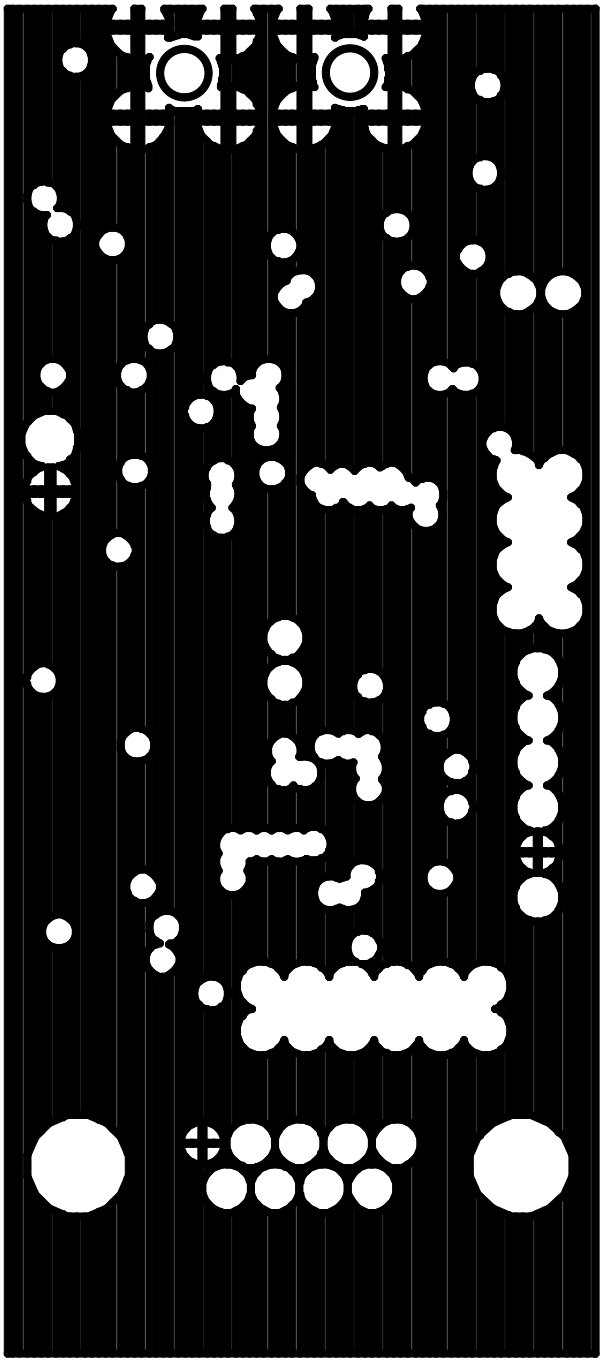

Figura B.4: Capa de tierra

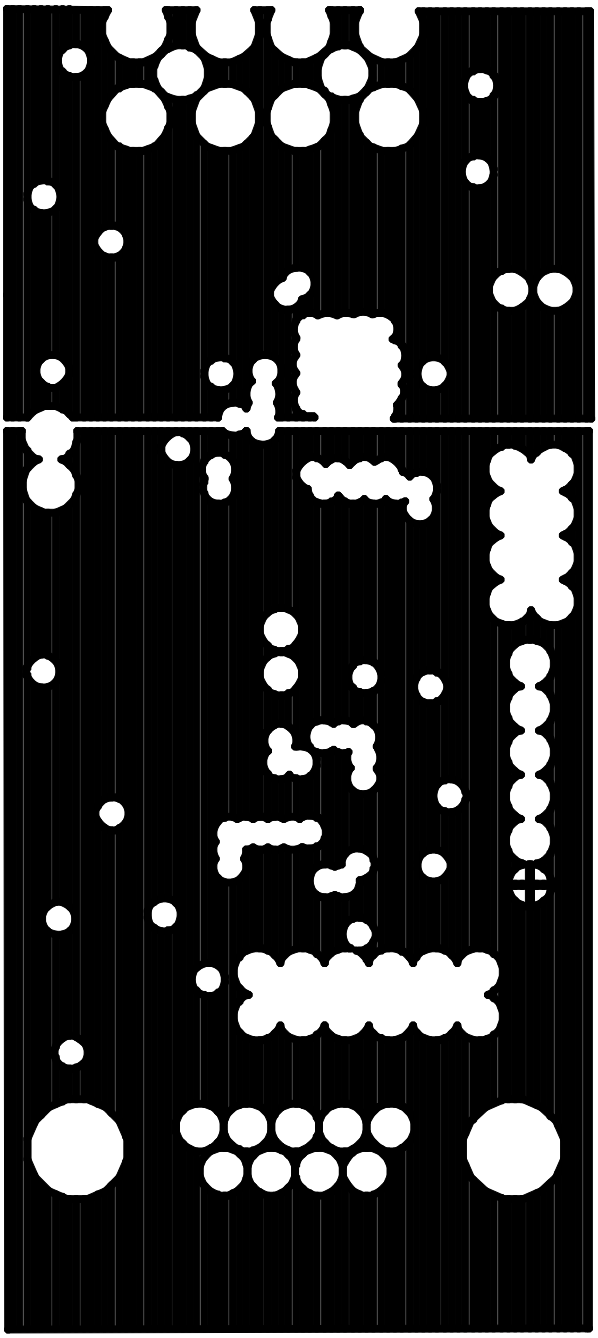

Figura B.5: Capa de alimentación

B.2. Capas del diseño final

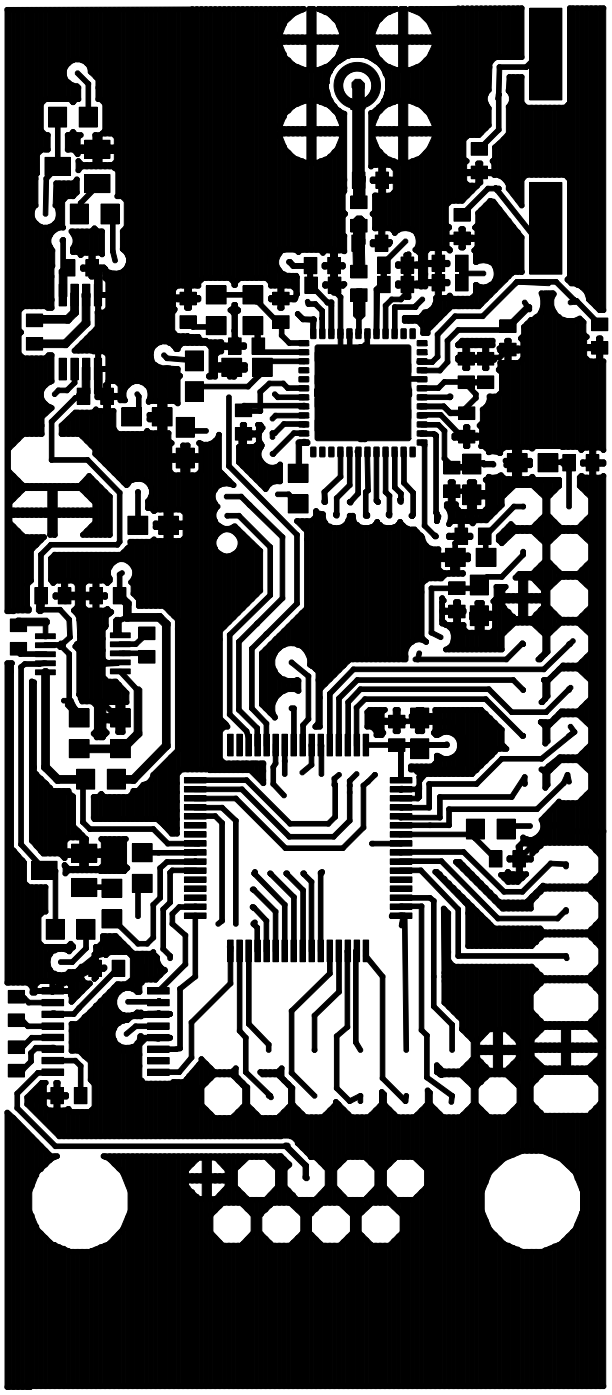

Figura B.6: Capa superior del PCB

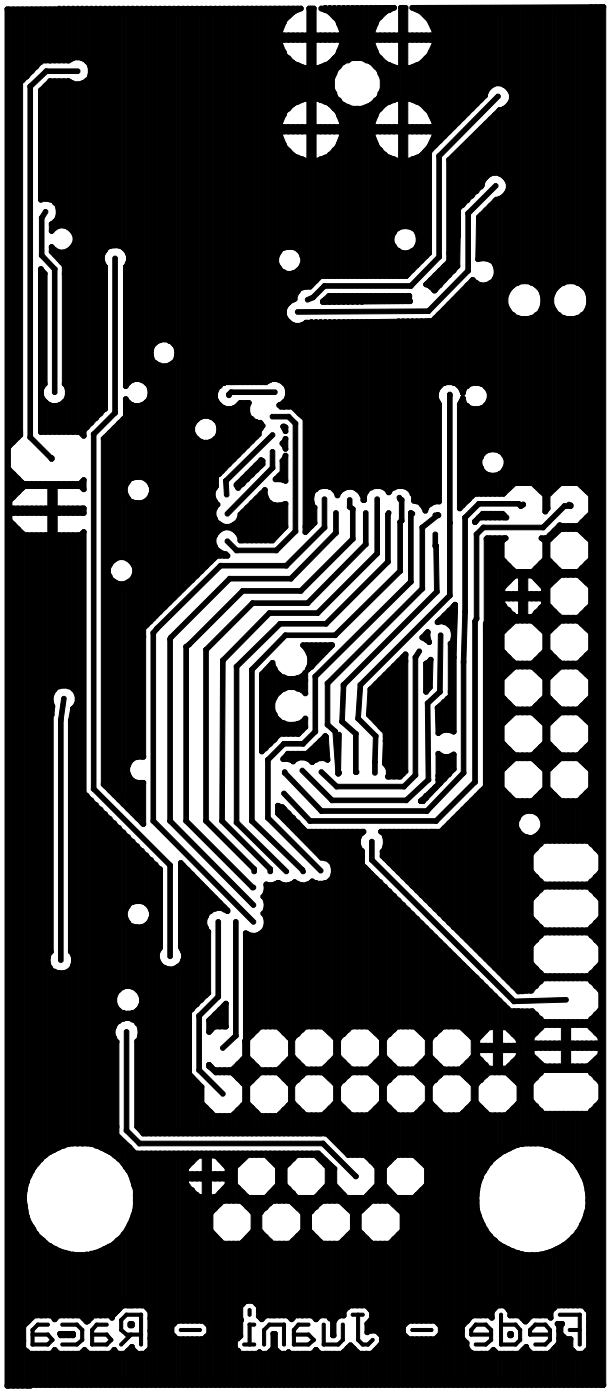

Figura B.7: Capa inferior del PCB

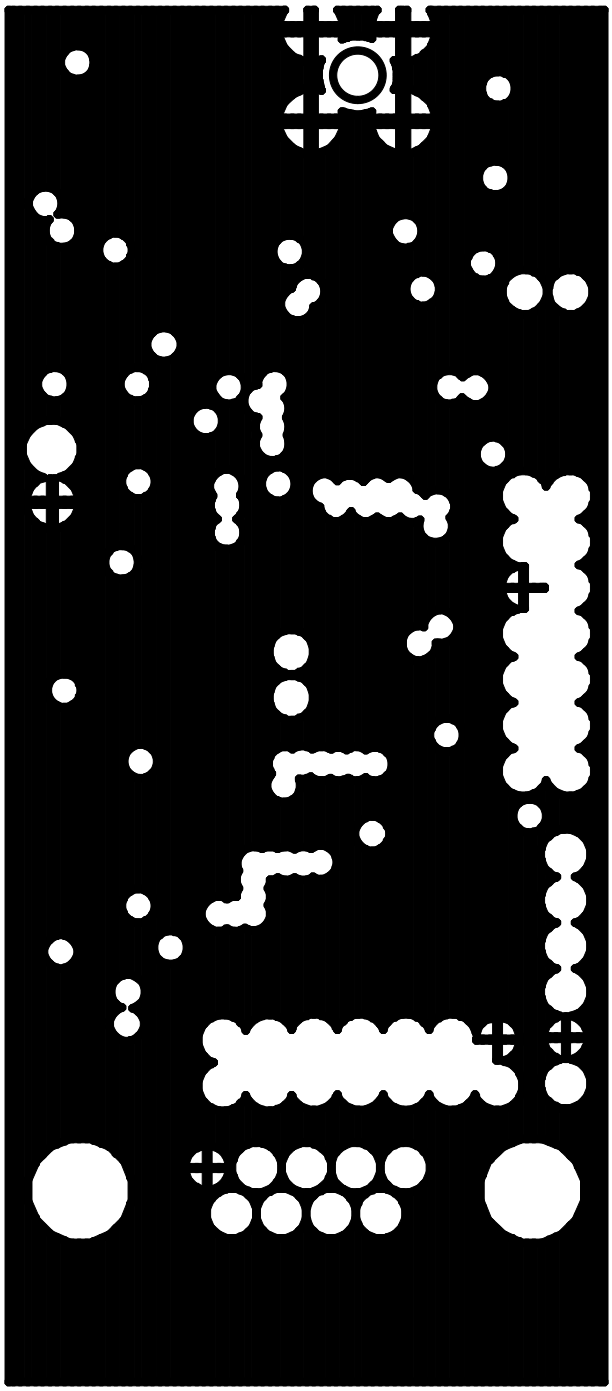

Figura B.8: Capa de tierra

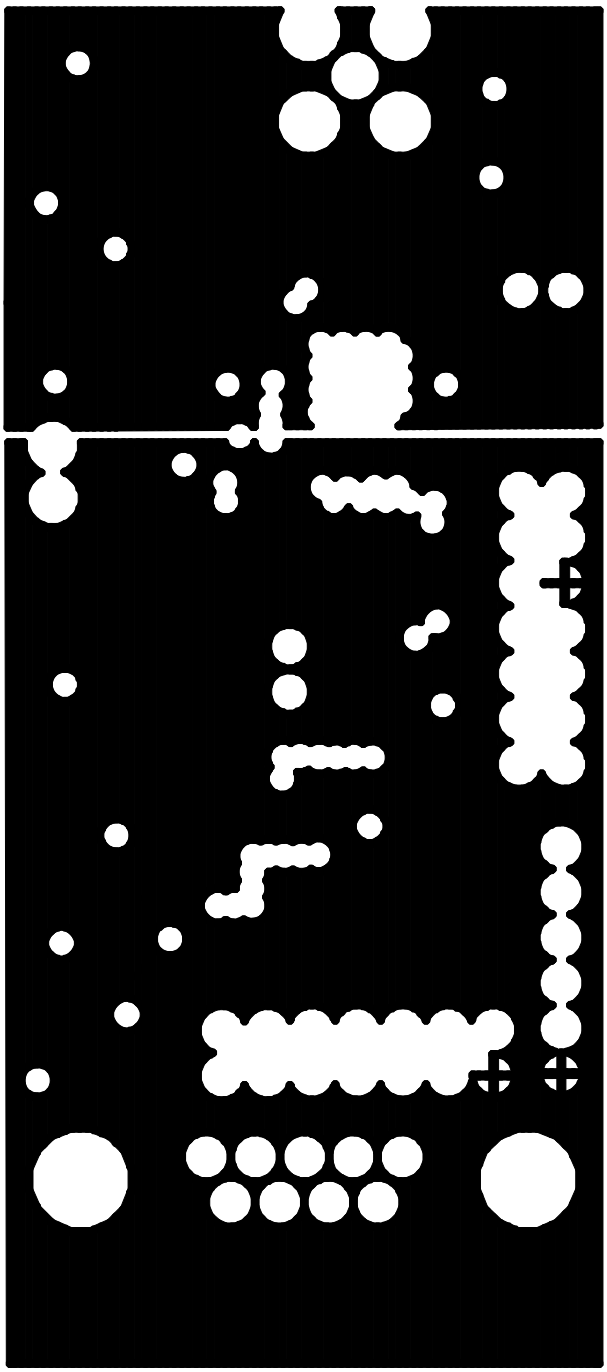

Figura B.9: Capa de alimentación

B.3. Lista de componentes

En la tabla B.1 se listan los componentes utilizados en el **IEMOTE** y en la tabla B.2 se listas los detalles de los capacitores y resistencias utilizados.

Tabla B.1: Lista de componentes

Componente	Descripción
ZMD44102	Transciver
MSP430F149IPM	Microcontrolador
TPS60212DGS	Regulador de 3.3V
REG711EA-2.5	Regulador de 2.5V
MAX3221CDBR	SR232
L1	Inductor 3.3uH
L2	Inductor 9.1nH
Y1	Cristal 32.768KHz
X1	Cristal 32.768KHz
Y2	Cristal 24.000MHz
Placa	PCB de 4 capas
CONN SMA PCB	Conector SMA
C1 C2	CAP CER 1.8PF 50V 0402 HIGH FREQ
C14 C19	CAP 15pF 50V CERAMIC 0402 SMD
C4	CAP 22pF 50V CERAMIC 0402 SMD
C3 C7	CAP CERAMIC 43pF 5 % 50V C0G 0402
C5 C6 C9 C18 C21	CAP 100pF 50V CERAMIC X7R 0402
C8	CAP 200pF 50V CERAMIC C0G 0402
C15	CAP 270pF 50V CERAMIC X7R 0402
C13	CAP 3900pF 50V CERAMIC X7R 0402
C23 C25 C26 C27 C28 C29 C31 C33 C35	CAP 0.1uF 10V CERAMIC X5R 0402
C10 C11 C12 C17 C20 CP2 CP3	CAP CER 1.0uF 10V Y5V 0402
C16 C22 C24 C30 C32 C34 C36 C41 C42	CAP CER 10uF 6.3V 20 % X5R 0603
CIN1 CIN2 COUT1 COUT2	CAP CERAMIC 2.2UF 6.3V X5R 0402
CP1	CAP CERM .22UF 10V Y5V 0402
R1 R4 R8	RES 100K OHM 1/10W 5 % 0603 SMD
R2	RES 2.2K OHM 1/10W 5 % 0603 SMD
R3	RES 1.6K OHM 1/10W 5 % 0603 SMD
R5 R10 R12	RES 10K OHM 1/10W 5 % 0603 SMD
R6	RES 604K OHM 1/10W 1 % 0603 SMD
R7 R9 R11	RES 402K OHM 1/10W 1 % 0603 SMD
Conector SR232	conector SR232 hembra
Conector ADC	pines
Conector de test	pines
Conector JTAG	pines

Tabla B.2: Costos de capacitores y resistencias

Componente	**Descripción**	**Digi-Key**	**Cantidad**	**precio por unidad**
C1 C2	CAP CER 1.8PF 50V 0402 HIGH FREQ	399-1003-1-ND	2	0.061
C14 C19	CAP 15pF 50V CERAMIC 0402 SMD	PCC150CQCT-ND	2	0.019
C4	CAP 22pF 50V CERAMIC 0402 SMD	PCC220CQCT-ND	1	0.019
C3 C7	CAP CERAMIC 43pF 5 % 50V C0G 0402	490-3222-1-ND	2	0.039
C5 C6 C9 C18 C21	CAP 100pF 50V CERAMIC X7R 0402	PCC1709CT-ND	5	0.019
C8	CAP 200pF 50V CERAMIC C0G 0402	490-3232-1-ND	1	0.051
C15	CAP 270pF 50V CERAMIC X7R 0402	PCC1714CT-ND	1	0.019
C13	CAP 3900pF 50V CERAMIC X7R 0402	PCC1728CT-ND	1	0.019
C23 C25 C26 C27 C28 C29 C31 C33 C35	CAP 0.1uF 10V CERAMIC X5R 0402	PCC2146CT-ND	9	0.026
C10 C11 C12 C17 C20 CP2 CP3	CAP CER 1.0uF 10V Y5V 0402	478-2583-1-ND	7	0.550
C16 C22 C24 C30 C32 C34 C36 C41 C42	CAP CER 10uF 6.3V 20 % X5R 0603	587-1256-1-ND	9	1.320
CIN1 CIN2 COUT1 COUT2	CAP CERAMIC 2.2UF 6.3V X5R 0402	PCC2421CT-ND	4	0.228
CP1	CAP CERM .22UF 10V Y5V 0402	478-1139-1-ND	1	0.220
R1 R4 R8	RES 100K OHM 1/10W 5 % 0603 SMD	RHM100KGCT-ND	3	0.069
R2	RES 2.2K OHM 1/10W 5 % 0603 SMD	RHM2.2KGCT-ND	1	0.069
R3	RES 1.6K OHM 1/10W 5 % 0603 SMD	RHM1.6KGCT-ND	1	0.069
R5 R10 R12	RES 10K OHM 1/10W 5 % 0603 SMD	RHM10KGCT-ND	3	0.069
R6	RES 604K OHM 1/10W 1 % 0603 SMD	RHM604KHCT-ND	1	0.076
R7 R9 R11	RES 402K OHM 1/10W 1 % 0603 SMD	311-402KHRCT-ND	3	0.077

Apéndice C

Programador JTAG

JTAG (Joint Test Action Group) es el nombre usual que se le da al standard IEEE 1149.1 para testear PCBs utilizando el método Boundary Scan. El protocolo JTAG permite conocer el estado interno de un chip que soporte este protocolo y en el caso del MSP430F149 también permite su programación.

Los pines de comunicación del Programador JTAG necesarios para programar el MSP430F149 son TDI, TDO, TCK y TMS. Se puede conectar un pin de reset para resetar el microcontrolador desde el JTAG pero no es necesario para programar. Para el MSP430F149 no hay que conectar el pin Test del protocolo JTAG, este pin se usa cuando el microcontrolador tiene menos pines y por lo tanto no tiene los pines de uso exclusivo para el JTAG (es el caso de los MSP430 de menos pines).

En la figura C.1 se muestra el esquemático que usamos para construir el Programador JTAG, este esquemático está basado en el esquemático propuesto por Olimex[34], las modificaciones hechas son para el caso particular de las necesidades del **IIEMOTE** .

C.1. Construcción del PCB

A continuación se describe la técnica utilizada para la construcción de PCB's. Esta técnica se usó en el prototipo de placa de control (segundo entregable) y en el programador JTAG. En la figuras C.4 y C.5 se muestra el Programador JTAG terminado.

Los materiales utilizados fueron los siguientes:

- Programa de edición de layout(ver H.4).
- Placa de cobre.
- Sierra.

- Esponja de aluminio.
- Alcohol isopropílico.
- Plancha.
- Impresora laser.
- Revista.
- Cloruro férrico (ácido).
- Pinza.
- Marcador permanente.
- Taladro con mecha fina.

Procedimiento:

- Primero se hace el esquemático y el layout del circuito en un programa de ruteo.
- Con una impresora laser imprimir el layout en papel satinado, por ejemplo cualquier papel de revista.
- Con la impresión medir la dimensiones en la placa de cobre y cortarla con una sierra pequeña, teniendo cuidado de no hacer saltar el cobre.
- Limpiar el trozo de placa de cobre con esponja de aluminio y alcohol isopropílico.
- Colocar el papel impreso sobre sobre el trozo de cobre, de tal manera que el papel no se corra.
- Con una plancha caliente, planchar el papel sobre la placa de cobre hasta que se noten las pistas del layout.
- Mojar el papel pegado en la placa y a la vez que se humedece, cuidadosamente y con paciencia, ir removiendo el papel de la placa.
- Una vez sacado todo el papel, lo único que queda sobre el cobre es el layout. A veces puede suceder que alguna pista no haya quedado cómo corresponde, si sucede esto retocar estas partes con un marcador permanente.
- Calentar el cloruro férrico (percloruro de hierro) a baño de maría, en un recipiente que entre el trozo de cobre.

- Una vez que el ácido este tibio, con una pinza inmune al ácido sumergir completamente la placa en el recipiente con ácido.
- Esperar 15 minutos mientras el ácido reacciona con el cobre (ir revisando la placa cada 5 minutos).
- Cuando el cobre es removido por el ácido se saca la placa del recipiente y se hace correr mucha agua sobre ella. Luego se limpia la placa con la esponja de aluminio removiendo los restos de papel que hayan quedado pegados.
- Con la placa seca y el layout de cobre a la vista, se usa un taladro pequeño para hacer las bías del PCB.
- Una vez agujereada la placa ya esta lista para soldar los componentes.
- En el caso del programador JTAG se estañaron las pistas del PCB, cómo se muestra en la figura C.5.

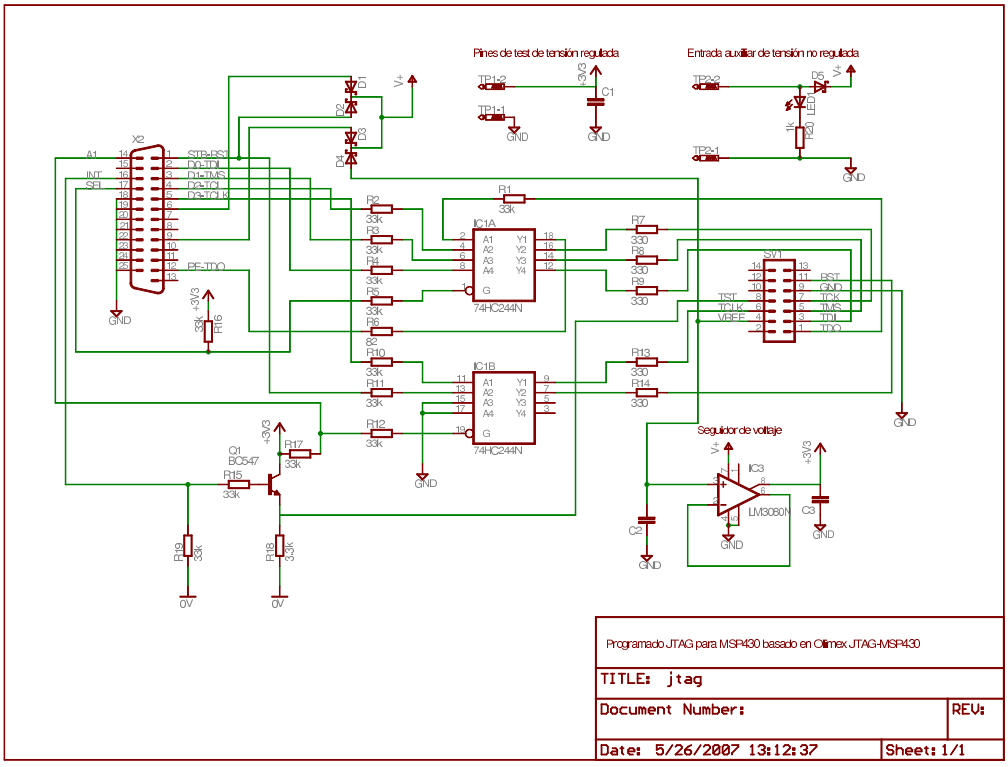

Figura C.1: Esquemático JTAG

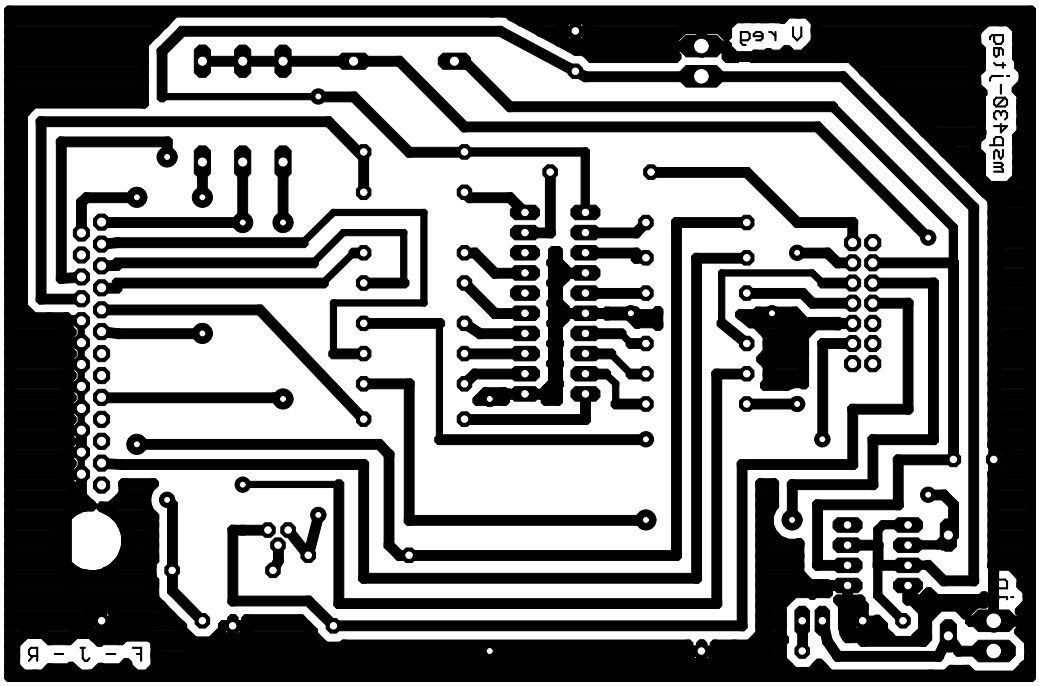

Figura C.2: Ruteo del Programador JTAG

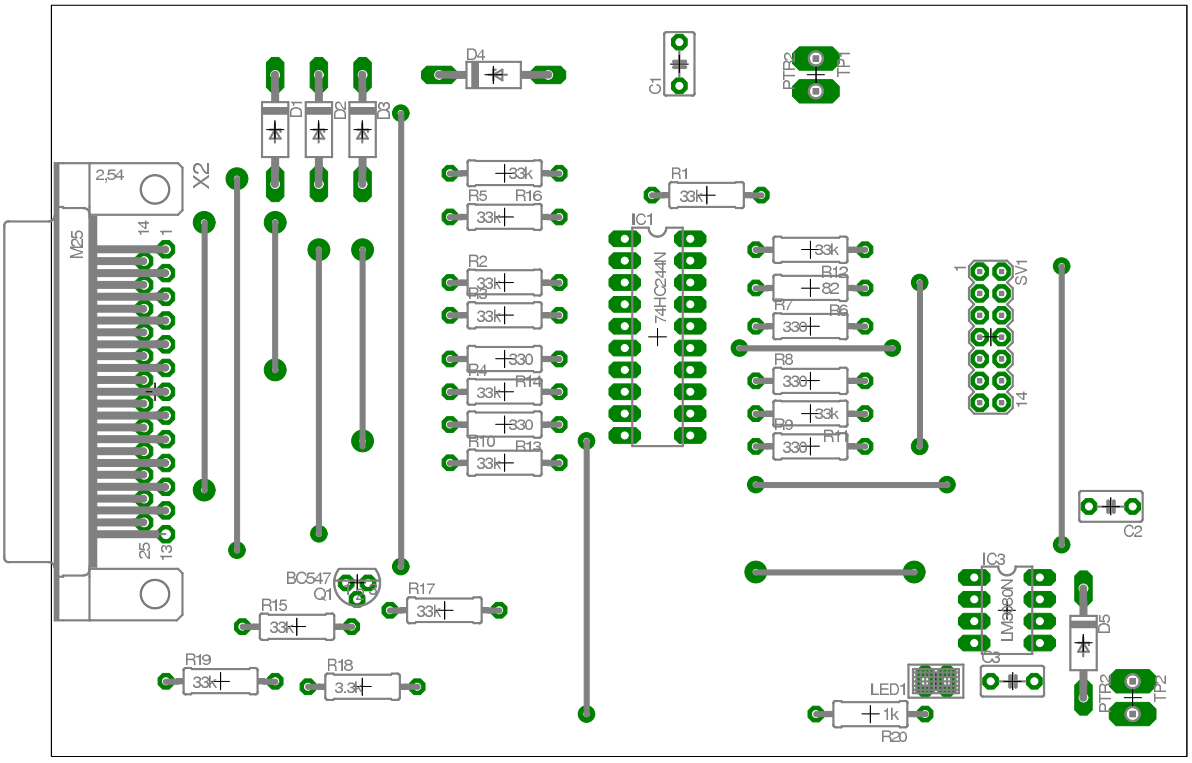

Figura C.3: Componentes del Programador JTAG

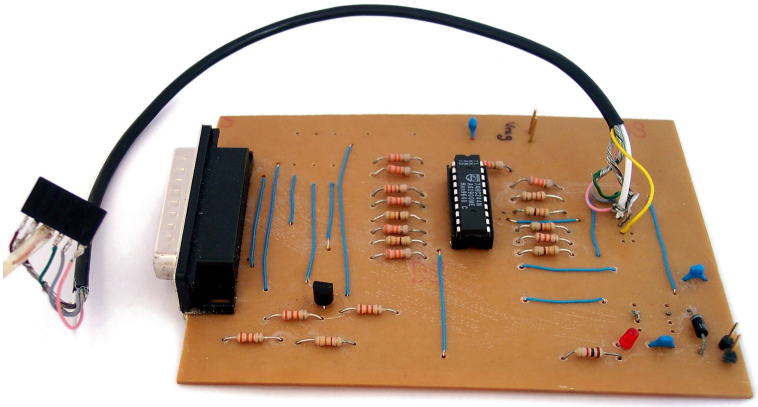

Figura C.4: Programador JTAG

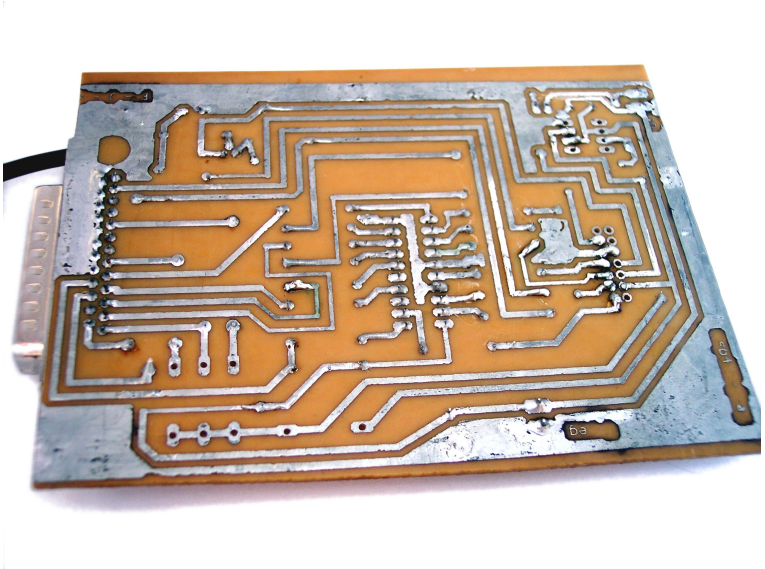

Figura C.5: Pistas del Programador JTAG

Apéndice D

Evaluación del plan de proyecto

D.1. Etapas del proyecto

Se describen las etapas principales del proyecto y se hace un estimativo de horas dedicadas en cada una de estas etapas.

D.1.1. Plan del proyecto

En esta etapa se tomó en cuenta las horas dedicadas para la realización del documento del plan del proyecto, las horas del curso de gestión de proyecto y las reuniones iniciales con los tutores.

D.1.2. Estudio de Radio Frecuencia

Dado que el perfil de los tres integrantes del proyecto es Electrónica, al inicio del proyecto no se contaba con conocimiento sobre radio frecuencia (la asignatura Antenas y Propagación no es parte del perfil Electrónica). Por esto se recomendó hacer la asignatura Circuitos Amplificadores de Radiofrecuencia que se dictó en el segundo semestre del 2005. Esta asignatura fue de gran ayuda para tener una base de RF. De todas maneras las horas dedicadas a esta asignatura NO se tomaron en cuenta en esta evaluación del proyecto ya que la asignatura daba créditos independientes al proyecto. Por lo tanto las horas contabilizadas en esta etapa son de lo estudiado de RF durante el transcurso del proyecto, más allá de las horas dedicadas a la asignatura mencionada. Por esto en la tabla D.1 hay una diferencia tan grande entre las horas planificadas y las horas realizadas.

D.1.3. Análisis

Esta etapa involucra todo lo descrito en el capítulo 2, a saber, elección del sistema a utilizar (Comblocks o transceiver), elección del transceiver, elección del microcontrolador, análisis de consumo y diseño conceptual. Esta etapa llevó un poco más de lo previsto en la planificación.

D.1.4. Prototipo de placa de control

Al comenzar el proyecto, se decidió que para el segundo entregable se construiría un prototipo de placa de control utilizando el microprocesador elegido para controlar el sistema. Si bien se llegó a diseñar el PCB, la placa no pudo ser construida debido a que no se pudo conseguir en el mercado una solución satisfactoria de fabricación y de montaje de componentes. Visto el tamaño de los componentes a utilizar sólo era viable el soldado en horno, y las tecnologías de plaza no nos aseguraban contar con la precisión necesaria en las máscaras como para lograr tal objetivo. Igualmente, se intentó el soldado en forma artesanal, sin obtener más beneficios que la mejora de nuestras habilidades en el área y la sistematización de algunos métodos artesanales de fabricación de PCBs (descritas en el apéndice C). Esta etapa demandó una importante inversión de tiempo (más de 1 mes de alta dedicación), y lamentablemente no nos ayudo a avanzar en nuestros objetivos centrales. La decisión que se tomó entonces, fue fabricar directamente la placa final y no hacer prototipo de placa de control.

D.1.5. Diseño de esquemático y PCB

Esta etapa se centró en el diseño del esquemático y el PCB descrito principalmente en el capítulo 6. Al momento de esta etapa del proyecto simultáneamente se estaba desarrollando una segunda versión del TXIIE (PCB desarrollado por el grupo de microelectrónica del IIE en adelante TXIIE 2), se decidió por razones de costos enviar a hacer a USA las dos placas a la vez y luego separarlas. En diciembre de 2006 la placa del receptor estaba pronta para fabricar, pero faltaban algunos detalles de la placa del TXIIE 2, que el equipo de microelectrónica estaba desarrollando. Esto retrasó la fabricación de la placa, la cual se envío a fines de enero, lo que nos retrasó 1 mes más. Luego por problemas en la aduana, la placa se demoró más de un mes en llegar a nuestras manos. Las horas dedicadas en esta etapa fueron similares a las estimadas en la planificación del proyecto.

D.1.6. Software

En la espera de la placa, y con las limitaciones del caso por no poder probar el software en el micro, se estudió y desarrolló el software necesario para el control del sistema. Se estudió sobre TinyOS y se comenzó

a desarrollar una plataforma para el $\mathbf{I_{IE}M_{OTE}}$. Esta etapa se describe en el capítulo 7. La carga horaria del software fue un poco menor a la planificada.

D.1.7. Armado y montaje

Esta etapa consistió en la fabricación de tres $\mathbf{I_{IE}M_{OTE}}$, tres antenas de loop, una antena de monopolo doblado y del programador JTAG. En la planificación del proyecto no se consideró el armado de 3 prototipos ni del programador JTAG, por esto la diferencia tan grande entre las horas dedicadas y las horas planificadas.

D.1.8. Pruebas y ajustes

Para esta etapa se contabilizan todos los ajustes y pruebas necesarias para que el $\mathbf{I_{IE}M_{OTE}}$ funcionara, a saber la programación del microcontrolador vía JTAG, comunicación con la PC vía SR232, comunicación entre el microcontrolador y el transceiver y comunicación RF. Además todas las pruebas de consumo, potencia y distancias que se hicieron con el $\mathbf{I_{IE}M_{OTE}}$. Esta etapa fue mal estimada en la planificación del proyecto, en parte porque estaba en dos partes (pruebas y ajustes del prototipo de prueba y del prototipo final) esto hacía no tener que hacer todas las pruebas y ajustes con un mismo prototipo sino tenerlo diseminado en dos etapas. Especialmente los ajustes de la programación del microcontrolador con JTAG ni siquiera estuvo prevista en la planificación. Por esto, como se observa en la tabla D.1, las horas dedicadas a esta etapa llevaron más del doble de lo planificado.

D.1.9. Documentación

Es la documentación de todo el proyecto y el manual de usuario del $\mathbf{I_{IE}M_{OTE}}$. La dedicación de esta etapa fue similar a la planificada en el inicio del proyecto.

D.1.10. Aplicación y presentación

Aquí se tomó en cuenta los tiempos que estimamos llevará la realización de la aplicación y la realización de la presentación del proyecto. En la planificación no se tomó en cuenta la aplicación, por esto la diferencia de horas en esta etapa.

D.2. Horas dedicadas y estimadas

En la tabla D.1 se comparan las horas estimadas de cada etapa del proyecto contra las horas planificadas.

Tabla D.1: Horas dedicadas

Etapa	**Horas dedicadas**	**Horas planificación**
Plan del proyecto	130	128
Estudio de Radio Frecuencia	70	260
Análisis	350	260
Prototipo de placa de control	300	174
Diseño de esquemático y PCB	400	388
Software	220	280
Armado y montaje	200	30
Pruebas y ajustes	550	200
Documentación	750	762
Aplicación y Presentación	150	48
Total de horas	3120	2530

En la tabla se observa claramente que las horas dedicadas al proyecto fueron más de las estimadas en la planificación, aunque se observa que la diferencia no excede el 25 %, con lo cual se podría llegar a considerar que la diferencia se encuentra dentro de un rango no tan elevado (teniendo en cuenta además la falta de experiencia del grupo en el tema).

D.3. Riesgos del proyecto

En esta sección se analizan los riesgos asumidos en la planificación del proyecto y sus impactos en el transcurso del proyecto.

D.3.1. Error en el diseño del prototipo

Hubo errores cometidos en el diseño, pero éstos no impidieron seguir adelante ya que fueron errores menores y corregibles.

D.3.2. Llegada de los componentes a destiempo

Todos los componentes llegaron a tiempo salvo la placa, la placa estuvo más de un mes en la aduana. El impacto fue alto, en este período se avanzó en el software.

D.3.3. No accesibilidad a los componentes

Se accedió a todos los componentes necesarios.

D.3.4. RRHH del grupo

Los recursos humanos del grupo cambiaron mucho durante el transcurso del proyecto, las obligaciones laborales de todos los integrantes aumentaron luego de la segunda etapa. Esto impactó fuertemente en la dedicación del proyecto.

D.3.5. RRHH de los tutores

Al ser dos tutores fue fácil la comunicación con ellos.

D.3.6. No accesibilidad a fabricantes de PCB

Se consiguió un fabricante y no hubo retrasos con la compra del PCB.

D.3.7. No accesibilidad de compras en el exterior

Se tuvo accesibilidad a las compras vía internet. También se tuvo accesibilidad para traer muestras gratis de Texas Instruments desde EEUU (las muestras gratis se envían sólo dentro de EEUU, no se envían a Uruguay).

D.3.8. Mala estimación del tiempo

Hubo diferencias entre lo estimado y lo dedicado realmente. Pero la diferencia era cubierta por el buffer propuesto. De todas maneras el impacto de este riesgo fue medio.

D.3.9. No disponibilidad de instrumental en fecha

Se complicó no tener un laboratorio fijo siempre a disposición. Hubo que coordinar para poder realizar las pruebas y ajustes. Este riesgo tuvo un impacto medio al no poder contar con el instrumental al momento de necesitarlo.

D.3.10. Fallas de hardware

Se tuvo problemas con el buffer del Programador JTAG y con uno de los $\mathbf{I_{IE}M_{OTE}}$ que no funcíonó, pero teníamos sustitutos.

D.3.11. Daños de componentes

No se nos dañó ningún componente. Solamente en la etapa de hacer el prototipo de control, pero teníamos varios microcontroladores de repuesto.

D.3.12. Dificultad para realizar medidas en etapa de testeo

Las medidas de consumo y distancia las tuvimos que hacer más de una vez, no hicimos una buena planificación hasta las últimas medidas realizadas.

D.3.13. Dificultad para conseguir módulos comerciales con el TXIIE

No se tuvo dificultad para esto, pero no se pudo realizar ninguna prueba (a la fecha de escribir esta documentación el TXIIE no estaba pronto).

D.4. Plan original del proyecto

A continuación se adjunta el plan original del proyecto, presentado el 01/11/05.

Plan del Proyecto

Sistema de recepción de RF de bajo consumo

INDICE DEL DOCUMENTO:

Figura D.1: plan de proyecto / página 1

RESUMEN

Nombre del Proyecto: Sistema de recepción de RF de bajo consumo

Integrantes:
Juan Ignacio Rivero Rivero 2843775-8 mvjuani@redfacil.com.uy
Federico Núñez Artigas 2744526-9 fna@adinet.com.uy
Ramiro Carlos Moreira Moreno 2827509-3 ramiroc@montevideo.com.uy

Cliente:
Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República

Tutores:
Rafaella Fiorelli
Fernando Silveira

Fecha prevista de finalización: 01/02/2007

Total de horas a realizar previstas por el grupo del proyecto: 2450

Fecha y descripción de los entregables intermedios:
entregable 1 15/03/2006
entregable 2 11/07/2006

DESCRIPCIÓN DEL PROYECTO

El Grupo de Microelectrónica del IIE, en el marco del proyecto PDT: "Sensores Inalámbricos Integrados de Bajo Consumo", envío a fabricación un prototipo de transmisor de bajo consumo para la banda ISM de 915MHz (en adelante TXIIE).
El proyecto consiste en diseñar un circuito que controle el chip transmisor TXIIE y un sistema de recepción. Para este sistema de recepción se evaluará en las fases iniciales del proyecto la viabilidad de implementarlo con transcievers comerciales o alternativamente con módulos configurables que hemos identificado en el mercado (cómo los suministrados por la empresa ComBlock, www.ComBlock.com).

OBJETIVO GENERAL DEL PROYECTO

En primera instancia, el objetivo general del proyecto es montar un "demostrador" de enlace de RF de corta distancia (algunos metros) y bajo consumo que permita probar las funcionalidades del TXIIE.
A fin de que la viabilidad del proyecto no esté atada al buen funcionamiento del TXIIE, se propone como camino alternativo la posibilidad de que el sistema de recepción desarrollado también pueda operar como transmisor. Esto permitirá probar el sistema de recepción con independencia del TXIIE y en este caso, el producto del proyecto sería el diseño de un enlace de RF de corta distancia con módulos de muy bajo consumo, tamaño y fácilmente reproducibles (ajustes sencillos y procedimientos de ajuste bien establecidos).
Los criterios de aceptación del proyecto son:

Figura D.2: plan de proyecto / página 2

- Consumo promedio aceptable: con transmisión de 10 bytes por segundo que permita duración mayor a 2 años con batería de 1Ah.
- Distancia mínima aceptable: 10 metros.
- Velocidad mínima de transmisión: 600 kbps.

ESPECIFICACIÓN FUNCIONAL DEL PROYECTO Y PREDISEÑO

Las especificaciones objetivo para este enlace son las siguientes:

- Potencia de transmisión programable para distancias de 50m y menores.
- Operación en los 10 canales de la banda de [906 + 2(k-1)]MHz.
- Consumo promedio: objetivo indicativo: con transmisión de 10 bytes por segundo que permita duración mayor a 2 años con batería de 1Ah.(sólo para transcivers)
- Objetivo de tamaño del módulo: menor a 4 x 4 cms. (sólo para transcivers)
- Alimentación de baterías (2V a 2.8V). (sólo para transcivers)
- Firmware modular, orientado a bajo consumo y que provea las funciones básicas de manejo del enlace y permita mostrar la correcta operación del mismo.
- Módulos fácilmente reproducibles: lista de componentes completamente especificada, ajustes sencillos y procedimientos de ajuste bien establecidos.

ESPECIFICACIÓN DE LOS OBJETIVOS ESPECÍFICOS Y LOS PRINCIPALES ENTREGABLES DEL PROYECTO--DEFINICIÓN DETALLADA DEL ALCANCE

El proyecto se divide en las en las siguientes etapas:

- *Estudio de conceptos básicos de RF*

En esta etapa se estudiarán los fundamentos básicos de Radio Frecuencia para poder pasar a la etapa de análisis.

- *Análisis*

Al concluir esta etapa se presentará el 1er entregable que consistirá en un estudio comparativo entre la implementación del circuito utilizando comblocks o utilizando transcievers comerciales. Se entregará un diseño conceptual teniendo en cuenta ambas opciones.

- *Diseño y prototipo*

En una primera instancia se armarán y probarán diferentes antenas para luego poder definir la red de adaptación de impedancias del receptor. Se realizará el diseño y armado de la placas de control del TXIIE y del receptor, así como el diseño y armado del circuito impreso del receptor (dado el caso en que no se disponga de la funcionalidad del TXIIE, también se realizará el diseño y armado del transmisor). Esta etapa culmina con la realización de testeos de los prototipos y con la presentación del 2o entregable que abarca lo realizado en esta etapa.

Figura D.3: plan de proyecto / página 3

- *Armado y optimización*

En esta etapa se implementarán las correcciones necesarias en el diseño del hardware además de la programación final del los microprocesadores de las placas de control. Esta etapa culmina con la fabricación y el testeo del producto final.

- *Elaboración final*

El proyecto concluye con la puesta a punto del sistema y la elaboración final de la documentación.

WORK BREACKDOWN STRUCTURE

No	Tarea	Horas	Inicio	Fin
1	**Curso Gestión de Proyecto**	**18**	**mar 2005-10-11**	**mié 2005-10-12**
2	**Planificación**	**110**	**mié 2005-10-12**	**mar 2005-10-25**
2.1	Especificar proyecto	30	mié 2005-10-12	vie 2005-10-14
2.2	Desglose de tareas	30	lun 2005-10-17	mié 2005-10-19
2.3	Asignación de tiempos a las tareas.	30	mié 2005-10-19	vie 2005-10-21
2.4	Documento de planificación de proyecto	20	lun 2005-10-24	mar 2005-10-25
3	**Entrega de Planificación de proyecto**	**0**	**mar 2005-11-01**	**mar 2005-11-01**
4	**Estudio de conceptos básicos de RF**	**170**	**mié 2005-12-07**	**mar 2005-12-27**
4.1	Comportamiento de componentes en RF	36	mié 2005-12-07	vie 2005-12-09
4.2	Líneas de transmisión	50	lun 2005-12-12	vie 2005-12-16
4.3	Redes de adaptación de impedancia	44	vie 2005-12-16	mié 2005-12-21
4.4	Antenas	40	jue 2005-12-22	mar 2005-12-27
5	**Análisis**	**260**	**mar 2005-12-27**	**mié 2006-02-08**
5.1	Transceivers	108	mar 2005-12-27	lun 2006-01-16
5.1.1	Búsqueda de fabricantes	24	mar 2005-12-27	jue 2005-12-29
5.1.2	Elección de chipset	84	jue 2005-12-29	lun 2006-01-16
5.1.2.1	Análisis de características	60	jue 2005-12-29	jue 2006-01-12
5.1.2.2	Análisis de disponibilidad	6	jue 2006-01-12	vie 2006-01-13
5.1.2.3	Estudio comparativo	18	vie 2006-01-13	lun 2006-01-16
5.2	Comblocks	52	mar 2006-01-17	mar 2006-01-24
5.2.1	antena--|RedAdapta|--|3005o3001 Rx - A/D AGC|--|1027|--|uP|	26	mar 2006-01-17	jue 2006-01-19
5.2.2	Otras opciones	26	jue 2006-01-19	mar 2006-01-24
5.3	Norma IEEE 802.15.4	16	mar 2006-01-24	mié 2006-02-08
5.4	Estudio comparativo de ambas opciones	24	mié 2006-01-25	vie 2006-01-27
5.5	Diseño conceptual de ambas opciones	60	vie 2006-01-27	mar 2006-02-07
6	Estudio puntuales sobre RF	90	jue 2006-01-12	jue 2006-02-23
7	**1er entregable - Diseños conceptuales**	**0**	**mié 2006-03-15**	**mié 2006-03-15**
8	Documentación v0.1	300	mié 2006-03-15	mar 2006-08-01
9	**Diseño y prototipo**	**804**	**mié 2006-03-15**	**mar 2006-07-11**
9.1	Antena	100	mié 2006-03-15	jue 2006-03-30
9.2	Pienso control del receptor	80	jue 2006-03-30	mar 2006-04-11
9.3	Red de adaptación	80	mié 2006-04-12	lun 2006-04-24
9.4	Pienso control del TXIEE	60	lun 2006-04-24	mié 2006-05-03
9.5	Definición de la plataforma de prueba	4	mié 2006-05-03	mar 2006-06-13
9.6	Pruebas con transceiver,TXIEE y uP	480	mié 2006-05-03	mar 2006-07-11
9.6.1	Diseño de PCBs	160	mié 2006-05-03	mar 2006-06-13

Figura D.4: plan de proyecto / página 4

No	Tarea	Horas	Inicio	Fin
9.6.1.1	Diseño de esquemáticos	70	mié 2006-05-03	lun 2006-05-15
9.6.1.2	Ruteo de la placas	90	mar 2006-05-30	mar 2006-06-13
9.6.2	Programación del los uP	100	lun 2006-05-15	mar 2006-05-30
9.6.3	Fabricación PCBs y componentes	130	mié 2006-06-14	mar 2006-06-27
9.6.3.1	Espera por el PCB	126	mié 2006-06-14	mar 2006-06-27
9.6.3.2	Gestiones para la fabricación	4	mié 2006-06-14	mié 2006-06-14
9.6.4	Soldado de componentes	30	mié 2006-06-28	lun 2006-07-03
9.6.5	Testeos del sistema	60	lun 2006-07-03	mar 2006-07-11
10	**2do entregable – Diseño**	**0**	**mar 2006-07-11**	**mar 2006-07-11**
11	**Armado y optimización**	**450**	**mié 2006-07-12**	**lun 2006-09-04**
11.1	Correcciones a diseño HW	40	mié 2006-07-12	mar 2006-07-18
11.2	Correcciones en esquemáticos	20	mar 2006-07-18	jue 2006-07-20
11.3	Programación del uP	80	vie 2006-07-21	mié 2006-08-02
11.4	Re-ruteo de la placa	40	mié 2006-08-02	lun 2006-08-07
11.5	Fabricación PCBs y componentes	160	mar 2006-08-08	lun 2006-08-21
11.5.1	Gestiónes para la fabricación	4	mar 2006-08-08	mar 2006-08-08
11.5.2	Espera por el PCB	156	mar 2006-08-08	lun 2006-08-21
11.6	Soldado de componentes	30	mar 2006-08-22	jue 2006-08-24
11.7	Testeos del sistema	80	jue 2006-08-24	lun 2006-09-04
12	**Elaboración final**	**328**	**lun 2006-09-04**	**mié 2006-10-11**
12.1	Corrección de documentación	180	lun 2006-09-04	lun 2006-09-25
12.2	Ajustes de programación	100	lun 2006-09-25	jue 2006-10-05
12.3	Preparación de la presentación final	48	jue 2006-10-05	mié 2006-10-11
13	**Final del Proyecto**	**0**	**mié 2006-10-11**	**mié 2006-10-11**

SUPUESTOS Y RESTRICCIONES

Supuestos:

- Disponibilidad de componentes necesarios
- Financiamiento para la compra de componentes y fabricación del PCB
- Disponibilidad de equipamiento para el testeo del PCB
- Dedicación horaria de los integrantes
- Colaboración de los tutores

Restricciones:

- Acceso al instrumental requerido
- Acceso a los componentes necesarios

ANÁLISIS DE RIESGOS

RIESGO	Probabilidad	Impacto	Acción
Error en el diseño del prototipo	Media	Alto	Recurrir a simulaciones y verificaciones exhaustivas antes de la fabricación.
Llegada de los componentes a	Baja	Medio	Planificar tareas alternativas a

Figura D.5: plan de proyecto / página 5

destiempo			realizar durante la espera.
No accesibilidad a los componentes	Muy baja	Muy alto	Aceptar el riesgo
RRHH grupo	Media	Bajo	Planificación
RRHH tutores	Baja	Medio	Coordinación con el tutor
No accesibilidad a fabricantes de PCB	Muy baja	Muy alto	Aceptar el riesgo
No accesibilidad de compras en el exterior	Baja	Alto	Buscar fuentes alternativas
Mala estimación del tiempo	Media	Medio	Utilizar el buffer
No disponibilidad de de instrumental en fecha	Media	Medio	Coordinar préstamo con suficiente anticipación
Fallas de hardware	Baja	Muy Alto	Tener hardware sustituto
Daños de componentes	Media	Alto	Tener hardware sustituto
Dificultad para realizar medidas en etapa de testeo	Media	Alto	Análisis en etapa de diseño de medidas a realizar y forma de realizarlas
Dificultad para conseguir módulos comerciales compatibles con TXIIE	Baja	Muy alto	Identificar fuentes alternativas con antelación

CRONOGRAMA DEL PROYECTO – LISTADO DE TAREAS

Del análisis del listado de tareas antes visto, ayudados por el diagrama de gantt adjunto (ver apéndice), vimos que las fechas preliminares para los distintos entregables son las siguentes:

entregable **1**	15/03/2006
entregable **2**	11/07/2006
fin del proyecto	01/02/2007

Para el entregable final incluimos un buffer de 1/3 del camino crítico. Es bueno destacar que **ninguno** de los entregables intermedios incluye un buffer.

Figura D.6: plan de proyecto / página 6

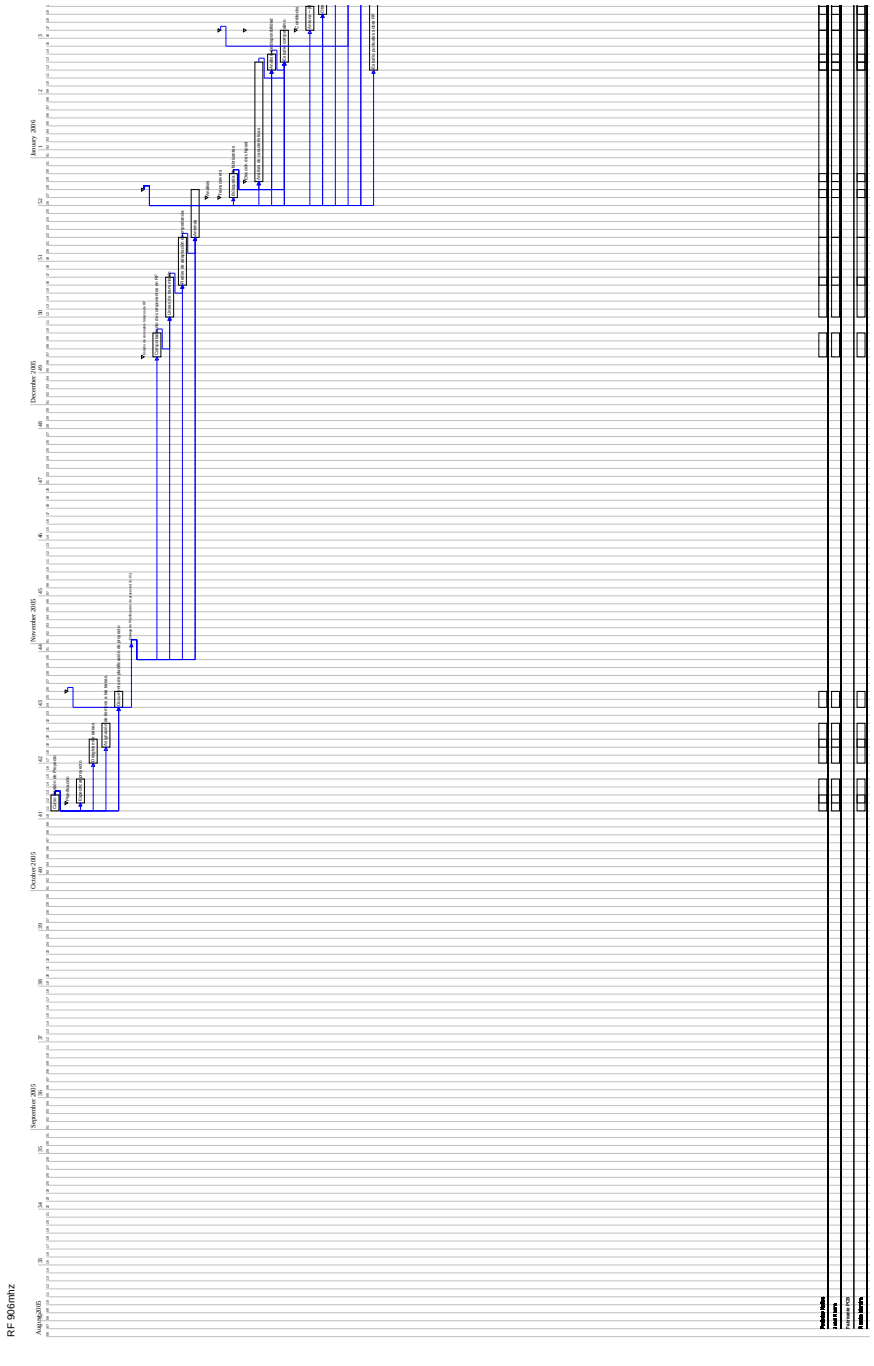

Figura D.7: plan de proyecto / página 7

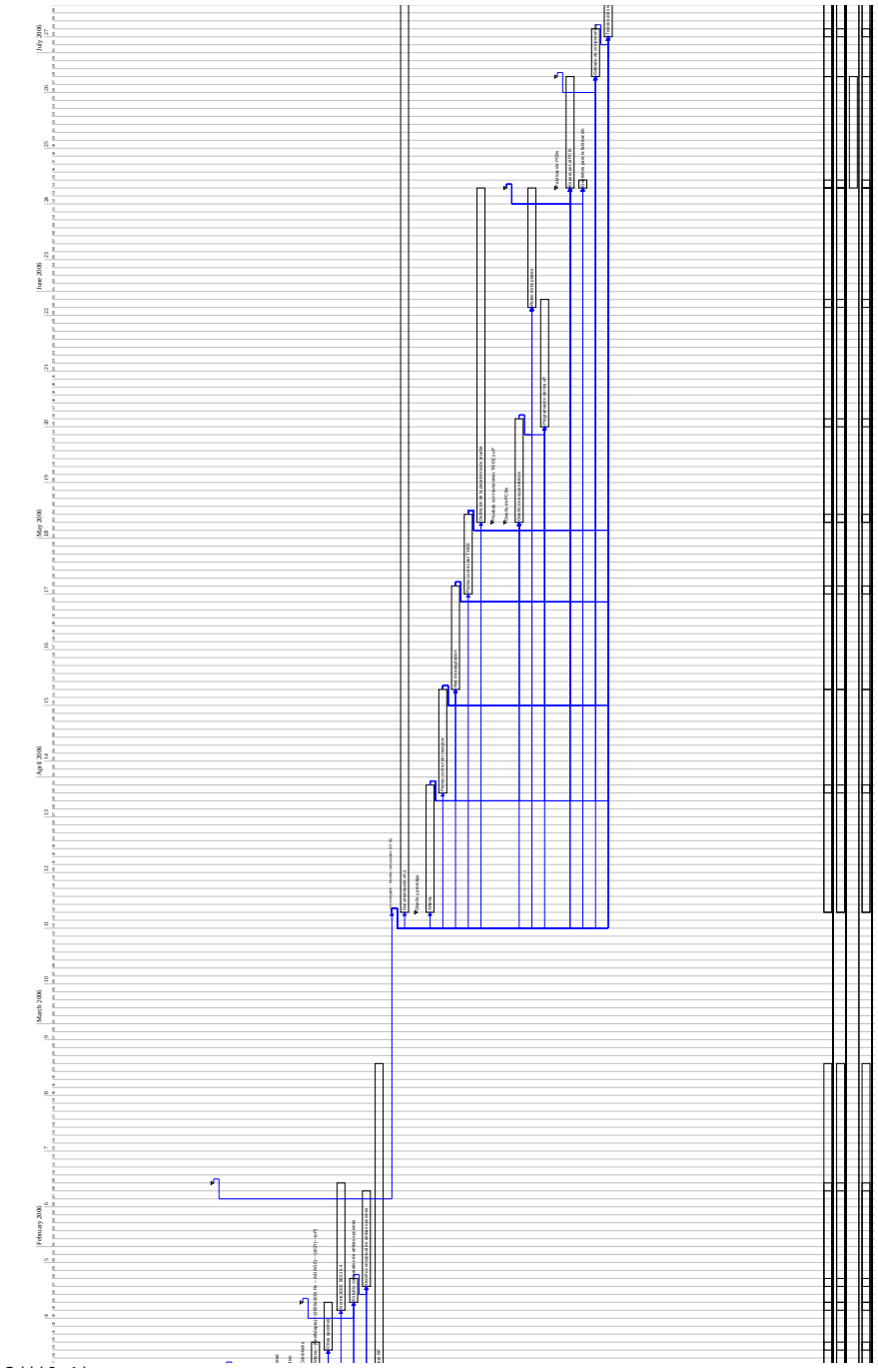

Figura D.8: plan de proyecto / página 8

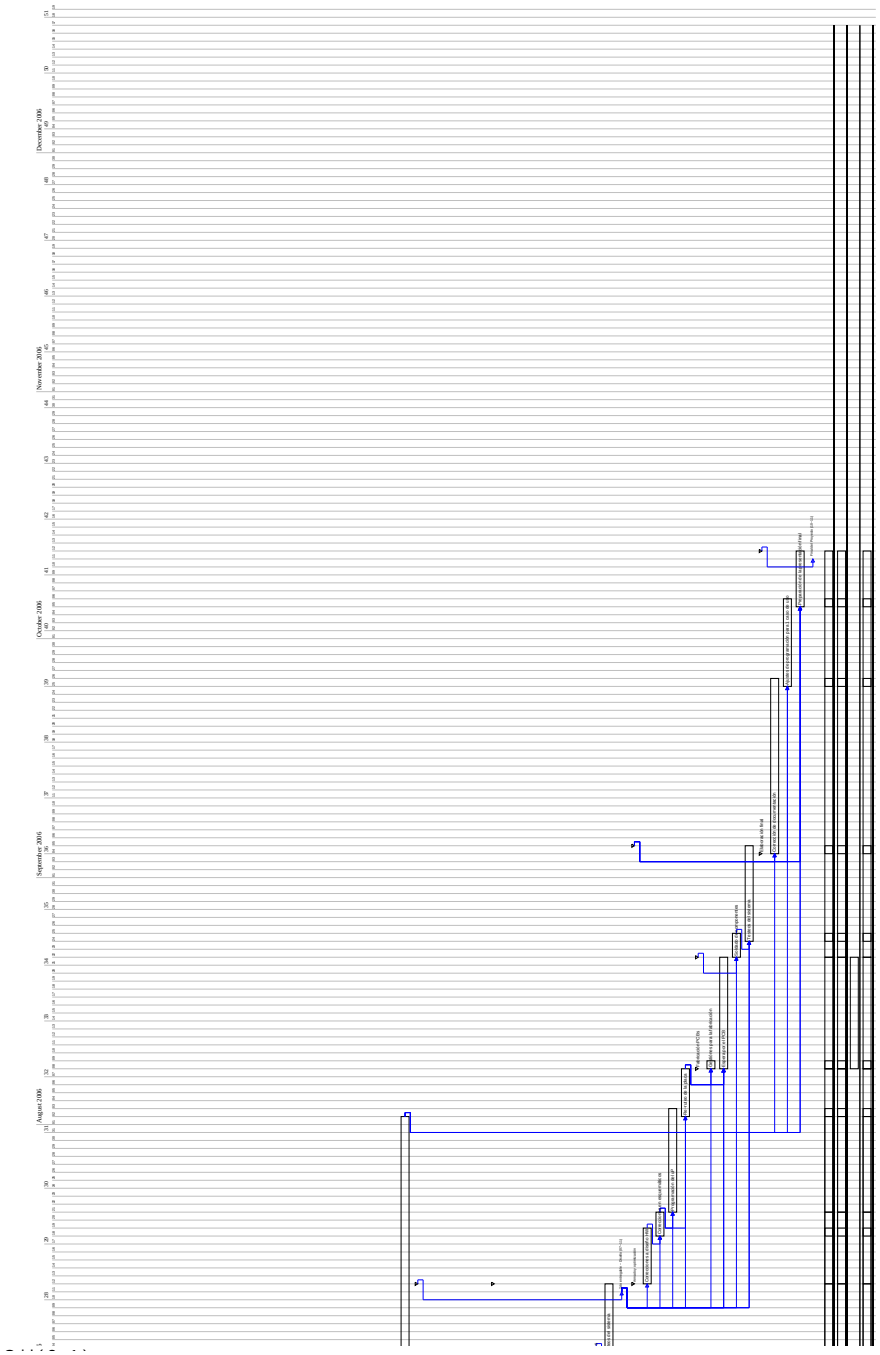

Figura D.9: plan de proyecto / página 9

Apéndice E

Fuentes del software desarrollado

E.1. Prueba de I/O

E.1.1. main.c

```
#include <io.h>

int main(void) {

  WDTCTL  = WDTPW + WDTHOLD;     /* Deshabilito watchdog */
  BCSCTL1 = XT2OFF;              /* Usar reloj de 32.768 kHz  */
  BCSCTL2 = DCOR;                /* Usar resistencia Interna */
  DCOCTL  = DCO2 + DCO1 + DCO0; /* DCO */

  P6DIR = 0xFF;  /* Configura todas las señales del puerto 6 como salida */
  P2DIR = 0xFF;  /* Configura todas las señales del puerto 2 como salida */
  P6OUT = 0x88;  /* set de los bit 3 y 7 */
  P2OUT = 0x88;  /* set de los bit 3 y 7 */

  for(;;) {                      /* bucle infinito */
    volatile int a,b,c;
    for(a=0;a<10;a++){
      for(b=0;b<20;b++){
        for(c=0;c<2000;c++){
          P6OUT ^= 0x44;         /* alterno estado de los bits 2 y 6 */
          P2OUT ^= 0x44;
        };
        P6OUT ^= 0x22;           /* alterno estado de los bits 1 y 5 */
        P2OUT ^= 0x22;
      };
      P6OUT ^= 0x11;             /* alterno estado de los bits 0 y 4 */
      P2OUT ^= 0x11;
    };

  }
```

```
}
/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

E.2. BCS

E.2.1. main.c

```
#include "globals.h"            // global defines

/*****************************************************************************
* SYSTEM INCLUDES
*****************************************************************************/
#include <stdio.h>              // Standard function declarations
#include <string.h>
#include <signal.h>

/*****************************************************************************
* USER INCLUDES
*****************************************************************************/
#include "zmd44102_reg.h"/* register defines for the ZMD44102 */
#include "leds.h"          /* functions to control the LEDs */
#include "zmd.h"           /* functions to control the ZMD44101 */
#include "uart.h"          /* functions to control the UART of the C8051F124 */
#include "cmd.h"           /* commands which can be executed by the terminal window */
#include "phy.h"
#include "smac.h"
#include "zmd_hal.h"
#include "adc.h"

/*****************************************************************************
* GLOBAL VARAIBLES
*****************************************************************************/
volatile BOOL irq_flag;    /* flag to keep track of the interrupt status */
                           /* 1 = IRQ has occured */

/*****************************************************************************
* Local prototype
*****************************************************************************/
void SYSCLK_Init (void);
void portInit (void);
void  zmdIRQ(void);

/*****************************************************************************
* Function      : SYSCLK_Init
* Description : inits system clock to
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void SYSCLK_Init (void)
```

```
{
  WDTCTL = WDTPW + WDTHOLD; /* deshabilita watchdog */
/* Frecuencia: 770kHz */
#if 0
  BCSCTL1 = XT2OFF + RSEL2;
  BCSCTL2 = DCOR;
  DCOCTL = DCO2 + DCO1 + DCO0;

#endif

/* Frecuencia: 2.38MHz */
  BCSCTL1 = XT2OFF + RSEL2 + RSEL1 + RSEL0;
  BCSCTL2 = DCOR;
  DCOCTL = DCO2 + DCO1 + DCO0;
}



void portInit (void)
{
  /*seteo todos los puertos como entrada por consejo de la hoja de datos*/
  /*con esto bajo consumo, luego quien necesite un puerto lo configura    */
  P1DIR = 0xFF;
  P2DIR = 0xFF;
  P3DIR = 0xFF;
  P4DIR = 0xFF;
  P5DIR = 0xFF;
  P6DIR = 0xFF;

  /* inicializo puertos para el resto */
  REG_INIT();
  LED_INIT();
  LED_ALL_OFF();

  /* configuro IRQ en P2.2 para wakeup */
  P2DIR &= ~0x01; /* Configuro el pin 1 como input          */
  P2IE = 0x01;    /* Habilito interrupciones en el pin 1 */
}



/*****************************************************************************
* Function      : zmdIRQ
* Description : ISR for the ZMD44102's interrupt line
* Parameters    : none
* Returns       : none
* Notes         : External Interrupt 1 (/INT1)
*****************************************************************************/
interrupt (PORT1_VECTOR) zmdIRQ(void)
{
        if (P1IFG & (1<<2))
  {
    irq_flag = 1;               /* set global variable 'irq_flag' to 1 */
  }
```

```
  P1IFG=0;
}

/*****************************************************************************
* Function      : wake_up
* Description : ISR para despertar el sistema
* Parameters    : none
* Returns       : none
* Notes         : External Interrupt 1 (/INT1)
*****************************************************************************/
interrupt (PORT2_VECTOR) wake_up(void)
{
  LPM3_EXIT;   /* salgo del modo Low Power Mode 3 */
  REG_INIT(); /* prendo reguladores */
  P2IFG=0;     /* limpio bandera de irq */
}

/*****************************************************************************
* Function      : main
* Description : main function
* Parameters    : none
* Returns       : none
* Notes         : includes the terminal window handling
*****************************************************************************/
int main (void)
{
  UBYTE input_char[2];

  SYSCLK_Init();   /* initializes system clock */
  portInit();      /* initializes crossbar, i/o lines and interrupts */
  uartInit();      /* initializes UART, used for terminal window */
  DEBUG(("\n- Llamando: zmdInitHal()..."));
  zmdInitHal();    /* initializes the ZMD's HAL */
  DEBUG(("\n- Llamando: mlmeResetRequest(TRUE)..."));
  mlmeResetRequest(TRUE);          /* init the PHY and MAC layer with its default values */

  DEBUG(("\n- eint()"));
  eint();          /* enable global interrupts */

  DEBUG(("\n- wakeup zmd"));
  zmdGpd(OFF);     /* prendiendo el zmd */

  DEBUG(("\n- reset zmd"));
  zmdReset(HOLD_TIME_3);           /* resets the ZMD44102 */

  DEBUG(("\n- loop principal"));
    while (1)
  {
    printf("\n--- BCS |Version IEEMote ---\n");
    printf("(0)  check LEDs                (1)  ZMD44102 reset\n");
    printf("(2)  GPD enable                (3)  GPD disable\n");
    printf("(4)  write to ZMD44102 reg.    (5)  read from ZMD44102 reg.\n");
    printf("(s)  set phyPIB values         (g)  get phyPIB values\n");
    printf("(I)  Init ZMD44102 (MTP)       (i)  get stored addr entr.\n");
```

```
printf("(S)  FrameSniffer              (D)  Go to Sleep\n");
printf("(R)  Hello World RX (SK)       (H)  Hello World TX (SK)  \n");
printf("(t)  transmit unslotted mode   (r)  receive unslotted mode\n");
printf("(o)  UnslottDataPoll_C         (p)  UnslottDataPoll_D\n");
printf("(c)  passive scan              (e)  energy detection scan\n");
printf("(v)  active scan               \n");
printf("(b)  auto beacon transmit      (a)  beacon tracking\n");
printf("(m)  BcTrafficCoord_d2c (C)    (n)  BcTrafficDev_d2c (D)\n");
printf("(G)  BcCoord_d2c_gts (C)       (T)  BcDev_d2c_Gts (D)\n");
printf("(X)  RF Tx test pn-seq         (x)  RF Tx test cont. carr.\n");
printf("(Y)  Rx test alcance           (U)  Tx test alcance .\n");
printf(">");
gets(input_char, 2);
DEBUG(("\n- se va a correr: %c \n", input_char[0]));

printf("\n--------------------------------------------------------------\n");
switch (input_char[0])
{
  case '0': ledTest(); break;
  case '1': zmdReset(HOLD_TIME_1); break;
  case '2':   zmdGpd(ENABLE);
      printf("Global power down enabled\n");
      break;
  case '3':   zmdGpd(DISABLE);
      printf("Global power down disbaled\n");
      break;
  case '4': cmdWriteReg(); break;
  case '5': cmdReadReg(); break;
  case 'S': cmdSniff(); break;
  case 'Y': cmdRxAlcance(); break;
  case 'U': cmdTxAlcance(); break;
  case 'I': cmdZmdInit(); break;
  case 'R': cmdRxHello(); break;
  case 'H': cmdTxHello(); break;
  case 's': cmdSetPibValue(); break;
  case 'g': cmdGetPibValue(); break;
  case 'i': cmdGetAddrEntries(); break;
  case 't': cmdUnslottedTx(); break;
  case 'D': cmdSleepMode(); break;
  case 'r': cmdUnslottedRx(); break;
  case 'o': cmdUnslottDataPoll_C(); break;
  case 'p': cmdUnslottDataPoll_D(); break;
  case 'c': cmdPassiveScan(); break;
  case 'e': cmdEnergyDetectScan(); break;
  case 'v': cmdActiveScan(); break;
  case 'b': cmdAutoBcTx(); break;
  case 'a': cmdBcTrack(); break;
  case 'm': cmdBcTrafficCoord_d2c(); break;
  case 'n': cmdBcTrafficDevice_d2c(); break;
  case 'G': cmdBcTrafficCoord_d2c_Gts(); break;
  case 'T': cmdBcTrafficDevice_d2c_Gts(); break;
  case 'X': cmdRfTxTestPnSeq(); break;
  case 'x': cmdRfTxTestContCarr(); break;
  default: break;
```

```
    }
  }
  return 0;
}
/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

E.2.2. cmd.h

```
/*****************************************************************************
* PREVENT DOUBLE-INCLUSION
*****************************************************************************/
#ifndef _CMD_H_
#define _CMD_H_


/*****************************************************************************
* Function      : cmdWriteReg
* Description : writes a value to a ZMD44102's register
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdWriteReg (void);


/*****************************************************************************
* Function      : cmdReadReg
* Description : reads a register value from the ZMD44102
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdReadReg (void);


/*****************************************************************************
* Function      : cmdSniff
* Description : starts a packet sniffer
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdSniff (void);


/*****************************************************************************
* Function      : cmdSetPibValue
* Description : sets values to the phyPIB
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
```

```
void cmdSetPibValue (void);


/*****************************************************************************
* Function      : cmdGetPibValue
* Description : gets values from the phyPIB
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdGetPibValue (void);


/*****************************************************************************
* Function      : cmdGetAddrEntries
* Description : gets all address entries from the trx
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdGetAddrEntries (void);


void cmdZmdInit (void);

void cmdTxHello (void);


/*****************************************************************************
* Function      : cmdUnslottedTx
* Description : handles unslotted data transmission
* Parameters   : none
* Returns       : none
* Notes         : you can choose between w/ or w/o ack
*****************************************************************************/

void cmdUnslottedTx (void);



/*****************************************************************************
* Function      :
* Description :
* Parameters   : none
* Returns       : none
* Notes         : you can choose between w/ or w/o ack
*****************************************************************************/


/*****************************************************************************
* Function      : cmdUnslottedTx
* Description : handles unslotted data transmission
* Parameters   : none
* Returns       : none
```

```
* Notes          : you can choose between w/ or w/o ack
*****************************************************************************/
void cmdTxAlcance (void);

/*****************************************************************************
 * Function      : cmdRxAlcance
 * Description : Recibe el paquete de 10 byte que envia cmdTxAlcance, verifica
 *                   que lo recibido sea igual a lo enviado y reporta los errores
 *                   que encuentra por consola, reportando tambien RSSI, LQI
 * Parameters    : none
 * Returns       : none
 * Notes         :
 *****************************************************************************/
void cmdRxAlcance (void);



/*****************************************************************************
* Function      : cmdUnslottedRx
* Description : handles unslotted data receiving
* Parameters    : none
* Returns       : none
* Notes         : w/ ack handling
*****************************************************************************/
void cmdUnslottedRx (void);

void cmdUnslottDataPoll_C (void);

void cmdUnslottDataPoll_D (void);



/*****************************************************************************
* Function      : cmdBcTrack
* Description : controls the beacon tracking
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdPassiveScan (void);


/*****************************************************************************
* Function      : cmdBcTrack
* Description : controls the beacon tracking
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdEnergyDetectScan (void);

void cmdActiveScan (void);
```

```
/*****************************************************************************
* Function      : cmdAutoBcTx
* Description : controls the auto beacon transmission
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdAutoBcTx (void);


/*****************************************************************************
* Function      : cmdBcTrack
* Description : controls the beacon tracking
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdBcTrack (void);



/*****************************************************************************
* Function      : cmdBcTrafficCoord_d2c
* Description : coordinator communicates with device using beacon based communication
*                this function is used by the coordinator to receive data
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdBcTrafficDevice_d2c (void);

void cmdBcTrafficCoord_d2c (void);



void cmdBcTrafficDevice_d2c_Gts (void);

void cmdBcTrafficCoord_d2c_Gts (void);


/*****************************************************************************
* Function      : cmdTSic
* Description : get the temperature value from the TSic
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdTSic (void);


/*****************************************************************************
* Function      : cmdPowerSupplyVoltage
* Description : get power supply voltage
```

```
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdPowerSupplyVoltage (void);


/*****************************************************************************
* Function     : cmdRfTxTestPnSeq
* Description : set the trx to test mode "pn-sequence"
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdRfTxTestPnSeq (void);


/*****************************************************************************
* Function     : cmdRfTxTestContCarr
* Description : set the trx to test mode continuous carrier
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdRfTxTestContCarr (void);



#endif  /* end of 'prevent double inclusion' */
```

E.2.3. cmd.c

```
/*****************************************************************************
* SYSTEM_INCLUDES
*****************************************************************************/
#include <stdio.h>              // Standard function declarations
#include <string.h>
#include <stdlib.h>


/*****************************************************************************
* USER_INCLUDES
*****************************************************************************/
#include "globals.h"            // global defines
#include "zmd_hal.h"
#include "smac.h"
#include "zmd.h"
#include "leds.h"
#include "zmd44102_reg.h"
#include "zmd_beacon_mode.h"
```

```
#include "zmd_unslotted.h"
#include "zmd_scan.h"
#include "adc.h"
#include "uart.h"
#include "cmd.h"


extern MAC_S_PIB macPIB; // see 'smac.c'

/*****************************************************************************
 * * USER_DEFINE
 * ****************************************************************************/
/* utilizados para test alcance */
#define SIZEOF_PAQUETE 10
#define BYTEPAQUETEALCANCE_0 0x32
#define BYTEPAQUETEALCANCE_1 0x05
#define BYTEPAQUETEALCANCE_2 0x33
#define BYTEPAQUETEALCANCE_3 0xA3
#define BYTEPAQUETEALCANCE_4 0xFF
#define BYTEPAQUETEALCANCE_5 0x1F
#define BYTEPAQUETEALCANCE_6 0xD1
#define BYTEPAQUETEALCANCE_7 0xE0
#define BYTEPAQUETEALCANCE_8 0x05
#define BYTEPAQUETEALCANCE_9 0x91
#define BOTON_1_INIT() P6DIR &= ~(1 << 1);          /* Init of Port Data-Direction Reg (Out=1 /
Inp=0) */
#define BOTON_2_INIT() P6DIR &= ~(1 << 2);          /* Init of Port Data-Direction Reg (Out=1 /
Inp=0) */
#define BOTON_1() (~P6IN & (1 << 1))
#define BOTON_2() (~P6IN & (1 << 2))
#define BOTONES_INIT() BOTON_1_INIT(); BOTON_2_INIT()
#define LED_ON LED_4_OFF
#define LED_OFF LED_4_ON
#define LED_TOGGLE LED_4_TOGGLE



/*****************************************************************************
* Function      : cmdWriteReg
* Description : writes a value to a ZMD44102's register
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdWriteReg (void)
{
  unsigned int value;
  unsigned char vval;
  unsigned char vreg;
  char input_str[6];

  printf("\nwrite:\n");
  printf("reg (hex,2) >");
  gets(input_str, 3);
```

```
  value = atoh(input_str);
  vreg = (unsigned char)value;
  printf("reg (hex,2) >");
  gets(input_str, 3);
  value = atoh(input_str);
  vval = (unsigned char)value;
  zmdWriteReg(vreg, vval);
  printf("\ndone\n");
}

/*****************************************************************************
* Function      : cmdSleepMode
* Description : lo mandamos a dormir
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdSleepMode (void)
{
  SLEEP_REG25(); // Escribo 0 en enable
  SLEEP_REG33(); // Escribo 0 en /SNOZZE
  LPM3;

  WAKE_REG33(); // ESCRIBO 1 en enable
  WAKE_REG25(); // ESCRIBO 1 en /SNOZZE
  zmdGpd(DISABLE);

  printf("\nnos prendemos\n");
}

/*****************************************************************************
* Function      : cmdReadReg
* Description : reads a register value from the ZMD44102
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdReadReg (void)
{
  unsigned int value;
  unsigned char vval;
  unsigned char vreg;
  char input_str[6];

  printf("\nread:\n");
  printf("reg (hex,2) >");
  gets(input_str, 3);
  value = atoh(input_str);
  vreg = (unsigned char)value;
  vval = zmdReadReg(vreg);
  printf("\nr(%02X)=%02X\ndone\n", vreg, vval);
}
```

```
/*****************************************************************************
* Function      : cmdZmdInit
* Description : inits the trx with its recommended register settings
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdZmdInit (void)
{
  printf("init ZMD44102 ");
  zmdInit();
  printf("done\n");
}

/*****************************************************************************
* Function      : cmdTxHello
* Description : starts a packet sniffer
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdTxHello (void)
{
  printf("\n Inicio // rutina Tx Hello World del StarterKit");
  zmdReset(HOLD_TIME_1);
  zmdWriteReg(MTP_CTRL, 0x09);                /* comienza programacion: ver zmd datasheet p.118 */
  while (zmdReadReg(MTP_CTRL) != 0x0C); /* ver zmd datasheet p.118 */
  printf("\n Arrancamos a escribir, 0x37=00");
  zmdWriteReg(0x37, 0x00); /* escribe 0x00 el registro MTPcontrol (carga en el registro mhrSrcAddr64Tx
el identificador IEEE de 64 bits EUI-64 asignado_de_fabrica */
  zmdWriteReg(0x80,0x48); /* escibe 0x48 en la FIFO de transmision */
  zmdWriteReg(0x80,0x65); /* escibe 0x65 en la FIFO de transmision */
  zmdWriteReg(0x80,0x6C); /* escibe 0x6C en la FIFO de transmision */
  zmdWriteReg(0x80,0x6C); /* escibe 0x6C en la FIFO de transmision */
  zmdWriteReg(0x80,0x6F); /* escibe 0x6F en la FIFO de transmision */
  zmdWriteReg(0x80,0x20); /* escibe 0x20 en la FIFO de transmision */
  zmdWriteReg(0x80,0x77); /* escibe 0x77 en la FIFO de transmision */
  zmdWriteReg(0x80,0x6F); /* escibe 0x6F en la FIFO de transmision */
  zmdWriteReg(0x80,0x72); /* escibe 0x72 en la FIFO de transmision */
  zmdWriteReg(0x80,0x6C); /* escibe 0x6C en la FIFO de transmision */
  zmdWriteReg(0x80,0x64); /* escibe 0x64 en la FIFO de transmision */
  zmdWriteReg(0x60, 0x0B); /* escribe un largo_de_trama = 11 */
  zmdWriteReg(0x62, 0xC0); /* escribe 0xC0 en el byte mas alto_de_la trama de control (bits[15:8]) */
  zmdWriteReg(0x61, 0x21); /* escribe 0x21 en el byte mas bajo_de_la trama de control (bits[7:0]) */
  zmdWriteReg(0x00, 0x01); /* define transmision en el canal 1 (906 MHZ) */
  zmdWriteReg(0x71, 0x12); /* le asigna 0x12 al campo_de_identificacion del encabezado_de_fuente MAC
para una trama de transmision */
  zmdWriteReg(0x70, 0x34); /* le asigna 0x34 al campo_de_identificacion del encabezado_de_fuente MAC
para una trama de transmision */
  zmdWriteReg(0xA0, 0x03); /* le decimos que arranque, que transmita lo que quiera */

  printf("\n FIN // rutina Tx Hello World del StarterKit");
}
/*****************************************************************************
```

```
* Function      : cmdTxHello
* Description : starts a packet sniffer
* Parameters   : none
* Returns      : none
* Notes        : none
*******************************************************************************/
void cmdRxHello (void)
{
  printf("\n Inicio // rutina Rx Hello World del StarterKit");
  zmdReset(HOLD_TIME_1);
  zmdWriteReg(MTP_CTRL, 0x09);                /* start programming */
  while (zmdReadReg(MTP_CTRL) != 0x0C); // Fede: zmd datasheet p.118
  zmdWriteReg(0x37, 0x00);

  printf("\n Arrancamos a escribir registros");
  zmdWriteReg(0x00, 0x01);
  zmdWriteReg(0x08, 0x62);
  zmdWriteReg(0xB6, 0x01);
  zmdWriteReg(0xA0, 0x04);

  printf("\n FIN // rutina Tx Hello World del StarterKit");
}

/*******************************************************************************
* Function      : cmdSniff
* Description : starts a packet sniffer
* Parameters   : none
* Returns      : none
* Notes        : none
*******************************************************************************/
void cmdSniff (void)
{
  zmdSniffer();
}


/*******************************************************************************
* Function      : cmdUnslottedTx
* Description : handles unslotted data transmission
* Parameters   : none
* Returns      : none
* Notes        : you can choose between w/ or w/o ack
*******************************************************************************/
void cmdUnslottedTx (void)
{
  unsigned int value;
  char input_str[6];
  unsigned char ack;
  unsigned char channel;
  unsigned char tx_data[125];
  unsigned char payload_length = 0;

  printf("transmit in unslotted mode\n\n");
  printf("Please enter hex ('00'...'0a') for channel >");
```

```
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);
  printf("\nwith acknowledge? ('01'-yes, '00'-no) >");
  gets(input_str, 2);
  value = atoh(input_str);
  //ack = (unsigned char)(value >> 8);
  ack = (unsigned char)(value);
  printf("\nenter text, transmit with 'ENTER' > ");

  uartGets(tx_data, 124);
#if 0
        // Esto no es necesario con nuestra rutina de UART
  for (i = 0; i < 124; i++)
  {
    character = getchar();
    if (character == 0x0A)
      break;
    tx_data[i] = character;
    payload_length++;
  }
#endif
  payload_length = strlen(tx_data);
  if (payload_length > 0)
  {
    //zmdUnslottedTx(channel, ack, tx_data, payload_length);
    plmeSetRequest(phyCurrentChannel, &channel);
    printf("\nvamos a enviar: \"%s\"",tx_data);
    zmdUnslottedTx(ack, tx_data, payload_length, 0x0102, 0x0304, AM_SHORT_ADDR);
  }
}


/*****************************************************************************
* Function      : cmdUnslottedRx
* Description : handles unslotted data receiving
* Parameters    : none
* Returns       : none
* Notes         : w/ ack handling
*****************************************************************************/
void cmdUnslottedRx (void)
{
  char input_str[6];
  UINT8 channel;
  UBYTE data_buffer[aMaxPHYPacketSize];
  UBYTE i;
  UINT16 lqi_16;
  UINT8 lqi_8, rssi;

  printf("receive in Unslotted mode\n\n");
  printf("Please enter hex ('00'...'0a') for channel >");
  gets(input_str, 2);
  channel = atoh(input_str);
  printf("\nRecibiendo en canal %02X... \n",channel);
```

```
  plmeSetRequest(phyCurrentChannel, &channel);
  while (1)
  {
    if (RX_FAILED == zmdUnslottedRx((UBYTE*) &data_buffer, TRUE)) // MAC_STATUS_ENUM zmdUnslRx (UBYTE channel, UBYTE *data_buffer, BOOL auto_ack)
      break;
    else
    {
      printf("rx packet: ");
      for (i = 0; i < data_buffer[0]+3; i++) // LQI + length
        printf("%02X ", data_buffer[i]);
      lqi_16 = (UINT16)data_buffer[data_buffer[0]+1] |((UINT16)data_buffer[data_buffer[0]+2] << 8);
      lqi_8 = getLqiValue(lqi_16);
      rssi = getRssiValue();
      printf(", LQI_16 %u, LQI_8 %u, RSSI -%u dBm\n", lqi_16, lqi_8, rssi);
    }
  }
}



/*****************************************************************************
* Function      : cmdUnslottedTx
* Description : handles unslotted data transmission
* Parameters    : none
* Returns       : none
* Notes         : you can choose between w/ or w/o ack
*****************************************************************************/
void cmdTxAlcance (void)
{
  UBYTE potencia;
  UINT16 secuencia;
  char potencia_txt[3];
  UBYTE paquete[SIZEOF_PAQUETE];

  printf("\nPor favor ingrese la potencia a la cual desea transmitir (0:0dBm; 1:-16dBm; 2:-20dBm; 3:-24dBm)");
  gets(potencia_txt, 2);
  potencia = atoi(potencia_txt);
  if (potencia & ~0x03){
    printf("\n La potencia ingresada no es correcta, saliendo...");
    return ;
  }

  BOTONES_INIT();

  zmdInitForSendCustomPackage();

  paquete[0] = BYTEPAQUETEALCANCE_0;
  paquete[1] = BYTEPAQUETEALCANCE_1;
  paquete[2] = BYTEPAQUETEALCANCE_2;
  paquete[3] = BYTEPAQUETEALCANCE_3;
  paquete[4] = BYTEPAQUETEALCANCE_4;
```

```
paquete[5] = BYTEPAQUETEALCANCE_5;
paquete[6] = BYTEPAQUETEALCANCE_6;
paquete[7] = BYTEPAQUETEALCANCE_7;
paquete[8] = BYTEPAQUETEALCANCE_8;
paquete[9] = BYTEPAQUETEALCANCE_9;

printf("\n Se inicia la rutina a potencia%u (periferico conectado?)",potencia);
printf("\n->Boton del periferico envia rafaga de 10000 paquetes");
printf("\n->Una tecla sale de la rutina");

DEBUG(("\n- loop principal"));
while (!inkey())
{
  LED_OFF();
  if (BOTON_1())
  {
    LED_ON();
    wait_ms(200);
    if (BOTON_1())
    {
      DEBUG(("\n- Boton 1 - Apretado"));
      while (BOTON_1())
      {
        LED_TOGGLE();
        wait_ms(100);
      }
    }
  } else if (BOTON_2())
  {
    LED_ON();
    wait_ms(200);
    if (BOTON_2())
    {
      DEBUG(("\n- Boton 2 - Apretado"));
      while (BOTON_2())
      {
        LED_TOGGLE();
        wait_ms(100);
      }
      DEBUG(("\n- Boton 2 - Disparo"));
      for (secuencia=0;secuencia<6000;secuencia++)
      {
        if (BOTON_2())
          if (BOTON_2()){
            LED_OFF();
            while (BOTON_2());
            break;
          }
        SendCustomPackage(SIZEOF_PAQUETE, potencia, paquete);
        LED_TOGGLE();
        wait_ms(60);
      }
    }
  }
```

```
  }
  LED_INIT();


}

/*****************************************************************************
 * Function        : cmdRxAlcance
 * Description : Recibe el paquete de 10 byte que envia cmdTxAlcance, verifica
 *                       que lo recibido sea igual a lo enviado y reporta los errores
 *                       que encuentra por consola, reportando tambien RSSI, LQI
 * Parameters : none
 * Returns         : none
 * Notes             :
 *****************************************************************************/
void cmdRxAlcance (void)
{
  char leyenda[11];
  UBYTE buffer[aMaxPHYPacketSize];
  UBYTE paquete_tipo[SIZEOF_PAQUETE];
  UBYTE i;
  UINT16 errores_de_paquete, errores_de_bit, errores_de_bit_previo, numero_de_paquete;
  UINT16 lqi_16;
  UINT8 lqi_8, rssi;
  UINT8 irq_reason;


  paquete_tipo[0] = BYTEPAQUETEALCANCE_0;
  paquete_tipo[1] = BYTEPAQUETEALCANCE_1;
  paquete_tipo[2] = BYTEPAQUETEALCANCE_2;
  paquete_tipo[3] = BYTEPAQUETEALCANCE_3;
  paquete_tipo[4] = BYTEPAQUETEALCANCE_4;
  paquete_tipo[5] = BYTEPAQUETEALCANCE_5;
  paquete_tipo[6] = BYTEPAQUETEALCANCE_6;
  paquete_tipo[7] = BYTEPAQUETEALCANCE_7;
  paquete_tipo[8] = BYTEPAQUETEALCANCE_8;
  paquete_tipo[9] = BYTEPAQUETEALCANCE_9;

  printf("\nPor favor ingrese la leyenda de esta medida (hasta 10 caracteres)");
  gets(leyenda, 10);

  zmdReset(HOLD_TIME_1);
  /* comienza programacion: ver zmd datasheet p.118 */
  zmdWriteReg(MTP_CTRL, 0x09);
  while (zmdReadReg(MTP_CTRL) != 0x0C); /* ver zmd datasheet p.118 */
  zmdWriteReg(0x37, 0x00);


  /* set up registers for best performance results */
  zmdWriteReg(CLIP_CHK_AGC_LVL_TH, 0x50);
  zmdWriteReg(AGC_LVL_INIT, 0x90);
  zmdWriteReg(ACQ_PEAK_TH_SC_TRIM, 0xC3);
  zmdWriteReg(RPATCW, 0x02);
  zmdWriteReg(RSTX, 0x0B);
```

```
  zmdWriteReg(EDO_TRIM, 0x87);
  zmdWriteReg(ACQ_PEAK_TH_1, 0x03);
  zmdWriteReg(ACQ_PEAK_TH_0, 0xA0);
  zmdWriteReg(CLIP_CNT_STARTSMP, 0xF0);

  /* switch clk output off, lower power consumtion */
  zmdWriteReg(CLK_OUT_CONFIG, 0x00);
  zmdWriteReg(0x00, 0x01); /* configuro canal 1 */
  zmdWriteReg(MAC_FILT_CONFIG, 0x00); /* Configuro para que no filtre nungun paquete */

  /* disable auto ack, store LQI, continue rx */
  zmdWriteReg(MAC_RX_CONFIG, MC_FIFO_STORE_LQI |MC_FIFO_STORE_ACK |MC_CONT_RX);
  zmdWriteReg(MAC_CTRL, MC_RX_ON);/* switch the receiver on */
  printf("\nRecibiendo en canal 0x01 para %s ... \n",leyenda);
  printf("\n leyenda; # paquete; largo; filtro; lqi_16; lqi_8; RSSI (dBm); paquete\n");
  printf("\n%s ; p:; l:; ep:; eb:; f:; L16:; L8:; RSSI:;fs:");
  numero_de_paquete = 0;
  errores_de_paquete = 0;
  errores_de_bit = 0;
  while (1)
  {
    irq_reason = zmdWaitForIRQ(); /* read the reason of the IRQ */
    if (irq_reason & IRQ_RX)                    /* RX_IRQ reason */
    {
      while (zmdGetRxPacket((UBYTE*)&buffer))
      {
        numero_de_paquete++;
        errores_de_bit_previo = errores_de_bit;
        for (i = 4; i < buffer[0]+1; i++)
          if (buffer[i]^paquete_tipo[i-4]){
            BYTE temp = buffer[i]^paquete_tipo[i-4];
            if (temp  &  0x01) errores_de_bit++;
            if (temp  &  0x02) errores_de_bit++;
            if (temp  &  0x04) errores_de_bit++;
            if (temp  &  0x08) errores_de_bit++;
            if (temp  &  0x10) errores_de_bit++;
            if (temp  &  0x20) errores_de_bit++;
            if (temp  &  0x40) errores_de_bit++;
            if (temp  &  0x80) errores_de_bit++;
          }
        if ((errores_de_bit!=errores_de_bit_previo)) errores_de_paquete++;
                  lqi_16 = buffer[buffer[0]+1] |((UINT16)buffer[buffer[0]+2] << 8);
         lqi_8 = getLqiValue(lqi_16);
         rssi = getRssiValue();
         printf("\n%s;%u;%u;%u;%u;%02X;%u;%u;-%u;%02X",
            leyenda, numero_de_paquete, buffer[0],errores_de_paquete,errores_de_bit,zmdReadReg(MAC_FILT_
lqi_16, lqi_8, rssi, zmdReadReg(MAC_FIFO_STATUS));
      }
    }
    else if (irq_reason == 0x00) /* el usuario toco una tecla */
    {
      zmdReset(HOLD_TIME_1); /* reset to stop receiving */
      printf("\nSaliendo_de_\"%s\";p:%u; errores_de_paquete:%u ; errores_de_bit:%u; ", leyenda,
          numero_de_paquete, errores_de_paquete, errores_de_bit);
```

```
      break;
    }
    else
    {
      printf("\n%s;%u;;;;;;;; irq_reason 0x%02X",leyenda,numero_de_paquete, irq_reason);
    }
  }
}



/*****************************************************************************
* Function      : cmdUnslottDataPoll_C
* Description : coordinator waits for device's polling
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdUnslottDataPoll_C (void)
{
  char input_str[6];
  UINT8 channel;
  UBYTE tx_data[aMaxPHYPacketSize];
  UINT8 i, payload_length;
  UINT8 character;

  printf("get polled in unslotted mode (Coor)\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  channel = atoh(input_str);
  printf("\n");
  payload_length = 0;
  printf("\nenter text, transmit with 'ENTER' > ");
  for (i = 0; i < 124; i++)
  {
    character = getchar();
    if (character == 0x0A)
      break;
    tx_data[i] = character;
    payload_length++;
  }
  if (payload_length > 0)
  {
    plmeSetRequest(phyCurrentChannel, &channel);
    //zmdUnslottedRx(channel);
    while (1)
    {
      if (RX_FAILED == zmdUnslottedDataPoll_C(tx_data, payload_length))
        break;
    }
  }
}
```

```
/*****************************************************************************
* Function      : cmdUnslottDataPoll_D
* Description : device polls for data at the coordinator
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdUnslottDataPoll_D (void)
{
  unsigned int value;
  char input_str[6];
  UINT8 i;
  unsigned char channel;
  UBYTE rx_data[aMaxPHYPacketSize];
  UINT16 dest_addr16, dest_pan_id;

  printf("poll in unslotted mode (Dev)\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  value = atoh(input_str);
  //channel = (unsigned char)(value >> 8);
  channel = (unsigned char)(value);
  printf("\n");
  plmeSetRequest(phyCurrentChannel, &channel);
  dest_pan_id = 0x1122;
  dest_addr16 = 0x3344;
  if (zmdUnslottedDataPoll_D((UBYTE *) &rx_data, dest_pan_id, dest_addr16) == SUCCESS)
  {
    printf("polled rx_data: ");
    for (i = 0; i < rx_data[0]+3; i++)
      printf("%02X ", rx_data[i]);
    printf("\n");
  }
}



/*****************************************************************************
* Function      : cmdAutoBcTx
* Description : controls the auto beacon transmission
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdAutoBcTx (void)
{
  UINT8 payload_length;
  UBYTE payload[2];

  char input_str[6];
  unsigned int value;
  unsigned char channel;
  unsigned char bo, so;
```

```
  printf("\nAutoBcTx:\n");

  printf("Please enter hex ('0'...'a') for channel > ");
  gets(input_str, 2);
  value = atoh(input_str);
  //channel = (unsigned char)(value >> 8);
  channel = (unsigned char)(value);
  plmeSetRequest(phyCurrentChannel, &channel);

  printf("\nPlease enter beacon order > ");
  gets(input_str, 2);
  value = atoh(input_str);
  //bo = (unsigned char)(value >> 8);
  bo = (unsigned char)(value);
  if (bo > 15)
    bo = 15;
  macPIB.macBeaconOrder = bo;

  printf("\nPlease enter superframe order (<=%u) > ", (UINT8)bo);
  gets(input_str, 2);
  value = atoh(input_str);
  so = (unsigned char)(value);
  if (so > bo)
    so = bo;
  macPIB.macSuperframeOrder = so;

  printf("\nAuto beacon transmission runs forever, interrupt with 'ESC'\n");
  payload_length = 2;
  payload[0] = '0';
  payload[1] = '1';
  //channel = 0;
  //plmeSetRequest(phyCurrentChannel, &channel);
  macPIB.macBeaconPayloadLength = 2;
  zmdAutoBtx(AM_SHORT_ADDR, payload);
}


/*****************************************************************************
* Function      : cmdBcTrack
* Description : controls the beacon tracking
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdBcTrack (void)
{
  printf("\nBcTr:\n");
  printf("\n...break with >Esc< key\n");
  zmdAutoBtr();
}


/*****************************************************************************
```

```
* Function      : cmdBcTrack
* Description : controls the beacon tracking
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdPassiveScan (void)
{
  UINT8 channel, i;
  UBYTE data_buffer[aMaxPHYPacketSize];
  UINT8 agc_lvl;

  printf("Passive Scan\n");

  for (channel = 0; channel <= 10; channel++)
  {
    printf("ch 0x%02X  ", channel);
    plmeSetRequest(phyCurrentChannel, &channel);
    if (zmdPassiveScan(data_buffer) == SUCCESS)
    {
      printf("SUCCESS: ");
      printf("rx packet: ");
      for (i = 0; i < data_buffer[0]+3; i++)
        printf("0x%02X ", data_buffer[i]);
      printf("\n");
      agc_lvl = zmdReadReg(AGC_LVL);
      printf("AGC_LVL 0x%02X\n", agc_lvl);
    }
    else
    {
      printf("NO_BEACON\n");
    }
  }
}


/*****************************************************************************
* Function      :
* Description :
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdEnergyDetectScan (void)
{
  UINT8 ed_value, channel, energy_value;

  printf("Energy Detection Scan . . .\n");
  printf("ch\tvalue\tPin\n");
  for (channel = 0; channel <= 0x0A; channel++)
  {
    plmeSetRequest(phyCurrentChannel, &channel);      /* set the required channel */
    ed_value = zmdEnergyDetectScan();              /* performs the scan and gets the macScanEd reg-
ister value */
```

```
    energy_value = getEnergyLevel(ed_value);
    printf("0x%02X\t0x%02X\t", channel, ed_value);
    switch (energy_value)
    {
      case 84: printf("<="); break;
      case 30: printf(">="); break;
      default: printf("  "); break;
    }
    printf(" -%u dBm\n", energy_value);
  }
  printf("ed scan completed\n");
}


/*****************************************************************************
* Function      : cmdActiveScan
* Description : runs an active scan
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdActiveScan (void)
{
  UINT8 channel;

  printf("Active Scan . . .\n");
  printf("ch\t\tbeacon\n");
  for (channel = 0; channel <= 10; channel++)
  {
    plmeSetRequest(phyCurrentChannel, &channel);
    if (zmdActiveScan() == SUCCESS)
    {
      printf("0x%02X\t\tyes\n", channel);
    }
    else
    {
      printf("0x%02X\t\tno\n", channel);
    }

  }
  printf("active scan done\n");
}


/*****************************************************************************
* Function      : cmdBcTrafficDevice_d2c
* Description : tracks beacons and sends payload with the CAP
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdBcTrafficDevice_d2c (void)
{
  char input_str[6];
```

```
  unsigned int value;
  unsigned char channel;
  UINT8 i;
  UBYTE payload[10];

  printf("cmdBcTrafficDevice_d2c\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);

  plmeSetRequest(phyCurrentChannel, &channel);

  for (i = 0; i < 5; i++)
    payload[i] = i;

  zmdAutoBtrData_D2C(5, payload);
}


/*****************************************************************************
* Function     : cmdBcTrafficCoord_d2c
* Description : transmits beacons and waits for data within the CAP
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void cmdBcTrafficCoord_d2c (void)
{
  char input_str[6];
  unsigned int value;
  unsigned char channel;
  UBYTE payload[20];
  UINT8 i;

  printf("BcTrafficCoord_d2c\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);
  printf("\n...break with >Esc< key\n");
  plmeSetRequest(phyCurrentChannel, &channel);

  for (i = 0; i < 5; i++)
  {
    payload[i] = i + '0';
  }
  macPIB.macBeaconPayloadLength = 5;

  zmdAutoBtxData_D2C(AM_SHORT_ADDR, payload);
}
```

```
/*****************************************************************************
* Function      : cmdSetPibValue
* Description : sets values to the phyPIB
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdSetPibValue (void)
{
  char input_str[6];
  unsigned int value;
  unsigned char channel;
  UINT8 tx_pwr;

  printf("set PIB value, press enter to skip the parameter:\n");
  printf("channel no (hex): ");
  gets(input_str, 2);
  if (((input_str[0] >= '0') && (input_str[0] <= '9'))
      ||((input_str[0] >= 'A') && (input_str[0] <= 'F')))
  {
  value = atoh(input_str);
  channel = (unsigned char)(value);
    if (plmeSetRequest(phyCurrentChannel, &channel) == PHY_SUCCESS)
      printf(" OK\n");
    else
      printf(" set failed\n");
  }
  else
  {
    printf(" skip channel\n");
  }
  printf("tx pwr (0: 0dBm, 1: -7dBm, 2: -14dBm, 3: -21dBm): ");
  gets(input_str, 2);
  switch (input_str[0])
    {
    case '0': tx_pwr = 0; if (plmeSetRequest(phyTransmitPower, &tx_pwr) == PHY_SUCCESS) printf("
OK0\n"); break;
    case '1': tx_pwr = 1; if (plmeSetRequest(phyTransmitPower, &tx_pwr) == PHY_SUCCESS) printf("
OK1\n"); break;
    case '2': tx_pwr = 2; if (plmeSetRequest(phyTransmitPower, &tx_pwr) == PHY_SUCCESS) printf("
OK2\n"); break;
    case '3': tx_pwr = 3; if (plmeSetRequest(phyTransmitPower, &tx_pwr) == PHY_SUCCESS) printf("
OK3\n"); break;
    default: printf("skip pwr\n");
  }
}


/*****************************************************************************
* Function      : cmdGetPibValue
* Description : gets values from the phyPIB
* Parameters   : none
* Returns       : none
```

```
* Notes         : none
*****************************************************************************/
void cmdGetPibValue (void)
{
  UINT8 temp_value;

  printf("GetPibValues:\n");

  printf("phyCurrentChannel: ");
  if (plmeGetRequest(phyCurrentChannel, &temp_value) == PHY_SUCCESS)
    printf("0x%02x\n", temp_value);
  else
    printf("?\n");

  printf("phyTransmitPower: ");
  if (plmeGetRequest(phyTransmitPower, &temp_value) == PHY_SUCCESS)
    printf("0x%02x\n", temp_value);
  else
    printf("?\n");

}


/*****************************************************************************
* Function      : cmdGetAddrEntries
* Description : gets all address entries from the trx
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdGetAddrEntries (void)
{
  UBYTE temp[8];
  UBYTE i;

  printf("IEEE address (high->low): ");
  temp[0] = zmdReadReg(MHR_TX_SRC_ADDR64_1);//mhrSrcAddr64_1Tx);
  temp[1] = zmdReadReg(MHR_TX_SRC_ADDR64_2);//mhrSrcAddr64_2Tx);
  temp[2] = zmdReadReg(MHR_TX_SRC_ADDR64_3);//mhrSrcAddr64_3Tx);
  temp[3] = zmdReadReg(MHR_TX_SRC_ADDR64_4);//mhrSrcAddr64_4Tx);
  temp[4] = zmdReadReg(MHR_TX_SRC_ADDR64_5);//mhrSrcAddr64_5Tx);
  temp[5] = zmdReadReg(MHR_TX_SRC_ADDR64_6);//mhrSrcAddr64_6Tx);
  temp[6] = zmdReadReg(MHR_TX_SRC_ADDR64_7);//mhrSrcAddr64_7Tx);
  temp[7] = zmdReadReg(MHR_TX_SRC_ADDR64_8);//mhrSrcAddr64_8Tx);
  for (i = 0; i < 8; i++)
    printf("0x%02X ", temp[7-i]);
  printf("\n");
}


/*****************************************************************************
* Function      : cmdBcTrafficDevice_d2c_Gts
* Description : tracks beacons and sends data using GTS
* Parameters    : none
```

```
* Returns      : none
* Notes        : none
*******************************************************************************/
void cmdBcTrafficDevice_d2c_Gts (void)
{
  char input_str[6];
  unsigned int value;
  unsigned char channel;
  UINT8 i;
  UBYTE payload[10];

  printf("cmdBcTrafficDevice_d2c_Gts\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);

  plmeSetRequest(phyCurrentChannel, &channel);

  for (i = 0; i < 5; i++)
    payload[i] = i;

  zmdAutoBtrData_GTS(5, payload);
}


/*******************************************************************************
* Function     : cmdBcTrafficCoord_d2c_Gts
* Description : transmits beacons and receives data using GTS
* Parameters   : none
* Returns      : none
* Notes        : none
*******************************************************************************/
void cmdBcTrafficCoord_d2c_Gts (void)
{
  char input_str[6];
  unsigned int value;
  unsigned char channel;
  UBYTE payload[20];
  UINT8 i;

  printf("cmdBcTrafficCoord_d2c_Gts\n");
  printf("Please enter hex ('0'...'a') for channel >");
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);
  printf("\n...break with >Esc< key\n");
  plmeSetRequest(phyCurrentChannel, &channel);

  for (i = 0; i < 5; i++)
  {
    payload[i] = i + '0';
  }
  macPIB.macBeaconPayloadLength = 5;
```

```
  zmdAutoBtxData_GTS(AM_SHORT_ADDR, payload);
}



/*****************************************************************************
* Function      : cmdPowerSupplyVoltage
* Description : get power supply voltage
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdPowerSupplyVoltage (void)
{
#if HW_PLATFORM == STARTER_KIT
  UINT16 vin;
  float vin_float;

  getPowerSupplyVoltage();      // get it twice: first value is not OK
  vin = getPowerSupplyVoltage();
  vin += 186;    // 0.45V = 186; drop-out voltage of the Schottky diode
  vin_float = (float)vin / 4095 * 2.5 * 2 * 2;  // gain 0.5 and voltage divider
  printf("ZMD44102SK board's power supply voltage: %2.2f V\n", vin_float);
#endif
}


/*****************************************************************************
* Function      : cmdRfTxTestPnSeq
* Description : set the trx to test mode "pn-sequence"
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdRfTxTestPnSeq (void)
{
  char input_str[2];
  unsigned int value;
  unsigned char channel;

  printf("Run RF_Tx test pn-sequence\n");
  printf("Enter channel number ('0'...'A')> ");
  gets(input_str, 2);
  value = atoh(input_str);
  channel = (unsigned char)(value);
  plmeSetRequest(phyCurrentChannel, &channel);
  zmdReset(HOLD_TIME_1);
  zmdInit();
  zmdSetPibValues();
  zmdWriteReg(TX_MODE, (TEST_MODE_PN |zmdReadReg(TX_MODE)));
  zmdWriteReg(MAC_TX_CONFIG, MC_DIRECT_TX);
  zmdWriteReg(MAC_CTRL, MC_TX_ON);
```

```
  printf("\nPress any key to stop\n");

  while(!(IFG2 & URXIFG1)); // wait for rx buf full
  zmdReset(HOLD_TIME_1);
}


/*****************************************************************************
* Function      : cmdRfTxTestContCarr
* Description : set the trx to test mode continuous carrier
* Parameters    : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void cmdRfTxTestContCarr (void)
{
  printf("\nIniciando Continuous Carrier....");
  zmdReset(HOLD_TIME_3);
  printf("\nreset");

  zmdWriteReg(MTP_CTRL, 0x09);          /* start programming */
  printf("\nstart programing");
  wait_us(500);
  zmdWriteReg(MTP_CTRL, 0x00);                /* start programming */
  printf("\nstart programing");
  zmdWriteReg(0x00, 0x01);              /* channel 1 */
  zmdWriteReg(0x05, 0x02);
  zmdWriteReg(0xB1, 0x04);
  zmdWriteReg(0xA0, 0x03);
  printf("\nCorriendo Continuous Carrier....\n");
}


/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

E.2.4. zmd.h

```
/*****************************************************************************
* PREVENT DOUBLE-INCLUSION
*****************************************************************************/
#ifndef _ZMD_H_
#define _ZMD_H_

#include "smac.h"

//#include "globals.h"


/*****************************************************************************
* GLOBAL DEFINES
*****************************************************************************/
```

```
#define SRC_PAN_ID  0x0102
#define SRC_ADDR16  0x0304

void zmdInit (void);

void zmdSetPibValues (void);
UINT8 zmdGetRxPacket (unsigned char *data_buffer);


void zmdSetupBoSo (UINT8 beacon_order, UINT8 superframe_order);

/*****************************************************************************
* Function      : zmdSniffer
* Description : sniffer
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
void zmdSniffer (void);




/*****************************************************************************
* Function      : zmdWaitForIRQ
* Description : waits for interrupt request from ZMD44101
* Parameters   : none
* Returns       : none
* Notes         : none
*****************************************************************************/
UINT8 zmdWaitForIRQ (void);

void zmdHandleRxPacket (UBYTE *data_buffer);

UINT8 getLqiValue (UINT16 lqi_16);


UINT8 getRssiValue (void);

void zmdInitForSendCustomPackage();

void SendCustomPackage(UBYTE largo, UBYTE potencia_de_tx, UBYTE *paquete);

#endif  /* end of 'prevent double inclusion' */
```

E.2.5. zmd.c

```
/*****************************************************************************
* SYSTEM INCLUDES
*****************************************************************************/
#include <stdio.h>              /* Standard function declarations */
#include <string.h>


/*****************************************************************************
* USER INCLUDES
*****************************************************************************/
#include "globals.h"            /* global defines */
#include "zmd.h"              /* include own prototypes */
#include "zmd44102_reg.h"       /* register defines for the ZMD44102 */
#include "zmd_hal.h"          /* prototypes for the ZMD's HAL */
#include "uart.h"             /* functions to control the UART of the C8051F124 */
#include "leds.h"             /* prototype includes for LED control */
#include "phy.h"              /* physical layer PIB, like channel setting */


/*****************************************************************************
* EXTERNAL VARIABLES
*****************************************************************************/
extern volatile unsigned char irq_flag;      /* see 'main.c' */
extern MAC_S_PIB macPIB;         /* PIB part of the mac layer */



/*****************************************************************************
* Function      : zmdInit
* Description : inits the ZMD44102's triming and register values
* Parameters    : none
* Returns       : none
* Notes         : run it only once after a reset command
*****************************************************************************/
void zmdInit (void)
{
  zmdWriteReg(MTP_CTRL, 0x09);              /* start programming */
  while (zmdReadReg(MTP_CTRL) != 0x0C);     /* zmd datasheet p.118 */

  zmdWriteReg(MTP_CTRL, 0x00);              /* end of programming */

  // set up registers for best performance results
  zmdWriteReg(CLIP_CHK_AGC_LVL_TH, 0x50);
  zmdWriteReg(AGC_LVL_INIT, 0x90);
  zmdWriteReg(ACQ_PEAK_TH_SC_TRIM, 0xC3);
  zmdWriteReg(RPATCW, 0x02);
  zmdWriteReg(RSTX, 0x0B);
  zmdWriteReg(EDO_TRIM, 0x87);

  if (phyPIB.phyCurrentChannel > 0)
  {
    zmdWriteReg(CLIP_CNT_STARTSMP, 0x78);
```

```
    zmdWriteReg(CLIP_CNT_TH, 0x04);
    zmdWriteReg(RX_MODE, 0x62);
    zmdWriteReg(ACQ_PEAK_TH_0, 0x00);
    zmdWriteReg(ACQ_PEAK_TH_1, 0x04);
    // for best performance at channel 10
    if (phyPIB.phyCurrentChannel == 0x0A)
    {
      zmdWriteReg(PHY_CHANNEL, 0xFF);
      zmdWriteReg(0x01, 0x00);
    }
  }
  else
  {
    zmdWriteReg(ACQ_PEAK_TH_1, 0x03);
    zmdWriteReg(ACQ_PEAK_TH_0, 0xA0);
    zmdWriteReg(CLIP_CNT_STARTSMP, 0xF0);
  }
  zmdWriteReg(CLK_OUT_CONFIG, 0x00);       /* switch clk output off, lower power consumtion */
}



/*****************************************************************************
* Function     : zmdSetPibValues
* Description : set the PIB to the transceiver
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void zmdSetPibValues (void)
{
  UINT8 temp_value;

  if (plmeGetRequest(phyCurrentChannel, &temp_value) == PHY_SUCCESS)
    zmdWriteReg(PHY_CHANNEL, temp_value);
  // for best performance at channel 10
  if (temp_value == 0x0A)
  {
    zmdWriteReg(PHY_CHANNEL, 0xFF);
    zmdWriteReg(0x01, 0x00);
  }
  if (plmeGetRequest(phyTransmitPower, &temp_value) == PHY_SUCCESS)
    zmdWriteReg(TX_MODE, (temp_value << 4));
}



/*****************************************************************************
* Function     : zmdSniffer
* Description : switches the trx to receive mode and prints the received
*                 frame and the LQI to the terminal window
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
```

```
void zmdSniffer (void)
{
  UINT8 irq_reason;
  UBYTE buffer[aMaxPHYPacketSize];
  UINT8 i;
  UINT16 lqi_16;
  UINT8 lqi_8;

  zmdReset(HOLD_TIME_1);                    /* ZMD44102's reset */
  zmdInit();
  zmdWriteReg(0x00, 0x01);
  zmdWriteReg(MAC_FILT_CONFIG, 0x00); // fede:  sin filtro.
  zmdWriteReg(MAC_RX_CONFIG, MC_FIFO_STORE_LQI |MC_FIFO_STORE_ACK |MC_CONT_RX);    // disable auto ack, store LQI, continue rx

  zmdWriteReg(MAC_CTRL, MC_RX_ON);          /* switch the receiver on */
  printf("\nSniff canal 1 |ZMD44102 waiting, press any char for exit\n");
  while (1)
  {
          irq_reason = zmdWaitForIRQ(); /* read the reason of the IRQ */
    if (irq_reason & IRQ_RX)                /* RX IRQ reason */
    {
      if (zmdGetRxPacket((UBYTE*)&buffer))
      {
        printf("rx packet");
        printf(" (%u bytes; filter_status%02X): ", buffer[0], zmdReadReg(MAC_FILT_STATUS));
        for (i = 1; i < buffer[0]+1; i++) // LQI + length
          printf("%02X ", buffer[i]);
        lqi_16 = buffer[buffer[0]+1] |((UINT16)buffer[buffer[0]+2] << 8);
        lqi_8 = getLqiValue(lqi_16);
        printf(", LQI_8%u\n", lqi_8);
      }
      else
      {
        printf("\n Se recibio un paquete vacio.");
      }
    }
    else if (irq_reason == 0x00)                /* user has interrupted the receive functionality */
    {
      zmdReset(HOLD_TIME_1);                /* reset to stop receiving */
      break;
    }
    else
    {
      printf("irq_reason 0x%02X\n", irq_reason);
    }
  }
}


/*****************************************************************************
* Function      : zmdGetRxPacket
* Description : read one rx packet from the RxFIFO and stores the read data
```

```
*                   to the given address
* Parameters   : data_buffer: pointer to storage area
* Returns      : number of read bytes, frame length, w/ length and w/o LQI
* Notes        : none
*****************************************************************************/
UINT8 zmdGetRxPacket (UBYTE *data_buffer)
{
  if (zmdReadReg(FIFO_FRM_PEND))   /* read the frame pending value in order to update it */
  {
    data_buffer[0] = zmdReadReg(RX_FIFO); /* get the length of the packet */
    if (data_buffer[0] == 0)
      return 0;
    zmdReadBlock(RX_FIFO, data_buffer[0]+2, &data_buffer[1]);
    return data_buffer[0];
  }
  else   // no data is pending in the fifo
    return 0;
}




/*****************************************************************************
* Function     : zmdWaitForIRQ
* Description : waits for interrupt request from ZMD44102
* Parameters   : none
* Returns      : irq_reason of the interrupt, or 0x00 indicating terminal input
* Notes        : blocking function
*****************************************************************************/
UINT8 zmdWaitForIRQ (void)
{
  unsigned char reason, key;

  while ((!inkey())&&(!irq_flag));

  if (irq_flag){
    reason = zmdReadReg(IRQ_REASON);   /* read the reason of the IRQ */
    irq_flag = FALSE;
    return reason;
  }
  key = getc();
  return 0x00;       /* irq_flag is set by ISR */
}


#define LQI_MAX_EU    1400
#define LQI_MIN_EU    600
#define LQI_OFFSET_EU LQI_MIN_EU
#define LQI_SLOPE_EU  0.32

#define LQI_MAX_US    1100
#define LQI_MIN_US    400
#define LQI_OFFSET_US LQI_MIN_US
#define LQI_SLOPE_US  0.36
```

```
/*****************************************************************************
 * Function      : getLqiValue
 * Description : transforms the provided 16-bit LQI value into a 8-bit
 *                   compliant value
 * Parameters    : lqi_16: 16-bit LQI value
 * Returns       : lqi_8: 8-bit compliant LQI value
 * Notes         : none
 *****************************************************************************/
UINT8 getLqiValue (UINT16 lqi_16)
{
  UINT8 lqi_8;

  if (phyPIB.phyCurrentChannel > 0) // US mode
  {
    if (lqi_16 > LQI_MAX_US)
      lqi_8 = 0xFF;
    else if (lqi_16 < LQI_MIN_US)
      lqi_8 = 0x00;
    else
      lqi_8 = (UINT8)((float)LQI_SLOPE_US * ((float)lqi_16 - LQI_OFFSET_US));
  }
  else  // EU mode, channel 0
  {
    if (lqi_16 > LQI_MAX_EU)
      lqi_8 = 0xFF;
    else if (lqi_16 < LQI_MIN_EU)
      lqi_8 = 0x00;
    else
      lqi_8 = (UINT8)((float)LQI_SLOPE_EU * ((float)lqi_16 - LQI_OFFSET_EU));
  }
  return lqi_8;
}



/*****************************************************************************
 * Function      : getRssiValue
 * Description : reads the AgcLvl form the last received frame and transforms
 *                   it to a RSSI value in (-1) dBm
 * Parameters    : none
 * Returns       : RSSI value in (-1) dBm
 * Notes         : cannot be used in ContRx mode
 *****************************************************************************/
UINT8 getRssiValue (void)
{
  UINT8 rssi;
  UINT8 agc_lvl;
  float agc, agc2, agc3, agc_temp1, agc_temp2, agc_sum;

  agc_lvl = zmdReadReg(AGC_LVL);

  agc = (float)agc_lvl;
  agc2 = agc * agc;
  agc3 = agc2 * agc;
```

```
  agc_temp1 = -0.00001 * agc3;
  agc_temp2 = (0.0025 * agc2);
  agc_sum = agc_temp1 + agc_temp2;
  agc_temp1 = -0.7033 * agc;
  agc_sum = agc_sum + agc_temp1;
  agc_sum = agc_sum - 16.961;
  agc_sum = -1 * agc_sum;
  rssi = (UINT8) agc_sum;

  if (phyPIB.phyCurrentChannel > 0) // US mode
  {
    if (rssi > 103)
      rssi = 103;
  }
  else     // EU mode
  {
    if (rssi > 105)
      rssi = 105;
  }

  return rssi;
}



void zmdSetupBoSo (UINT8 beacon_order, UINT8 superframe_order)
{
  UINT32 beacon_interval;
  UINT8 i;

  beacon_interval = 960;                    /* beacon interval, 960*2^beacon order, start value */
  for (i = 0; i < beacon_order; i++)
    beacon_interval <<= 1;                   /* shift beacon order entry to the corresponding bit */

  zmdWriteReg(T_BCN_INTERVAL_1, beacon_interval);
  zmdWriteReg(T_BCN_INTERVAL_2, (beacon_interval >> 8));
  zmdWriteReg(T_BCN_INTERVAL_3, (beacon_interval >> 16));
  zmdWriteReg(SF_ORDER, superframe_order);
}

void zmdInitForSendCustomPackage()
{
  zmdReset(HOLD_TIME_1);

  /* comienza programacion: ver zmd datasheet p.118 */
  zmdWriteReg(MTP_CTRL, 0x09);
  while (zmdReadReg(MTP_CTRL) != 0x0C); /* ver zmd datasheet p.118 */
  zmdWriteReg(0x37, 0x00);

  /* Configuramos registros como recomienda en datasheet p.118 */
  zmdWriteReg(CLIP_CHK_AGC_LVL_TH, 0x50);
  zmdWriteReg(AGC_LVL_INIT, 0x90);
```

```
  zmdWriteReg(ACQ_PEAK_TH_SC_TRIM, 0xC3);
  zmdWriteReg(RPATCW, 0x02);
  zmdWriteReg(RSTX, 0x0B);
  zmdWriteReg(EDO_TRIM, 0x87);
  zmdWriteReg(ACQ_PEAK_TH_1, 0x03);
  zmdWriteReg(ACQ_PEAK_TH_0, 0xA0);
  zmdWriteReg(CLIP_CNT_STARTSMP, 0xF0); /* switch clk output off, lower power consumtion */
  zmdWriteReg(CLK_OUT_CONFIG, 0x00);
  zmdWriteReg(PHY_CHANNEL, 0x01);    /* define transmision en el canal 1 (906 MHZ) */

                                     /* configuro macTxConfig para escribir mi propia */
  zmdWriteReg(MAC_TX_CONFIG, 0x22); /* trama y usar CSMA y que no maneje */
}

void SendCustomPackage(UBYTE largo, UBYTE potencia_de_tx, UBYTE *paquete)
{
  UBYTE i;

  /* copio paquete a la cola de salida del ZMD */
  for (i=0;i<largo;i++)
    zmdWriteReg(TX_FIFO,paquete[i]);

  zmdWriteReg(MSDU_TX_LENGTH, largo); /* escribe un largo de trama            */
  zmdWriteReg(TX_MODE, (potencia_de_tx & 0x03)<<4); /* configuramos la potencia_de_tx dada */
  zmdWriteReg(MAC_CTRL, 0x03); /* le decimos que arranque, que transmita lo que quiera */
  //DEBUG(("Paquete custom enviado..."));
  return 0;
}




/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

E.2.6. uart.h

```
/*****************************************************************************
* PREVENT DOUBLE-INCLUSION
*****************************************************************************/
#ifndef UART_INCL
#define UART_INCL

#include <io.h>

/*****************************************************************************
*!Prototipos para uart defines para compatibilidad con stdio y funciones inline.
*
* Las funciones inline son definidas acá acorde a:
* http://www.greenend.org.uk/rjk/2003/03/inline.html.
*
* \note: Estas biblioteca esta basada en el trabajo de Peter Baxendale
```

```
* (http://www.dur.ac.uk/peter.baxendale/)
*****************************************************************************/

/*****************************************************************************
* GLOBAL DEFINES
*****************************************************************************/
#define gets uartGets
#define getchar uartRx
#define puts    uartTxString
#define putc    uartTx
#define getc    uartRx
#define printf(format, ...) uprintf(pchar1, format, ## __VA_ARGS__)

#ifdef   _DEBUG_
# define DEBUG(s)   printf s
#else
# define DEBUG(s)
#endif


/*****************************************************************************
* Function      : pchar1
* Description : Transmite un caracter a UART0 (para usar con uprintf).
* Parameters   : c- caracter a enviar
* Returns      : 1 - retorna siempre esto para que funcione con uprintf
* Notes        : none
*****************************************************************************/
int pchar1(int);

/*****************************************************************************
* Function      : uartInit
* Description : Inicia el puerto UART0 configurándolo a 9600 utilizando el reloj
*       ACLK.
* Parameters   : none
* Returns      : none
* Notes        : none
*****************************************************************************/
void uartInit(void);

/*****************************************************************************
* Function      : uartTxString
* Description : Envía un string por el UART0.
* Parameters   : *p puntero a el string a enviar (terminado por 0x00)
* Returns      : none
* Notes        : none
*****************************************************************************/
void uartTxString(char *);

/*****************************************************************************
* Function      : uartGets
* Description : Lee una linea delimitada por CR o por n caracteres.
* Parameters   : s - Puntero a el string (deimitado por NULL)
     n - Máximo largo del string
* Returns      : Numero de caracteres capturados
```

```
* Notes          : Esta función espera a que se reciba una linea limitada por CR
*       (0x0D) o a que se reciban n caracteres. La rutina también
*       maneja el caracter BACKSPACE, permitiendo que se ingrese pueda
*       borrar.
*******************************************************************************/
unsigned int uartGets(char *s, unsigned int n);

/*******************************************************************************
* Function      : atoh
* Description : Convierte ascii a byte
* Parameters    : p_escape_hex hexadecimal escapado, 2 o 3 caracteres de la forma FF
*                   o \FF
* Returns       : el byte representado por el hexadecimal escapado, en caso de
*                   no ser un hexadecimal escapado retorno 0
* Notes         :
*       Entra un puntero a caracter, lee los primeros caracteres y si son
*       números hexadecimales de la forma forma \af o AF... los convierte a byte.
*       No es sensible a mayúsculas.
*******************************************************************************/
unsigned char atoh(char * p_escape_hex);


/*******************************************************************************
                                Funciones InLine
 *******************************************************************************/

/*******************************************************************************
* Function      : uartTx
* Description : Funcion inline para enviar un caracter.
* Parameters    : c - caracter a ser enviado
* Returns       : none
* Notes         : none
*******************************************************************************/
static inline void uartTx(char c) {
  while(!(U0IFG & UTXIFG0));
  U0TXBUF = c;
}

/*******************************************************************************
* Function      : uartRx
* Description : Función inline pare recibir un caracter.
*                   Espera a que se llene el buffer y retorna el caracter en el
*                   mismo.
* Parameters    : none
* Returns       : caracter recibido
* Notes         : Bloquea hasta recibir
*******************************************************************************/
static inline unsigned int uartRx(void) {
  while(!(U0IFG & URXIFG0));
  return U0RXBUF;
}


/*******************************************************************************
```

```
* Function      : inkey
* Description : Funcion inline para chequear el estado del buffer de recepción.
* Parameters    : none
* Returns       : 0 - buffer esta vació
*              >1 - hay algo en el buffer
* Notes         : none
*******************************************************************************/
static inline unsigned int inkey(void) {
  return (U0IFG & URXIFG0);
}

#endif
```

E.2.7. uart.c

```
/* BIBILOTECA UART PARA MSP430 */

#include <io.h>
#include "uart.h"


/*******************************************************************************
* Function      : pchar1
* Description : Transmite un caracter a UART0 (para usar con uprintf).
* Parameters    : c- caracter a enviar
* Returns       : 1 - retorna siempre esto para que funcione con uprintf
* Notes         : none
*******************************************************************************/
int pchar1(int c) {
  if(c == '\n') uartTx('\r');
  uartTx(c);
  return 1;
}

/*******************************************************************************
* Function      : uartInit
* Description : Inicia el puerto UART0 configurándolo a 9600 utilizando el reloj
*                 ACLK.
* Parameters    : none
* Returns       : none
* Notes         : none
*******************************************************************************/
void uartInit(void) {
  U0CTL = SWRST + CHAR;       /* se mantiene en reset hasta configurarlo */
                              /* se setea CHAR para largo de caracter=8   */

  U0TCTL = SSEL_ACLK;         /* selección del reloj ACLK (32khz)         */
  U0RCTL = 0;

    /* distintas configuración de BAUDRATE, calculadas utilizando
     * BaudRateCalculator del proyecto MSPGCC.
     * (http://mspgcc.sourceforge.net/baudrate.html)
```

```
     */
//   UBR00=0x0D; UBR10=0x00; UMCTL0=0x6D;       /* uart0 32768Hz 2409bps   */
     UBR00=0x03; UBR10=0x00; UMCTL0=0x29;       /* uart0 32768Hz 9637bps   */
//   UBR00 = 0x3e; UBR10 = 01; U0MCTL = 0xbb; /* 9600, 3.06MHz dco clock */

  U0ME |= UTXE0;   /* habilita transmisión */
  U0ME |= URXE0;   /* habilita recepción   */

  U0IE = UTXIE0 + URXIE0;   /* habilita interrupciones */
                            /*   - TX - se envió       */
                            /*   - RX - llego dato     */

  P3SEL |=  (1 << 4) |(1 << 5);   /* P3 bits 4,5 */
  P3DIR |=  (1 << 4);              /* P3.4entrada (tx) */
  P3DIR &= ~(1 << 5);              /* p3.5 salida (rx)  */

  U0CTL &= ~SWRST;   /* levanto el reset del puerto 0 */

  uartTx(0x00); /* envió un caracter nulo */
}


/*****************************************************************************
* Function      : uartTxString
* Description : Envía un string por el UART0.
* Parameters    : *p puntero a el string a enviar (terminado por 0x00)
* Returns       : none
* Notes         : none
*****************************************************************************/
void uartTxString(char *p) {
  while (*p) {                      /* mientras no llego a null                */
    uartTx(*p++);                   /* se envía caracter e incrementa puntero */
  }
}


/*****************************************************************************
* Function      : uartGets
* Description : Lee una linea delimitada por CR o por n caracteres.
* Parameters    : s - Puntero a el string (deimitado por NULL)
    n - Máximo largo del string
* Returns       : Numero de caracteres capturados
* Notes         : Esta función espera a que se reciba una linea limitada por CR
*       (0x0D) o a que se reciban n caracteres. La rutina también
*       maneja el caracter BACKSPACE, permitiendo que se ingrese pueda
*       borrar.
*****************************************************************************/
unsigned int uartGets(char *s, unsigned int n) {
  unsigned int len=0;
  char endline=0;

  while(n && !endline){
    *s = uartRx();        /* leo un caracter */
```

```
    if(*s == '\r') {      /* si es retorno de carro */
      uartTx('\n');       /* envio nueva linea      */
      *s = 0;             /* y cierro el string     */
      endline=1;          /* salgo del bucle        */
    }

    else if(*s == '\b') {     /* si es BACKSPACE                                  */
      if(len != 0){
        uartTxString(" \b"); /* borrar el caracter ya enviado                    */
        len--;               /* decrementa len pues se borro un caracter         */
        s--;                 /* se remueve el caracter del buffer                */
        n++;                 /* decrementa n, pues hay que leer char borrado */
      }
      n++;             /* el BACKSPACE no se guardo, hay que leer un byte mas   */
    }

    else {
                         /* si llegue hasta aquí el caracter no es especial, */
      len++;             /* por lo cual incremento len,                      */
      uartTx(*s);        /* ejecuto el eco                                   */
      s++;               /* e incremento puntero                             */
    }

  }
  return len;            /* devuelvo el largo del string */
}

/*****************************************************************************
* Function      : atoh
* Description : Convierte ascii a byte
* Parameters    : p_escape_hex hexadecimal escapado, 2 o 3 caracteres de la forma FF
*                   o \FF
* Returns       : el byte representado por el hexadecimal escapado, en caso de
*                   no ser un hexadecimal escapado retorno 0
* Notes         :
*       Entra un puntero a caracter, lee los primeros caracteres y si son
*       números hexadecimales de la forma forma \af o AF... los convierte a byte.
*       No es sensible a mayúsculas.
*****************************************************************************/
unsigned char atoh(char * p_escape_hex)
{
  unsigned char output;

  if (p_escape_hex[0] == '\\')        /* si esta precedido por un \ lo salto */
    p_escape_hex++;

  if (( p_escape_hex[0] >= '0') &&          /* si primer byte esta entre 0-F */
      ( p_escape_hex[0] <= '9'))            /* guardo la parte del byte que   */
    output = (p_escape_hex[0] & 0x0F) << 4;/* representa el numero, corrida */
  else
    if (( (p_escape_hex[0]&(~(1<<5))) >= 'A') &&
        ( (p_escape_hex[0]&(~(1<<5))) <= 'F'))
      output = ((p_escape_hex[0] & 0x0F) + 9) << 4;
  else return 0;                            /* sino devuelvo 0 */
```

```
  if (( p_escape_hex[1] >= '0') &&        /* si el byte esta entre 0-F   */
      ( p_escape_hex[1] <= '9'))           /* sumo la parte del byte que */
    output += (p_escape_hex[1] & 0x0F);    /* representa el numero       */
  else
    if (( (p_escape_hex[1]&~(1<<5)) >= 'A') &&
     ((p_escape_hex[1]&(~(1<<5))) <= 'F'))
      output += ((p_escape_hex[1] & 0x0F) + 9);
  else return 0;                            /* sino retorno 0 */


  return output; /* retorno el byte */
}


/* vim: set shiftwidth=2 tabstop=2 expandtab: */
```

Apéndice F

ZMD44102

F.1. Características generales

A continuación se resumen las características generales más destacadas del ZMD44102:

- Transceiver RF con operación en la banda ISM
- Cumple con IEEE 802.15.4
- Soporte de hardware integrado (PHY y Thin HW-MAC) para control de subcapa MAC
- Direct Sequence Spread Spectrum (DSSS)
- Tasa de transmisión de 20 kbits/s (para 868.3MHz), 40 kbits/s (para 915MHz)
- Rango de transmisión de 100 metros (0 dBm, LoS)
- Modos de operación de bajo consumo
- Interfaces SPI y puerto paralelo
- Package Quad Flat pack No lead (QFN) de 48 pines (7x7 mm)

F.2. Características eléctricas

A continuación se resumen las características eléctricas de operación más destacadas:

- Rango de temperatura: -40^oC a +85^oC
- Voltaje de alimentación (Analógico): +2.4V

- Votaje de alimentación (Digital): +3.3V
- Corriente de alimentación, modo Tx activo (0 dBm): 28mA
- Corriente de alimentación, modo Rx activo: 29mA
- Corriente de alimentación, modo Off: 1.3μA
- Corriente de alimentación, modo Sleep: 2.3μA
- Bandas de frecuencia: 868MHz y 915MHz

F.3. Pinout

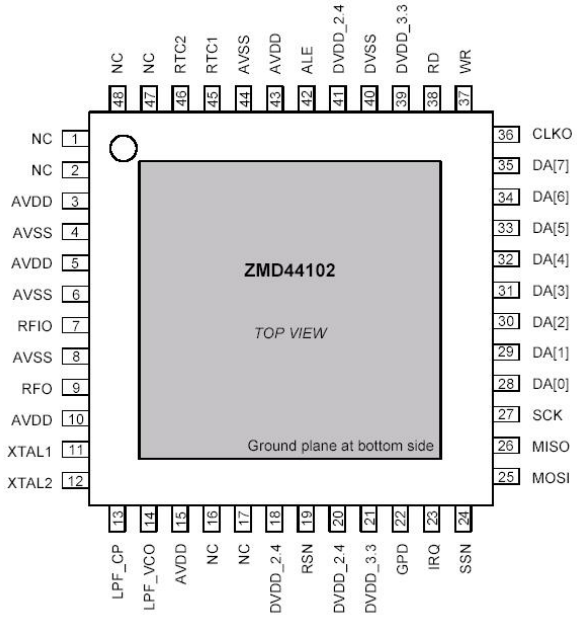

Figura F.1: Asignación de pines del ZMD44102

La tabla siguiente muestra la descripción de los pines del transceiver:

PIN	NOMBRE	TIPO	DESCRIPCIÓN
1	NC	-	Sin conexión
2	NC	-	Sin conexión
3	AVDD	AVDD	Fuente analógica, típico 2.4V
4	AVSS	AVSS	Tierra analógica
5	AVDD	AVDD	Fuente RF, típico 2.4V
6	AVSS	Tierra	Tierra RF
7	RFIO	RF IO	Entrada del receptor y salida del transmisor RF
8	AVSS	Tierra	Tierra RF
9	RFO	Salida RF	Salida del transmisor RF
10	AVDD	AVDD	Fuente analógica del PLL, típico 2.4V
11	XTAL1	Entrada Analógica	Entrada del cristal de 24 MHz
12	XTAL2	Salida Analógica	Salida del cristal de 24 MHz
13	LPF_CP	Salida Analógica	Loop filter, charge-pump node
14	LPF_VCO	Entrada Analógica	Filtro de loop, nodo de sintonización del VCO
15	AVDD	AVDD	Fuente analógica del PLL VCO, típico 2.4V
16	NC	-	Sin conexión
17	NC	-	Sin conexión
18	DVDD_2.4	DVDD	Fuente digital del PLL, típico 2.4V
19	RSN	Entrada CMOS	Reset del chip (activo en nivel bajo)
20	DVDD_2.4	DVDD	Fuente digital del núcleo, 2.4 (núcleo y pre-driver)
21	DVDD_3.3	DVDD_3.3	Fuente de entrada y salida digital, 3.3V (post-driver)
22	GPD	Entrada CMOS	Apagado global, desde dispositivo externo (activo en nivel alto)

PIN	NOMBRE	TIPO	DESCRIPCIÓN
23	IRQ	Salida CMOS	Pedido de interrupción, hacia dispositivo externo (activo en nivel bajo)
24	SSN	Entrada y salida CMOS	Selección de SPI en modo no esclavo (activo en nivel bajo)
25	MOSI	Entrada y salida CMOS	Selección de salida maestro/entrada esclavo (en modo esclavo)
26	MISO	Entrada y salida CMOS	Selección de entrada maestro/salida esclavo (en modo esclavo)
27	SCK	Entrada y salida CMOS	Reloj serial
28	DA[0]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
29	DA[1]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
30	DA[2]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
31	DA[3]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
32	DA[4]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
33	DA[5]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
34	DA[6]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
35	DA[7]	Entrada y salida CMOS	Transmisión y recepción de datos, usada para el la interfaz del puerto paralelo
36	CLKO	Salida CMOS	Reloj (hacia dispositivo externo)
37	WR	Entrada CMOS	Escritura de datos, usada para el la interfaz del puerto paralelo

PIN	NOMBRE	TIPO	DESCRIPCIÓN
38	RD	Entrada CMOS	Lectura de datos, usada para el la interfaz del puerto paralelo
39	DVDD_3.3	DVDD_3.3	Fuente de entrada y salida digital, 3.3V (post-driver)
40	DVSS	Tierra	Tierra digital
41	DVDD_2.4	DVDD	Fuente digital del núcleo, 2.4 (núcleo y pre-driver)
42	ALE	Entrada CMOS	Activación de dirección de registro, usada para el la interfaz del puerto paralelo
43	AVDD	AVDD	Fuente Analógica, típico 2.4V
44	AVSS	Tierra	Tierra Analógica
45	RTC1	Entrada Analógica	Entrada del cristal de 32.768 kHz crystal
46	RTC2	Salida Analógica	Salida del cristal de 32.768 kHz crystal
47	NC	-	Sin conexión
48	NC	-	Sin conexión

Nota: El ZMD44102 tiene un plano de tierra por debajo del chip, el cual disipa calor hacia el PCB y fundamentalmente brinda mayor estabilidad a las tierras analógicas y digitales.

Apéndice G

MSP430F149

G.1. Características generales

A continuación se resumen las características generales más destacadas:

- Cinco modos de bajo consumo
- Cambio de modo de bajo consumo a modo activo en menos de 6 μs
- Arquitectura RISC de 16 bits, ciclo de instrucciones de 125 ns
- Conversor A/D de 12 bits con referencia interna, Sample-And-Hold y autoscan
- Timer_B de 16 bits con siete registros Capture/Compare-With-Shadow
- Timer_A de 16 bits con tres registros Capture/Compare
- Comparador integrado
- Protección de código programado a traves de fusible de seguridad
- No necesita voltaje externo de programación
- 2 interfaces universales de comunicación serie síncrona/asíncrona (USART)
- Memoria Flash de 60KB + 256Bytes, memoria RAM de 2KB
- Package Quad Flat Pack (QFP) o Quad Flat pack No lead (QFN) de 64 pines

G.2. Características eléctricas

A continuación se resumen las características eléctricas de operación más destacadas:

- Rango de voltaje de alimentación: 1.8 V . . . 3.6 V
- Corriente de alimentación, modo Activo: 280 μA at 1 MHz, 2.2V
- Corriente de alimentación, modo Standby: 1.6 μA
- Corriente de alimentación, modo Off (RAM Retention): 0.1 μA

G.3. Pinout

La tabla siguiente muestra la descripción de los pines del microcontrolador:

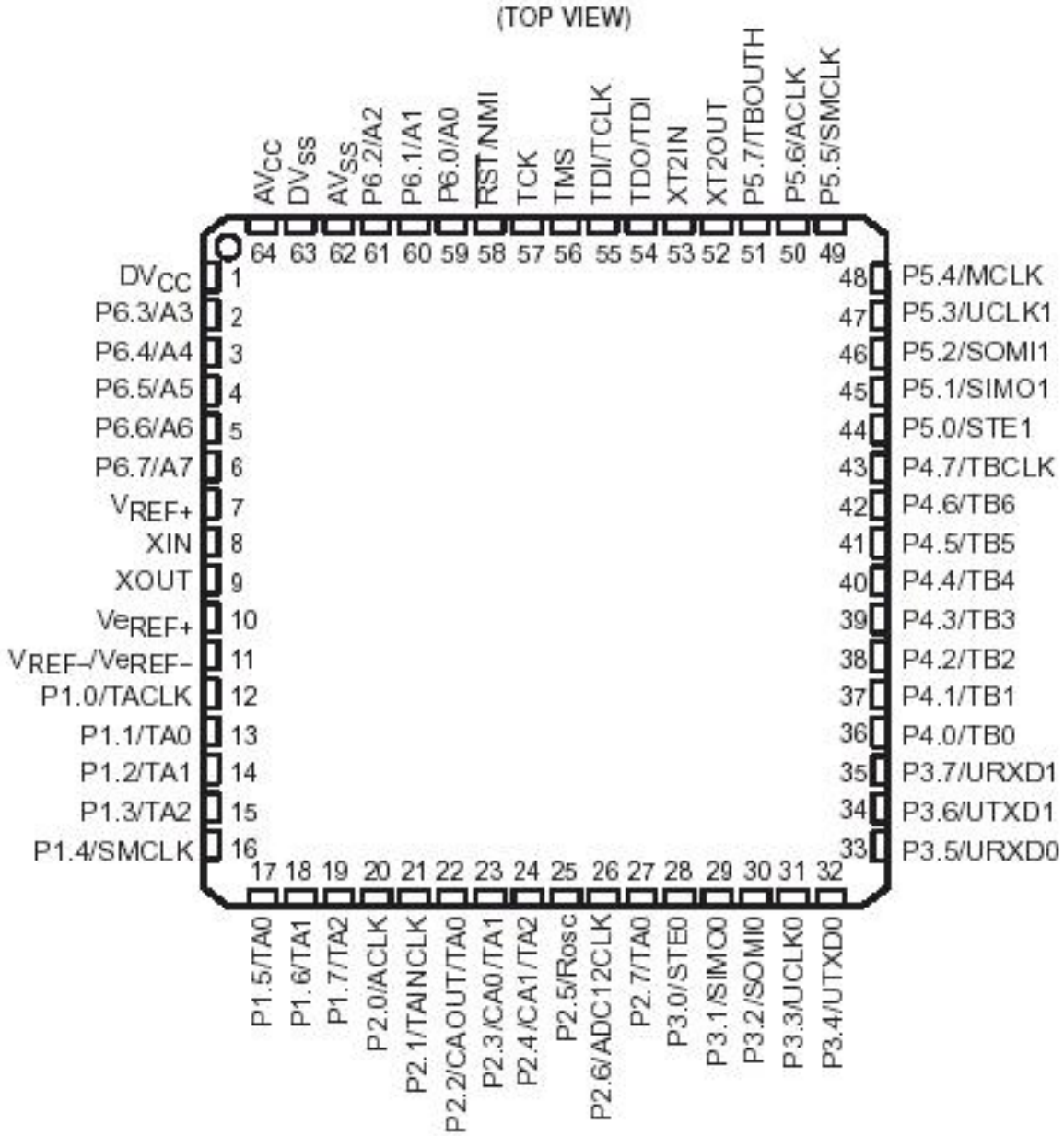

Figura G.1: Asignación de pines del MSP430F149

NOMBRE	PIN	I/O	DESCRIPCIÓN
AVCC	64		Fuente analógica. Alimenta la parte analógica del ADC
AVSS	62		Tierra analógica. Alimenta la parte analógica del ADC
DVCC	1		Fuente digital. Alimenta todas las partes digitales
DVSS	63		Tierra digital. Alimenta todas las partes digitales
P1.0/TACLK	12	I/O	Pin I/O digital de propósito general / Timer_A, entrada señal de reloj TACLK
P1.1/TA0	13	I/O	Pin I/O digital de propósito general / Timer_A, capture: entrada CCI0A, compare: salida Out0/transmisión BSL
P1.2/TA1	14	I/O	Pin I/O digital de propósito general / Timer_A, capture: entrada CCI1A, compare: salida Out1
P1.3/TA2	15	I/O	Pin I/O digital de propósito general / Timer_A, capture: entrada CCI2A, compare: salida Out2
P1.4/SMCLK	16	I/O	Pin I/O digital de propósito general / Salida señal SMCLK
P1.5/TA0	17	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out0
P1.6/TA1	18	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out1
P1.7/TA2	19	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out2
P2.0/ACLK	20	I/O	Pin I/O digital de propósito general / Salida ACLK
P2.1/TAINCLK	21	I/O	Pin I/O digital de propósito general / Timer_A, señal de reloj INCLK
P2.2/CAOUT/TA0	22	I/O	Pin I/O digital de propósito general / Timer_A, capture: entrada CCI0B/salida Comparator_A/recepción BSL
P2.3/CA0/TA1	23	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out1/entrada Comparator_A

NOMBRE	PIN	I/O	DESCRIPCIÓN
P2.4/CA1/TA2	24	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out2/entrada Comparator_A
P2.5/ROSC	25	I/O	Pin I/O digital de propósito general / Entrada resistor externo que define la frecuencia nominal dle DCO
P2.6/ADC12CLK	26	I/O	Pin I/O digital de propósito general / Reloj de conversión - 12-bit ADC
P2.7/TA0	27	I/O	Pin I/O digital de propósito general / Timer_A, compare: salida Out0
P3.0/STE0	28	I/O	Pin I/O digital de propósito general / Activación transmisión esclavo - modo USART0/SPI
P3.1/SIMO0	29	I/O	Pin I/O digital de propósito general / Entrada esclavo/salida master del modo USART0/SPI
P3.2/SOMI0	30	I/O	Pin I/O digital de propósito general / Salida esclavo/entrada master del modo USART0/SPI
P3.3/UCLK0	31	I/O	Pin I/O digital de propósito general / Reloj USART0: entrada externa - modo UART o SPI, salida - modo SPI
P3.4/UTXD0	32	I/O	Pin I/O digital de propósito general / Transmisión de datos de salida - modo USART0/UART
P3.5/URXD0	33	I/O	Pin I/O digital de propósito general / Recepción de datos de entrada - modo USART0/UART
P3.6/UTXD1	34	I/O	Pin I/O digital de propósito general / Transmisión de datos de salida - modo USART1/UART
P3.7/URXD1	35	I/O	Pin I/O digital de propósito general / Recepción de datos de entrada - modo USART1/UART
P4.0/TB0	36	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI0A o CCI0B, compare: salida Out0
P4.1/TB1	37	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI1A o CCI1B, compare: salida Out1
P4.2/TB2	38	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI2A o CCI2B, compare: salida Out2

NOMBRE	PIN	I/O	DESCRIPCIÓN
P4.3/TB3	39	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI3A o CCI3B, compare: salida Out3
P4.4/TB4	40	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI4A o CCI4B, compare: salida Out4
P4.5/TB5	41	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI5A o CCI5B, compare: salida Out5
P4.6/TB6	42	I/O	Pin I/O digital de propósito general / Timer_B, capture: entrada CCI6A o CCI6B, compare: salida Out6
P4.7/TBCLK	43	I/O	Pin I/O digital de propósito general / Timer_B, entrada de señal de reloj TBCLK
P5.0/STE1	44	I/O	Pin I/O digital de propósito general / Activación de transmisión esclavo - modo USART1/SPI
P5.1/SIMO1	45	I/O	Pin I/O digital de propósito general / Entrada esclavo/salida master del modo USART1/SPI
P5.2/SOMI1	46	I/O	Pin I/O digital de propósito general / Salida esclavo/entrada master del modo USART1/SPI
P5.3/UCLK1	47	I/O	Pin I/O digital de propósito general / Reloj USART1: entrada externa - modo UART or SPI, salida - modo SPI
P5.4/MCLK	48	I/O	Pin I/O digital de propósito general / Salida de reloj MCLK del sistema principal primario
P5.5/SMCLK	49	I/O	Pin I/O digital de propósito general / Salida de reloj SMCLK del sistema principal secundario
P5.6/ACLK	50	I/O	Pin I/O digital de propósito general / Salida auxiliar de reloj ACLK
P5.7/TBOUTH	51	I/O	Pin I/O digital de propósito general / Switchea todos los puertos de salida digital PWM a alta impedancia - Timer_B7: TB0 a TB6
P6.0/A0	59	I/O	Pin I/O digital de propósito general / Entrada analógica a0 - 12-bit ADC
P6.1/A1	60	I/O	Pin I/O digital de propósito general / Entrada analógica a1 - 12-bit ADC

NOMBRE	PIN	I/O	DESCRIPCIÓN
P6.2/A2	61	I/O	Pin I/O digital de propósito general / Entrada analógica a2 - 12-bit ADC
P6.3/A3	2	I/O	Pin I/O digital de propósito general / Entrada analógica a3 - 12-bit ADC
P6.4/A4	3	I/O	Pin I/O digital de propósito general / Entrada analógica a4 - 12-bit ADC
P6.5/A5	4	I/O	Pin I/O digital de propósito general / Entrada analógica a5 - 12-bit ADC
P6.6/A6	5	I/O	Pin I/O digital de propósito general / Entrada analógica a6 - 12-bit ADC
P6.7/A7	6	I/O	Pin I/O digital de propósito general / Entrada analógica a7 - 12-bit ADC
RST/NMI	58	I	Entrada de reset, puerto de entrada de interrupciones no-enmascarables, o encendido del BSL (en dispositivos Flash)
TCK	57	Entrada	Reloj de testeo. TCK es el puerto de entrada de reloj para testeo de dispositivos de programación y para el BSL (en dispositivos Flash)
TDI/TCLK	55	Entrada	Testeo de entrada de datos / Testeo de entrada de reloj. El fusible de protección del dispositivo está conectado a TDI/TCLK
TDO/TDI	54	I/O	Puerto de testeo de datos de salida / Terminal de programación de datos de entrada
TMS	56	Entrada	Selección de modo de testeo. El pin TMS es utilizado como puerto de entrada para dispositivos de programación y testeo
VeREF+	10	Entrada	Entrada de voltaje externo de referencia del ADC
VREF+	7	Salida	Salida del terminal positivo del voltaje de referencia del ADC
VREF-/VeREF-	11	Entrada	Terminal negativo del voltaje de referencia del ADC para ambas fuentes, el voltaje de referencia interno o el voltaje de referencia aplicado externamente

NOMBRE	PIN	I/O	DESCRIPCIÓN
XIN	8	Entrada	Puerto de entrada del oscilador de cristal XT1. Cristales estandar o de relojes pulsera pueden ser conectados
XOUT	9	Salida	Terminal de salida del oscilador de cristal XT1
XT2IN	53	Entrada	Puerto de entrada del oscilador de cristal XT2. Solamente cristales estandar pueden ser conectados
XT2OUT	52	Salida	Terminal de salida del oscilador de cristal XT2
QFN Pad	NA	NA	QFN package pad connection to DVSS recommended

Notas:

- I/O - Entrada/Salida
- BSL - El MSP430 bootstrap loader (BSL) permite al usuario programar la memoria flash RAM utilizando la interfaz UART serie. El acceso a la memoria del MSP430 via BSL está protegido por una contraseña definida por el usuario

Apéndice H

Herramientas utilizadas

H.1. MIMP

El programa MIMP (Motorola Impedance Matching Program) es un entorno simple para el ingreso y análisis de circuitos de adaptación de impedancia. En esencia el software es un diagrama de smith asistido por computadora.

Todas las redes de adaptación utilizadas en el proyecto fueron previamente analizadas y verificadas utilizando el MIMP.

H.2. TWiki

TWiki[38] es una implementación de WikiWiki (o wiki) bajo la licencia GPL[2]. WikiWiki es un sitio web que cuenta con la particularidad de ser editable desde el mismo navegador, de una forma interactiva, fácil y rápida. Dichas facilidades hacen de un wiki una herramienta efectiva para la escritura colaborativa.

En este proyecto, el TWiki fue utilizado para centralizar toda la información que generada de una manera ordenada. La posibilidad de poder acceder a la información desde cualquier navegador, tanto para su lectura como para su escritura, logro que el grupo pueda trabajar eficazmente de forma remota y asíncrona.

H.3. LaTeX

LaTeX[39] es un procesador de textos basado en un "lenguaje de marcas" (markup language). LaTeX se basa en la idea de que el autor de un texto, tiene que centrarse en lo que escribe, sin distraerse con la presentación visual del mismo. Al escribir un documento en latex, el autor solo escribe el texto con su

estructura lógica (capítulos, secciones, tablas, figuras, etc.) y el compilador latex luego se encarga de crear una representación visual para esta.

Toda la documentación de este proyecto fue escrita en LaTeX.

H.4. Eagle

Eagle[40] (Easily Applicable Graphical Layout Editor) es un programa de diseño electrónico asistido por computador (ECAD) producido por la empresa Cadsoft[41].

Los esquemáticos y el layout que componen el $\mathbf{I_{IE}M_{OTE}}$, así como el programador JTAG fueron diseñados utilizando el Eagle.

H.5. Subversion

Subversion[42] (SVN) es un sistema de control de versiones. Se entiende por sistema de control de versiones un sistema que mantiene registro de todo el trabajo y los cambios en los ficheros que forman parte de un determinado proyecto. También permite que distintas personas colaboren utilizando este mecanismo. Esta aplicación es software libre.

Para el desarrollo del hardware, el software, y la documentación se utilizo un repositorio subversión. Esto permitió una mejor eficiencia en el trabajo en grupo.

H.6. Inkscape

Inkscape[43] es un editor de gráficos vectoriales de código abierto, con capacidades similares a Illustrator, Freehand, CorelDraw o Xara X, usando el estándar de la W3C: el formato de archivo Scalable Vector Graphics (SVG). Esta aplicación es software libre, y se distribuye bajo la licencia GPL[2].

Todas las figuras creadas para esta documentación fueron realizadas en Inkscape.

Referencias

[1] GNU, *Compilador de C para el MSP430 (GCC)*,
http://mspgcc.sourceforge.net

[2] GNU, *GPL*,
http://www.gnu.org/licenses/licenses.es.html

[3] Warneke Brett, *Smart Dust*
http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/index.html

[4] Mark Weiser, *The Computer for the 21st Century*,
http://www.stanford.edu/class/cs344a/papers/computer-for-21-century.pdf

[5] Federico de Mula, Rafaella Fiorelli, Virgina Marchesano, *Proyecto de Fin de Carrera: Sistema de Comunicación en Radiofrecuencia para Dispositivos de Bajo Consumo*. Montevideo, Facultad de Ingeniería, Instituto de Ingeniería Eléctrica. 2002.

[6] Comblock, *COM-1019 Data Sheet*,
http://www.ComBlock.com/download/com1019.pdf

[7] Comblock, *COM-2001 Data Sheet*,
http://www.ComBlock.com/download/com2001.pdf

[8] Comblock, *COM-4001 Data Sheet*,
http://www.ComBlock.com/download/com4001cd.pdf

[9] Comblock, *COM-3001 Data Sheet*,
http://www.ComBlock.com/download/com3001.pdf

[10] Comblock, *COM-1018 Data Sheet*,
http://www.ComBlock.com/download/com1018.pdf

[11] Atmel. *Página web del AT86RF210*,
http://www.atmel.com/dyn/products/product_card.asp?part_id=3255

[12] ZMD, *Página web del ZMD44101*,
http://www.zmd.de/zigbee.php?content=zig&product=zmd44101

[13] ZMD, *Página web del ZMD44102*,
http://www.zmd.de/zigbee.php?content=zig&product=zmd44102

[14] Digi-Key, *Digi-Key*,
http://www.digikey.com

[15] Texas Instruments, *TPS70758 Data Sheet*,
http://focus.ti.com/lit/ds/symlink/tps70758.pdf

[16] ZMD, *ZMD44102 Data Sheet and User Manual*,
http://www.zmd.de/downloads/ZMD44102_DataSheet_v1_1.zip

[17] IEEE, *Institute of Electrical and Electronics Engineers (IEEE)*,
http://www.ieee.org

[18] Texas Instruments, *Página web del MSP430F149*,
http://focus.ti.com/lit/ds/symlink/msp430f135.pdf

[19] ZMD, *ZMD44102 Application Note 01*,
http://www.zmd.de/downloads/ZMD44102_AN01_RF_Ref_Design_v1_0.zip

[20] ZMD, *ZMD44102 Starter Kit Documentation and Tools*,
http://www.zmd.de/downloads/ZMD44102SK_Documentation_Tools_v1_0.zip

[21] Texas Instruments, *Página web del REG710-25*,
http://focus.ti.com/docs/prod/folders/print/reg710-25.html

[22] Texas Instruments, *Página web del REG711-25*,
http://focus.ti.com/docs/prod/folders/print/reg711-25.html

[23] Texas Instruments, *MSP430 Data Sheet*,
http://focus.ti.com/lit/ds/symlink/msp430f135.pdf

[24] Texas Instruments, *MSP430x1xx Family User's Guide*,
http://focus.ti.com/lit/ug/slau049f/slau049f.pdf

[25] Texas Instruments, *Application Report SLAA264*,
http://focus.ti.com/lit/an/slaa264/slaa264.pdf

[26] Texas Instruments, *MAX3221 Data Sheet*,
http://focus.ti.com/lit/ds/symlink/max3221.pdf

[27] Texas Instruments, *TPS60212 Data Sheet*,
http://focus.ti.com/docs/prod/folders/print/tps60212.html

[28] Texas Instruments, *REG711-25 Data Sheet*,
http://focus.ti.com/docs/prod/folders/print/reg711-25.html

[29] *Matching Small Loop Antennas to rfPIC Devices*,
http://ww1.microchip.com/downloads/en/AppNotes/00831b.pdf

[30] Constantine A. Balanis, *Antenna Theory: Analysis and Design*. Wiley, 2005.

[31] Frederik W. Grover, *Inductance Calculations Working Formulas and Tables*. Dover, 1946.

[32] Cris Bowick, *RF Circuit Design*. Newnes, 1982.

[33] Motorola, *MIMP Smith Chart simulation freeware software (v1.0)*.

[34] Olimex, *Olimex*,
http://www.olimex.com/dev/msp-jtag.html

[35] Realterm team, *Realterm*,
http://realterm.sf.net

[36] RAE, "Diccionario Real Academia Española'
http://buscon.rae.es/

[37] Ricreations Inc, *JTAG Cable Considerations*,
http://www.ricreations.com/AppNote003.html

[38] TWiki, *TWiki*,
http://www.twiki.org

[39] LaTeX project, *LaTeX*,
http://www.latex-project.org/

[40] Cadsoft, *Eagle*,
http://www.cadsoft.de/

[41] Cadsoft, *Cadsoft*,
http://www.cadsoft.de/

[42] Tigris.org *SubVersion*,
http://subversion.tigris.org/

[43] Inkscape, *Inkscape*,
http://www.inkscape.org/

Bibliografía

I. Cris Bowick, *RF Circuit Design*. Newnes, 1982.

II. Constantine A. Balanis, *Antenna Theory: Analysis and Design*. Wiley, 2005.

III. Sedra, Smith, *Circuitos Microelectrónicos*. Cuarta edición. Oxford.

IV. A. Bruce Carlson, *Communication Systems. An introduction to Signals and Noise in Electrical Communication*. Third Edition. Mc Graw-Hill.

V. Andrew S. Tanenbaum, *Redes de Computadoras*. México, cuarta edición. Prentice Hall, 2003.

VI. *Wikipedia: la encicopedia libre*,
http://www.wikipedia.org

VII. *Institute of Electrical and Electronics Engineers (IEEE)*,
http://www.ieee.org

VIII. *Comblock*,
http://www.comblock.com

IX. Federico de Mula, Rafaella Fiorelli, Virgina Marchesano, *Proyecto de Fin de Carrera: Sistema de Comunicación en Radiofrecuencia para Dispositivos de Bajo Consumo*. Montevideo, Facultad de Ingeniería, Instituto de Ingeniería Eléctrica. 2002.

X. *Información sobre montaje de componentes SMD*,
http://www.answers.com/topic/surface-mount-technology

XI. *Esquemático jtag*,
http://www.dlinkpedia.net/images/hardware/Marven-jtag.jpg

XII. *Tutorial de TinyOS 2.0*,
http://www.tinyos.net/tinyos-2.x/doc/html/tutorial

XIII. *Programación con TinyOS 2.0*,
http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf

XIV. *Resumen de TinyOS*,
http://www.tinyos.net/tinyos-2.x/doc/html/overview.html

XV. *Documentación de TinyOS*,
http://www.tinyos.net/tinyos-2.x/doc

XVI. *Tutorial de lenguaje NesC*,
http://csl.stanford.edu/ pal/pubs/nesc-pldi03.pdf

XVII. *Featured Technology Designing Smart Phone Power Management Systems*,
http://www.wirelessdesignmag.com

XVIII. *Linear Regulators in Portable Applications*,
http://www.maxim-ic.com/appnotes.cfm/appnote_number/751

xix. *Understanding Portable Applications - Requirements to Improve System Performance*, http://www.national.com/appinfo/power/files/PowerDesigner103_portable.pdf

xx. *Antenna Design Considerations*, http://www.seapraha.cz/Download/AntennaDesign.pdf

xxi. *Antennas For Low Power Applications*, http://www.numatechnologies.com/pdf/foilantennas.pdf

xxii. *ISM Evaluation and Development Testboards*, http://www.integration.com/docs/IA_ISM-UGDB.pdf

xxiii. Integration, *Antenna Selection Guide for the IA4220 and IA4320 ISM Band FSK Transmitter/Receiver Chipset*, http://www.integration.com/docs/IA_ISM-AN1.pdf

xxiv. Integration, *Antenna Development Guide for the IA4220 and IA4320 ISM Band FSK Transmitter/Receiver Chipset*, http://www.integration.com/docs/IA_ISM-AN2.pdf

xxv. Berne Institute of Engineering and Architecture, *Smith Chart simulation software (v1.91)*.