



UNIVERSIDAD DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS  
DE TELECOMUNICACIÓN, MENCIÓN EN SISTEMAS  
ELECTRÓNICOS

**Diseño de un sistema de  
comunicación digital via radio para  
entornos industriales**

**Design of a radio digital  
communication system for industrial  
environments**

Autor:

D. Javier Santos Romo

Tutor:

D. Jesús Arias Álvarez

Valladolid, 24 de junio de 2015



---

TÍTULO:	<b>Diseño de un sistema de comunicación digital via radio para entornos industriales</b>
AUTOR:	<b>D. Javier Santos Romo</b>
TUTOR:	<b>D. Jesús Arias Álvarez</b>
DEPARTAMENTO:	<b>Electricidad y Electrónica</b>

---

### **TRIBUNAL**

---

PRESIDENTE:	D. JESÚS HERNÁNDEZ MANGAS
VOCAL:	D. JESÚS ARIAS ÁLVAREZ
SECRETARIO:	DÑA. RUTH PINACHO GÓMEZ
SUPLENTE:	D. IVÁN SANTOS TEJIDO
SUPLENTE:	D. LUIS ALBERTO MARQUÉS CUESTA

---

---

FECHA DE LECTURA: JULIO DE 2015

CALIFICACIÓN:

---

## **Resumen del TFG**

El objetivo del proyecto consiste en el diseño y construcción de un sistema basado en el transceptor de radio de *Texas Instruments* CC2500 para la comunicación de módulos de E/S digitales en entornos industriales y demostrar su viabilidad.

Para poder llevar a cabo esto, se realizará un diseño *hardware* que incluya además del periférico de radio, un microcontrolador para poder gestionar todas las operaciones deseadas. Así, el microcontrolador escogido será uno basado en la arquitectura ARM Cortex-M0 debido a su bajo consumo y su idealidad para dispositivos portátiles, como es el caso.

Tras este diseño se desarrollará el *software* correspondiente para la comunicación fiable via radio entre los módulos remotos y el ordenador personal desde donde se gestionarán todas las operaciones. Para facilitar su usabilidad y portabilidad, se implementará una interfaz gráfica muy intuitiva para la comunicación con los diferentes módulos.

## **Palabras clave**

Microcontrolador, LPC11E13, módulo de radio, CC2500, ARM Cortex-M0, *layout*, diseño *hardware/software*.

## Abstract

The aim of this project is to design and build a system based on *Texas Instruments* CC2500 radio transceiver to communicate digital I/O modules in industrial environments and demonstrate its viability.

In order to perform this project, a *hardware* design will be made consisting of a microcontroller to manage all the required operations, as well as the radio transceiver. Because of this, the selected microcontroller will be one based on ARM-Cortex-M0 architecture due to its low consumption and recommendation for portable devices, as in this case.

Later, the *software* part will be performed to get a reliable communication between remote modules and personal computer where all operations will be managed. For easy use and portability, a very intuitive graphic interface will be designed to communicate the different modules.

## Keywords

Microcontroller, LPC11E13, radio module, CC2500, ARM Cortex-M0, *layout, hardware/software* design.



*A mi familia*





# Agradecimientos

*Quisiera agradecer en primer lugar a toda mi familia su apoyo y paciencia a lo largo de estos cuatro años. Sin ellos no habría llegado tan lejos.*

*Asimismo, quisiera dar las gracias a mi tutor D. Jesús Arias Álvarez por brindarme la oportunidad de realizar este proyecto así como de solventar cualquier duda que surgiera en torno al mismo. Además, quisiera extender mi gratitud a todos los profesores que forman parte de la E.T.S.I. de Telecomunicación por los conocimientos y formación adquiridos.*

*Dar las gracias también a todos aquellos compañeros de universidad con los que he compartido clase, su dedicación y colaboración. Quisiera agradecer especialmente a mis compañeros Diego y Luis Antonio su trabajo y apoyo a lo largo de toda la carrera. Con amigos así todo es mucho más fácil.*

*También quisiera acordarme de Roberto, quien ya no está pero cuya presencia fue importante en el comienzo de todo esto.*

*Por último, quisiera agradecer a todo el mundo que de una forma u otra ha estado implicado en el desarrollo de este trabajo, su paciencia y apoyo.*

*Gracias a todos y a los demás también.*



# Prólogo

A lo largo del presente documento se detallan los pasos que se han ido dando en la consecución del objetivo propuesto, que no es otro que el del diseño e implementación de un sistema de comunicación digital via radio para entornos industriales.

Los sistemas inalámbricos son conocidos por su relativa sencillez y bajo coste frente a los sistemas cableados que requieren de una mayor obra e infraestructura. Sin embargo, sus problemas de interferencias y cobertura hace que tomar una decisión en el diseño de un proyecto no sea tan trivial. Por todo ello, en el *Capítulo 1* expondremos cuales han sido nuestras motivaciones y objetivos para declinarnos por la comunicación via radio.

Tras este primer paso, a continuación toca proceder a la implementación. Para ello, y teniendo en cuenta lo comentado en el *Capítulo 1*, es necesario comprender al detalle los dispositivos que emplearemos en el diseño. Así, en el *Capítulo 2* comentaremos los aspectos teóricos de los principales elementos utilizados.

A continuación, se hace preciso realizar el diseño *hardware* del proyecto. Todos los aspectos de rutado y elaboración de las PCBs serán tratados en el *Capítulo 3*.

Una vez mandadas fabricar las placas de circuito impreso y verificado el correcto funcionamiento de las mismas, es el momento de dotarlas del *firmware* necesario para lograr la comunicación. Este aspecto, así como la elaboración de la aplicación para PC necesaria para poder gobernar los módulos, serán analizados en el *Capítulo 4*.

Con todo el diseño realizado y verificado, será el momento de realizar medidas experimentales (*Capítulo 5*) así como de poner precio a todo el trabajo realizado a través de la elaboración de un presupuesto económico. El *Capítulo 6* contendrá este último aspecto.

Y finalmente, nuestro último paso será recapacitar sobre todo lo elaborado y analizar las posibles mejoras que se puedan aplicar a nuestro diseño. El *Capítulo 7* recogerá estas ideas.

~ *El autor* ~



# Índice de contenidos

<b>1. Introducción.Motivación.Objetivos</b>	<b>23</b>
1.1. Diseño propuesto . . . . .	23
1.2. ¿Por qué via radio? . . . . .	25
<b>2. Fundamento teórico de los CIs empleados</b>	<b>29</b>
2.1. CC2500 . . . . .	29
2.1.1. Descripción del producto . . . . .	29
2.1.2. Características principales . . . . .	30
2.1.3. Funcionamiento interno . . . . .	30
2.1.4. Circuito de aplicación . . . . .	31
2.1.5. Máquina de estados . . . . .	32
2.1.6. Comandos y registros de configuración . . . . .	33
2.1.7. Interfaz de datos serie . . . . .	36
2.1.8. FIFOs de datos . . . . .	38
2.1.9. Pines de propósito general . . . . .	40
2.1.10. Formato de paquete . . . . .	40
2.1.11. Whitening . . . . .	43
2.1.12. Corrección de errores . . . . .	44
2.1.13. Formatos de modulación . . . . .	45
2.1.14. Tasa de transmisión/recepción . . . . .	46
2.1.15. Frecuencia de transmisión . . . . .	46
2.1.16. Potencia de emisión . . . . .	48
2.1.17. SmartRF Studio 7 . . . . .	49
2.2. LPC11E13 . . . . .	53
2.2.1. Descripción del producto . . . . .	53
2.2.2. ARM Cortex-M0 . . . . .	54
2.2.3. Características principales . . . . .	57

2.2.4.	Mapa de memoria . . . . .	59
2.2.5.	Bloque de Control del Sistema . . . . .	62
2.2.6.	Controlador de interrupciones vectorizadas (NVIC) . . . . .	67
2.2.7.	Configuración de pines . . . . .	70
2.2.8.	Entrada/Salida de propósito general (GPIO) . . . . .	72
2.2.9.	Temporizadores . . . . .	75
2.2.10.	Interfaz de periféricos serie (SPI) . . . . .	78
2.2.11.	Interfaz USART . . . . .	81
<b>3.</b>	<b>Implementación hardware del proyecto</b>	<b>85</b>
3.1.	Elección de los componentes . . . . .	85
3.1.1.	Microcontrolador LPC11E13 . . . . .	85
3.1.2.	Módulo de radio ANAREN . . . . .	86
3.1.3.	Convertidor USB-UART FTDI . . . . .	87
3.1.4.	Regulador de tensión TLV70033 . . . . .	89
3.2.	Elaboración de los esquemas electrónicos . . . . .	91
3.2.1.	Red de conversión USB-UART . . . . .	91
3.2.2.	Red de regulación de tensión . . . . .	93
3.2.3.	Red del microcontrolador y módulo de radio . . . . .	94
3.2.4.	Listado de materiales . . . . .	96
3.3.	Layout de la PCB . . . . .	97
3.3.1.	Requerimientos en el diseño de la PCB . . . . .	97
3.3.2.	Rutado y placement de los componentes . . . . .	98
3.3.3.	Pedido al fabricante . . . . .	100
<b>4.</b>	<b>Diseño software del proyecto</b>	<b>101</b>
4.1.	Protocolo empleado en la comunicación . . . . .	101
4.1.1.	Formatos de trama empleados . . . . .	104
4.2.	Implementación de la lógica del esclavo . . . . .	105
4.2.1.	El problema de la dirección . . . . .	106
4.2.2.	El problema del consumo . . . . .	110
4.3.	Programación de los módulos . . . . .	112
4.4.	Programa para PC . . . . .	113
4.4.1.	wxDev-C++ . . . . .	114
4.4.2.	Manual de usuario . . . . .	115

---

<b>5. Resultados experimentales</b>	<b>119</b>
5.1. Diseño realizado . . . . .	119
5.2. Consumo de potencia . . . . .	122
5.3. Rango de cobertura . . . . .	123
<b>6. Presupuesto económico</b>	<b>125</b>
6.1. Costes materiales . . . . .	125
6.2. Costes personales . . . . .	127
6.3. Coste total . . . . .	129
<b>7. Conclusiones y líneas futuras</b>	<b>131</b>
<b>A. Conjunto de instrucciones del procesador Cortex-M0</b>	<b>133</b>
<b>B. Esquemáticos del diseño realizado</b>	<b>137</b>
<b>C. Layouts del diseño realizado</b>	<b>141</b>
<b>D. Código del módulo maestro</b>	<b>145</b>
<b>E. Código del módulo esclavo</b>	<b>163</b>
<b>F. Código de la aplicación para PC</b>	<b>185</b>
<b>Índice terminológico</b>	<b>217</b>
<b>Bibliografía</b>	<b>219</b>



# Índice de figuras

1.1. Topología del diseño a realizar . . . . .	24
2.1. Diagrama de bloques del integrado CC2500 . . . . .	31
2.2. Circuito de aplicación del integrado CC2500 . . . . .	32
2.3. Diagrama de estados del integrado CC2500 . . . . .	33
2.4. Configuración de las operaciones de lectura/escritura . . . . .	37
2.5. Formato de paquete . . . . .	41
2.6. Whitening . . . . .	43
2.7. Interleaving . . . . .	44
2.8. Elección de dispositivo en SmartRF Studio 7 . . . . .	51
2.9. Panel de configuración en SmartRF Studio 7 . . . . .	52
2.10. Elección de formato de salida en SmartRF Studio 7 . . . . .	53
2.11. Periféricos que componen el microprocesador Cortex-M0 . . . . .	55
2.12. Conjunto de registros del microprocesador Cortex-M0 . . . . .	56
2.13. Diagrama de bloques del microcontrolador LPC11E13FBD48/301 . . . . .	60
2.14. Mapa de memoria de la familia LPC11E1x . . . . .	61
2.15. Unidad generadora del reloj . . . . .	63
2.16. Esquema de Reset del microcontrolador LPC11E13 . . . . .	66
2.17. Tabla de vectores de interrupción del microcontrolador LPC11E13 . . . . .	68
2.18. Encapsulado del microcontrolador LPC11E13 . . . . .	70
2.19. Configuración estándar I/O de los pines del microcontrolador LPC11E13 . . . . .	71
2.20. Diagrama de bloques de los temporizadores del microcontrolador LPC11E13 . . . . .	77
2.21. Transferencia SPI en el microcontrolador LPC11E13 . . . . .	80
2.22. Diagrama de bloques del módulo USART en el microcontrolador LPC11E13 . . . . .	82
2.23. Transmisión del módulo USART en el microcontrolador LPC11E13 . . . . .	84
3.1. Familia ARM Cortex-M . . . . .	86
3.2. Diagrama de bloques del módulo de radio A2500R24A . . . . .	87

3.3.	Diagrama de bloques del integrado FTDI232RL . . . . .	89
3.4.	Diagrama de bloques del regulador de tensión TLV70033 . . . . .	90
3.5.	Red de alimentación y circuito de conversión USB-UART . . . . .	92
3.6.	Red de regulación de tensión . . . . .	94
3.7.	Red de comunicación externa y via radio . . . . .	95
4.1.	Solución de errores en ARQ de parada y espera . . . . .	103
4.2.	Formato de las tramas intercambiadas . . . . .	104
4.3.	WOR gestionado por el microcontrolador . . . . .	112
4.4.	Captura de pantalla del IDE wxDev-C++ . . . . .	115
4.5.	Captura de pantalla de la interfaz gráfica desarrollada . . . . .	116
4.6.	Partes de la interfaz gráfica desarrollada . . . . .	117
5.1.	Módulo transceptor completo <i>Top layer</i> . . . . .	120
5.2.	Módulo transceptor completo <i>Bottom layer</i> . . . . .	120
5.3.	Módulo de radio ANAREN conectado a la placa <i>Education Board</i> . . . . .	122
5.4.	Consumo experimental del módulo de radio maestro . . . . .	123
B.1.	Esquemático correspondiente al módulo de radio . . . . .	138
B.2.	Esquemático correspondiente a la red de alimentación y conversión . . . . .	139
B.3.	Esquemático correspondiente al consumo de corriente estimado . . . . .	140
C.1.	Placa PCB realizada: <i>Top copper</i> . . . . .	141
C.2.	Placa PCB realizada: <i>Top mask</i> . . . . .	142
C.3.	Placa PCB realizada: <i>Bottom copper</i> . . . . .	142
C.4.	Placa PCB realizada: <i>Top resist</i> . . . . .	143
C.5.	Placa PCB realizada: <i>Top silk</i> . . . . .	143
C.6.	Placa PCB realizada: <i>Mechanic and drill holes</i> . . . . .	144

# Índice de tablas

1.1. Comparativa de redes cableadas & redes inalámbricas . . . . .	26
2.1. Comandos del módulo CC2500 . . . . .	34
2.2. Registros de configuración del módulo CC2500 . . . . .	35
2.3. Registros de estado del módulo CC2500 . . . . .	36
2.4. Resumen del byte de estado . . . . .	38
2.5. Espacio de direcciones SPI del módulo CC2500 . . . . .	39
2.6. Tasas de transmisión posibles . . . . .	47
2.7. Canales Wi-Fi en España para los estándares 802.11b y 802.11g . . . . .	48
2.8. Canales Bluetooth disponibles en el estándar 802.15 . . . . .	49
2.9. Niveles de potencia de RF . . . . .	50
2.10. Fuentes de interrupción del microcontrolador LPC11E13 . . . . .	69
3.1. Listado de materiales empleados en la aplicación . . . . .	97
4.1. Comandos implementados en la aplicación . . . . .	105
5.1. Cobertura y número de errores a +1dBm de potencia de tx y 250kBaudios para órdenes individuales . . . . .	124
6.1. Costes de los componentes electrónicos . . . . .	126
6.2. Costes de fabricación y montaje . . . . .	127
6.3. Costes de los ingenieros . . . . .	127
6.4. Parte de trabajo realizado y coste asociado . . . . .	129
6.5. Costes totales del proyecto realizado . . . . .	129
A.1. Conjunto de instrucciones del procesador Cortex-M0 . . . . .	135



# Índice de códigos

4.1. Programa encargado de proporcionar la dirección a un módulo . . . . .	107
4.2. Parámetros de configuración del microcontrolador LPC11E13FBD48/301	112
4.3. Fichero Makefile empleado en la programación . . . . .	113
D.1. Fichero de inicialización del módulo maestro . . . . .	145
D.2. Programa principal del módulo maestro . . . . .	148
E.1. Fichero de inicialización de los módulos esclavos . . . . .	163
E.2. Programa principal de los módulos esclavos . . . . .	166
F.1. Fichero de cabecera de la aplicación para PC . . . . .	185
F.2. Programa principal de la aplicación para PC . . . . .	191



# Capítulo 1

## Introducción.Motivación.Objetivos

*The best way to get started is quit talking and begin doing.*

~ Walt Disney ~

A la hora de llevar a cabo un proyecto de cualquier índole, siempre se hace necesario detallar rigurosamente el objetivo que se persigue, así como la motivación que ello nos supone. Por este motivo, en las siguientes secciones se detallan estos aspectos clave en el arranque de cualquier diseño.

### 1.1. Diseño propuesto

En este trabajo fin de grado se pretende llevar a cabo el diseño y la implementación de un sistema de comunicación digital via radio para entornos industriales. El objetivo no es otro que el de permitir la posibilidad de poder comunicar las diferentes máquinas y mecanismos que puedan encontrarse en la industria con un ordenador central que lleve un registro de todas las operaciones que tienen lugar en cada momento. Para poder conseguir esto, los diferentes artilugios que se quieran monitorizar irán provistos, cada uno de ellos, de un módulo de radio esclavo capaz de atender órdenes. Por otro lado, el ordenador central contará con un módulo de radio maestro capaz de transmitir dichas órdenes. Para facilitar la comunicación entre el ordenador y el módulo de radio maestro, se elaborará una aplicación *software* gráfica muy intuitiva capaz de abstraer cualquier conocimiento relativo a cómo tienen lugar las comunicaciones. La topología de dicho proyecto se puede observar en la figura 1.1.

El modo de funcionamiento que se pretende conseguir es el siguiente:

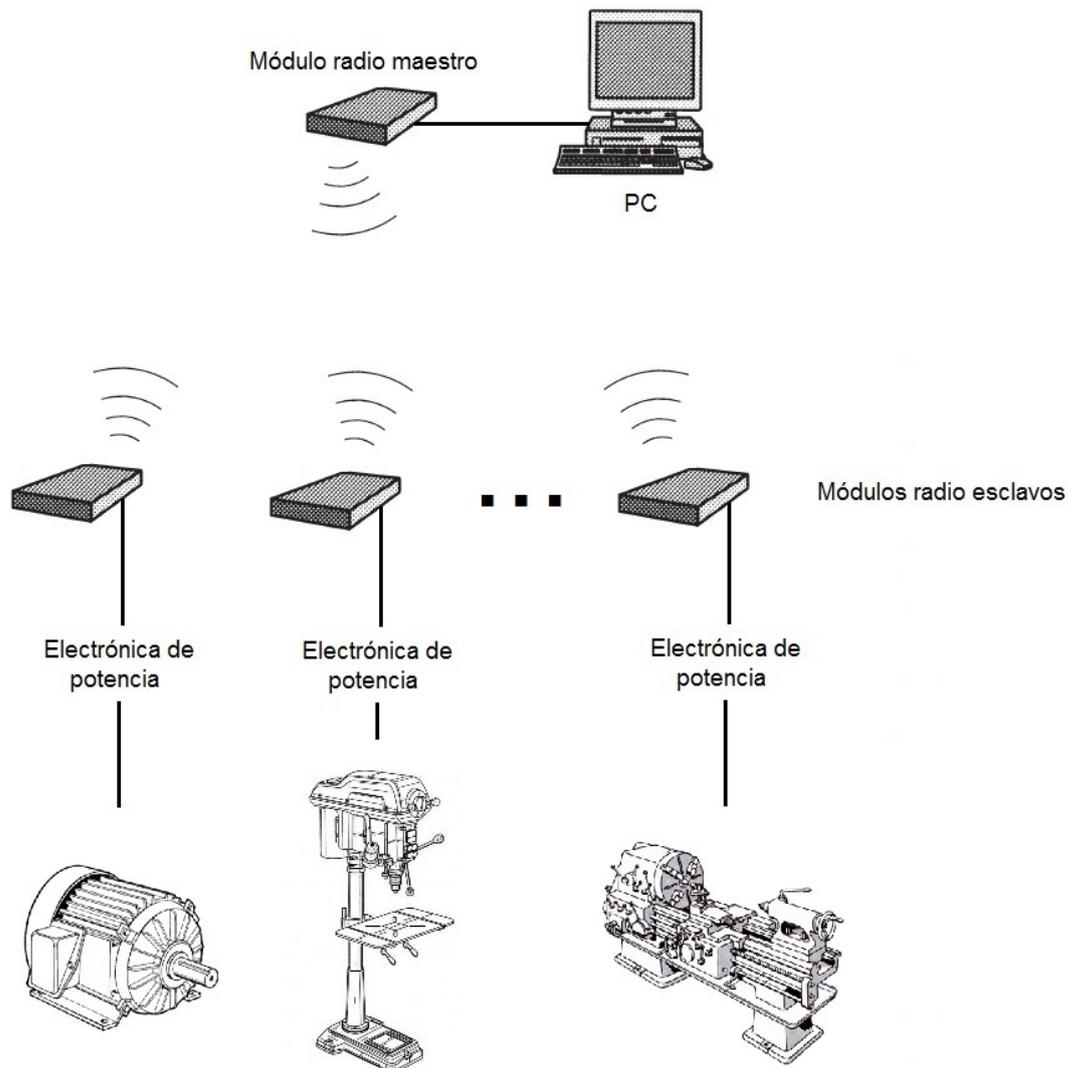


Figura 1.1: Topología del diseño a realizar

1. Cuando el operario encargado de controlar las instalaciones desea realizar alguna acción, solicita ésta a través del *software* del PC.
2. Después la aplicación del PC se encarga de comunicar dicha orden al módulo de radio maestro.
3. El módulo de radio maestro comunica la orden al módulo de radio esclavo al que va dirigida.
4. A continuación el módulo de radio esclavo recibe la petición e interactúa con la máquina a la que se encuentra conectado.

5. Después el módulo de radio esclavo confirma la realización de la acción enviando una respuesta al módulo de radio maestro.
6. Finalmente, el módulo de radio maestro traslada la respuesta al PC para que el operario pueda comprobar que la acción se ha realizado satisfactoriamente.

Un último aspecto a comentar respecto al diseño, reside en que la implementación de la electrónica de potencia de que consta cada módulo esclavo queda fuera de los objetivos de este TFG. De este modo, la viabilidad del sistema desarrollado quedará demostrada con el control de las entradas y salidas digitales de que constan los módulos. De hecho, es esta parte la que engloba todas las dificultades y donde reside el mayor esfuerzo a realizar. La necesidad de electrónica de potencia simplemente responde, como su propio nombre indica, a una cuestión de balances de potencias, puesto que no podemos pretender alimentar desde un módulo que opera con corrientes de miliamperios grandes motores, ya sean monofásicos o trifásicos, que consumen decenas o centenas de amperios. Así, dicha electrónica de potencia (relés, MOSFET de potencia, etc.) nos permite controlar con niveles de tensión y corrientes bajos de los módulos, circuitos de alta tensión y corriente que atacan a la maquinaria industrial. De este modo, además de lograr el objetivo, se garantiza también la seguridad, puesto que la zona de alta potencia queda aislada de la electrónica y así se evitan posibles riesgos de daños personales y materiales.

## 1.2. ¿Por qué via radio?

Como ha quedado patente en la sección anterior, la parte crítica de nuestro sistema se concentra en la comunicación entre los módulos maestro y esclavo. Para llevar a cabo su implementación, dos son las grandes filosofías de diseño que podríamos aplicar, i.e. cableada o inalámbrica. Cada una de ellas presenta unas ventajas e inconvenientes que no están exclusivamente ligadas a la propia tecnología en sí, sino que también depende de cada situación concreta. En la tabla 1.1 se pueden contemplar las principales características achacables a cada una de las tecnologías.

Como se puede observar en dicha tabla resumen, parece concluirse que declinarse por un sistema cableado implica una mayor inversión a costa de unas buenas prestaciones, mientras que optar por un sistema inalámbrico supone ahorrar costes en perjuicio de la calidad final obtenida. Sin embargo, esto no es del todo cierto hasta que uno analiza las prestaciones del diseño que pretende realizar.

Así pues, si retomamos ahora nuestro proyecto, podemos extraer varias conclusiones respecto a las características de que se disponen, a saber:

	<b>Sistema cableado</b>	<b>Sistema inalámbrico</b>
<b>Instalación</b>	(✗) Lenta y con cierta complejidad al requerir obras.	(✓) Rápida y sencilla, sin apenas obras ni mantenimiento.
<b>Coste</b>	(✗) Alto, requiere una inversión en una red de cables.	(✓) Bajo, apenas se necesitan elementos.
<b>Movilidad y crecimiento de la red</b>	(✗) Mala, expandir o cambiar el sistema significa tener que realizar obras de nuevo.	(✓) Excelente, no se precisa de ningún tipo de cambio físico en la red.
<b>Alcance y cobertura</b>	(✓) Bueno, la señal transmitida viaja confinada en un cable.	(✗) Regular, la potencia de emisión se reparte por el espacio.
<b>Fiabilidad</b>	(✓) Alta, al tratarse de un medio guiado las interferencias son menores.	(✗) Regular, la compartición del espectro con diversas tecnologías provoca interferencias.
<b>Tasa de transmisión</b>	(✓) Elevada, limitada por componentes parásitas.	(✗) Normal, limitada por anchura espectral y consumo de potencia.

Tabla 1.1: Comparativa de redes cableadas & redes inalámbricas

- Sería interesante un diseño lo más barato posible (no requiere justificación), fácil de instalar y mantener, y que permitiese aumentar la escalabilidad del sistema de una manera rápida y sencilla.
- Si bien se requiere de una buena cobertura y alcance, lo más probable es que las diferentes máquinas a controlar de un mismo tipo se encuentren en una misma sala.
- La fiabilidad es una característica crucial, ya que necesitamos estar seguros de que las diferentes órdenes emitidas han llegado a su destino correctamente.
- Teniendo en cuenta el propósito de la aplicación, no se precisa de una tasa de transmisión elevada (¿Cuántas orden puede requerir solicitar un operario de un conjunto de máquinas, dos o tres cada cuarto de hora?).

En vista de estas necesidades de nuestro diseño, parece ser que las prestaciones que mejor se adaptan a las mismas corresponden a las de los sistemas inalámbricos. Así, en primer lugar, estos sistemas son los más económicos. Por otro lado, una elevada cobertura no es tan necesaria en vistas de lo dicho anteriormente. Además, en caso de existir diferentes salas distantes, mejor que incrementar la cobertura, sería disponer de otro sistema completo PC - módulo maestro - módulos esclavo. Si bien esta solución no supone un incremento elevado del coste (se requiere un PC y un módulo maestro adicionales) permite solucionar el problema de la cobertura así como lograr una descentralización del sistema,

lo cual puede resultar muy recomendable en caso de producirse un fallo en el sistema para que no se vea afectado todo él. Esto último sería más costoso de emplearse la solución cableada.

El aspecto de la fiabilidad, por otra parte, se puede conseguir a través del empleo de un protocolo de comunicaciones entre los diferentes módulos que garantice la correcta recepción de los mensajes. De esta forma, aunque el sistema inalámbrico es más propenso a las interferencias, el protocolo nos asegurará que las órdenes serán ejecutadas sin pérdidas de información. De haber optado por la solución cableada, salvo el uso dedicado de líneas entre cada módulo esclavo y el maestro que no requeriría de ningún protocolo aunque sí de un coste aún mayor, un protocolo sería igualmente necesario, pues los diferentes módulos estarían conectados a un bus y por tanto a un medio compartido.

Finalmente, el hecho de que no se precise de una tasa de transmisión elevada encaja perfectamente con las características que el sistema inalámbrico proporciona.

Por todo lo mencionado en esta sección queda claro que la comunicación inalámbrica es la mejor opción para nuestro proyecto, si bien plantea una serie de retos que será necesario resolver para la consecución de los objetivos propuestos.



# Capítulo 2

## Fundamento teórico de los CIs empleados

*In learning, like in life, more important than  
play for time, is definitely, lose it.  
~ Jean-Jacques Rousseau ~*

En las siguientes secciones se pretende recoger los principales fundamentos teóricos que explican el funcionamiento de los dos circuitos integrados clave en el diseño realizado. De esta forma, se pretende dar una visión detallada del modo de trabajo del transceptor CC2500, atendiendo especialmente a la configuración que le ha sido otorgada en el diseño propuesto; y del microcontrolador LPC11E13, analizando el conjunto de periféricos que han sido utilizados para poder llevar a cabo exitosamente el problema propuesto.

### 2.1. CC2500

#### 2.1.1. Descripción del producto

El módulo CC2500 (*Texas Instruments*) es un transceptor de bajo coste que opera en la banda de 2.4 GHz y que ha sido especialmente diseñado para aplicaciones inalámbricas que requieren de relativamente poca potencia. El integrado en cuestión ha sido desarrollado para operar en las bandas de frecuencias denominadas ISM (*Industrial, Scientific and Medical*) y SRD (*Short Range Device*).

Este transceptor de RF soporta varios formatos de modulación y puede proporcionar una tasa de datos de hasta 500 kBaudios. Además destaca por disponer de dos *buffers* FIFO (*First In First Out*) de transmisión y recepción de 64 bytes y todo un soporte *hardware* para soportar la generación de paquetes de datos.

En una configuración típica, el integrado CC2500 será usado junto con un microcontro-

lador y controlado vía una interfaz SPI (*Serial Peripheral Interface*) donde el módulo de radio jugará el papel de esclavo y el microcontrolador el de maestro.

### 2.1.2. Características principales

Las características más destacables del transceptor CC2500 son las siguientes [1]:

- Alta sensibilidad (-104 dBm a 2.4 kBaudios)
- Tasa de datos programable desde 1.2 a 500 kBaudios
- Potencia de salida programable hasta 1 dBm
- Modulaciones soportadas: OOK (*On-Off Keying*), 2-FSK (*Frequency Shift Keying*), MSK (*Minimum Shift Keying*) y GFSK (*Gaussian Frequency Shift Keying*).
- Soporte para la generación y detección automática de Preámbulos, Palabras de sincronización (*Sync Word*), Direcciones y CRC (*Cyclic Redundancy Check*).
- Dispone de 47 registros de configuración y 12 de estado, todos ellos de 8 bit.
- Eficiente interfaz SPI que permite la programación de todos los registros con una sola ráfaga (*burst*) de datos.
- Consumo de corriente de 400 nA en modo SLEEP.
- Cumple con las normativas europeas EN 300 328 y EN 300 440 y con la normativa RoHS.

### 2.1.3. Funcionamiento interno

Un diagrama de bloques simplificado del integrado CC2500 se puede observar en la figura 2.1.

En dicha figura podemos observar que el transceptor consta de una arquitectura homodina mediante la cual, la señal de radiofrecuencia proveniente del exterior se convierte en banda base en una sola etapa de mezclado. Así, en primer lugar, la señal recibida es amplificada con un amplificador de bajo ruido (LNA, *Low Noise Amplifier*). La misión de dicho amplificador es la de reducir el ruido de la señal, contribuyendo especialmente a mejorar (reducir) el ruido de toda la cadena receptora, así como de proporcionar un rechazo de la frecuencia imagen. Seguidamente, la señal sufre un batido en frecuencia gracias a dos mezcladores en cuadratura que proporcionan a su salida dos señales idénticas

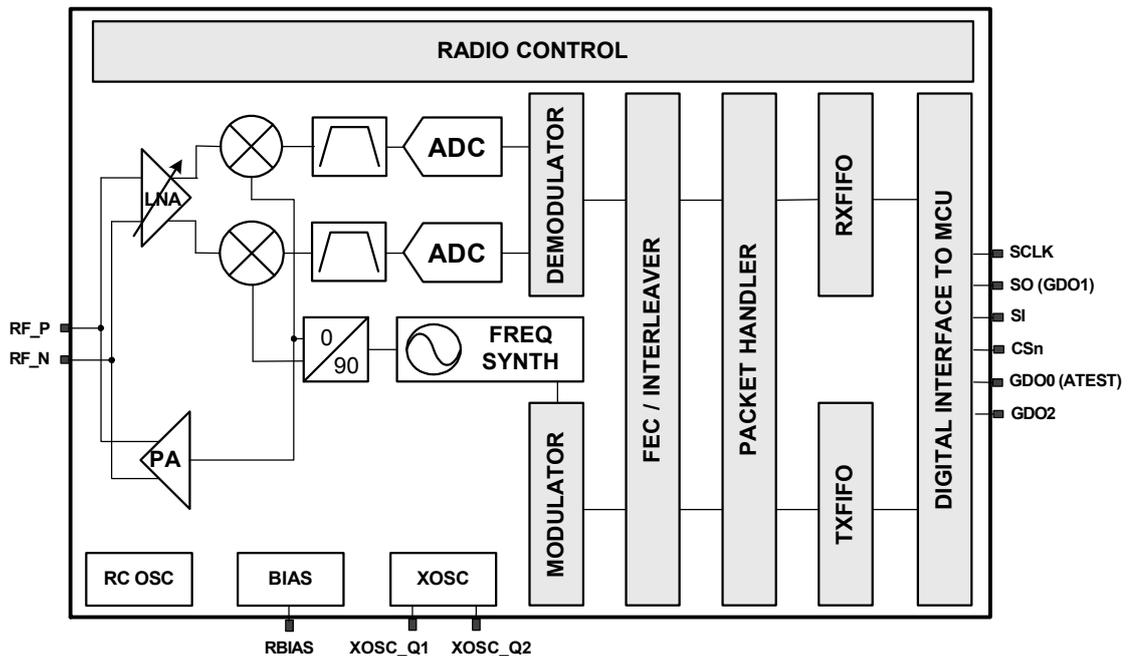


Figura 2.1: Diagrama de bloques del integrado CC2500

pero desfasadas  $90^\circ$ . De esta forma, dichas señales, tras su conversión analógica a digital (ADC, *Analog to Digital Converter*), pueden ser fácilmente demoduladas, y los datos a su salida colocados en la FIFO de recepción.

El funcionamiento como transmisor vuelve a ser de nuevo una cadena homodina. De esta forma, la señal digital banda base es modulada a radiofrecuencia gracias a un sintetizador de frecuencia *on chip* que consta de un VCO (*Voltage Controlled Oscillator*). Una vez modulada, la señal se hace pasar por un amplificador de potencia para poder alcanzar mayores distancias.

#### 2.1.4. Circuito de aplicación

El circuito de aplicación recomendado para el integrado CC2500, y cuyo esquema se ha seguido en el diseño realizado, se puede observar en la figura 2.2.

Analizando dicho circuito, en primer lugar nos encontramos con que se precisa de un cristal de cuarzo y sus correspondientes condensadores de carga para poder generar la frecuencia de referencia para el sintetizador, así como los relojes para el ADC y la parte digital.

Por otra parte se requiere una resistencia en el pin 17 para poder desarrollar una precisa corriente continua que emplea la lógica del integrado, y un condensador de desacoplo en el pin 5 para el regulador de voltaje de la parte digital integrado en el *chip*.

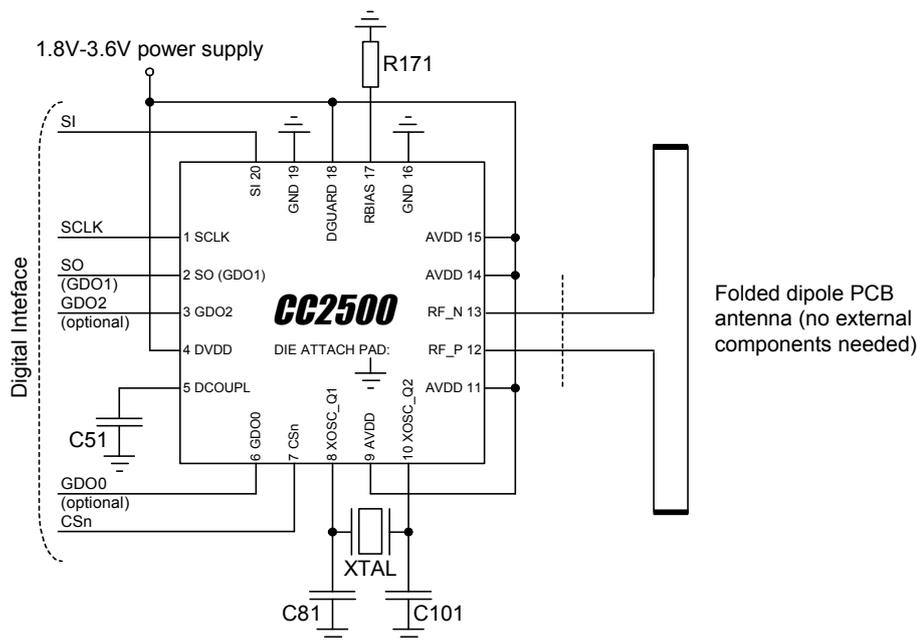


Figura 2.2: Circuito de aplicación del integrado CC2500 sin condensadores de desacoplo

Por otro lado se precisa disponer de una adecuada antena en los pines 12 y 13 para poder transmitir y recibir las señales. En este sentido, en el diseño realizado el integrado cuenta con una antena de tipo parche que ha sido realizada empleando una pista de la PCB. Para poder llevar a cabo su realización, ha sido necesario disponer de un circuito balun (*balanced-unbalanced*) para poder convertir la señal diferencial que sale del integrado a una señal de radiofrecuencia *single-ended*, y de la adecuada red LC para la transformación de la impedancia a una igual a la de la impedancia característica de la pista de la PCB. Finalmente, y aunque en la figura 2.2 no aparecen explícitamente, son necesarios condensadores de desacoplo entre cada pin conectado a la alimentación (para proporcionar la potencia a la lógica digital y analógica) y tierra. Dichos condensadores juegan un papel fundamental en el correcto funcionamiento del sistema. Así, su cometido principal consiste en desviar los pulsos de corriente que se generan en la lógica digital debido a la conmutación hacia tierra, y que no provoquen una caída de tensión en la línea de alimentación que podría afectar a todos los dispositivos conectados a la misma.

### 2.1.5. Máquina de estados

En la figura 2.3 se puede observar la máquina de estados que implementa el integrado. Para cambiar entre los diferentes estados se emplean comandos (*strokes*) e.g. STX, aunque también pueden darse una serie de eventos internos que hacen pasar de un estado a otro

e.g. TX FIFO Underflow.

Los posibles comandos, así como el modo de hacérselos llegar al integrado CC2500, serán tratados en las siguientes subsecciones.



Figura 2.3: Diagrama de estados completo del integrado CC2500

### 2.1.6. Comandos y registros de configuración

La configuración del integrado CC2500 se realiza programando registros de 8 bit de ancho de palabra. Después de producirse un *reset* en el chip, todos estos registros contie-

nen valores por defecto, y en ningún caso conservan su anterior configuración. De igual modo, el estado del módulo de radio pasa a ser IDLE. Por tanto, se hace necesario programar, cada vez que este fenómeno se produzca, todos los registros con sus adecuados valores y suministrar al integrado los comandos necesarios para alcanzar el estado que se requiera en cada momento.

Comenzando por los comandos, éstos están formados por 13 posibles órdenes que tienen su correspondiente dirección dentro del mapa de registros del CC2500. Accediendo a estos registros conseguiremos cambiar el estado en que se encuentra el integrado. En la figura 2.1 se recoge una tabla con todos los comandos y su significado.

Address	Strobe Name	Description
0x30	SRES	Reset chip.
0x31	SFSTXON	Enable and calibrate frequency synthesizer (if <code>MCSM0.FS_AUTOCAL=1</code> ). If in RX (with CCA): Go to a wait state where only the synthesizer is running (for quick RX / TX turnaround).
0x32	SXOFF	Turn off crystal oscillator.
0x33	SCAL	Calibrate frequency synthesizer and turn it off. <code>SCAL</code> can be strobed from IDLE mode without setting manual calibration mode ( <code>MCSM0.FS_AUTOCAL=0</code> )
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and <code>MCSM0.FS_AUTOCAL=1</code> .
0x35	STX	In IDLE state: Enable TX. Perform calibration first if <code>MCSM0.FS_AUTOCAL=1</code> . If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x38	SWOR	Start automatic RX polling sequence (Wake-on-Radio) as described in Section 19.5 if <code>WORCTRL.RC_PD=0</code> .
0x39	SPWD	Enter power down mode when <code>CSn</code> goes high.
0x3A	SFRX	Flush the RX FIFO buffer. Only issue <code>SFRX</code> in IDLE or <code>RXFIFO_OVERFLOW</code> states.
0x3B	SFTX	Flush the TX FIFO buffer. Only issue <code>SFTX</code> in IDLE or <code>TXFIFO_UNDERFLOW</code> states.
0x3C	SWORRST	Reset real time clock to Event1 value.
0x3D	SNOP	No operation. May be used to get access to the chip status byte.

Tabla 2.1: Comandos del integrado CC2500

En lo que respecta al conjunto de registros del módulo de radio, éstos comprenden 47 registros de configuración de los que no todos necesitan ser programados, y 12 registros de estado que son solamente de lectura. Las figuras 2.2 y 2.3 muestran respectivamente estos registros.

Address	Register	Description	Preserved in SLEEP State
0x00	IOCFG2	GDO2 output pin configuration	Yes
0x01	IOCFG1	GDO1 output pin configuration	Yes
0x02	IOCFG0	GDO0 output pin configuration	Yes
0x03	FIFOTHR	RX FIFO and TX FIFO thresholds	Yes
0x04	SYNC1	Sync word, high byte	Yes
0x05	SYNC0	Sync word, low byte	Yes
0x06	PKTLEN	Packet length	Yes
0x07	PKTCTRL1	Packet automation control	Yes
0x08	PKTCTRL0	Packet automation control	Yes
0x09	ADDR	Device address	Yes
0x0A	CHANNR	Channel number	Yes
0x0B	FSCTRL1	Frequency synthesizer control	Yes
0x0C	FSCTRL0	Frequency synthesizer control	Yes
0x0D	FREQ2	Frequency control word, high byte	Yes
0x0E	FREQ1	Frequency control word, middle byte	Yes
0x0F	FREQ0	Frequency control word, low byte	Yes
0x10	MDMCFG4	Modem configuration	Yes
0x11	MDMCFG3	Modem configuration	Yes
0x12	MDMCFG2	Modem configuration	Yes
0x13	MDMCFG1	Modem configuration	Yes
0x14	MDMCFG0	Modem configuration	Yes
0x15	DEVIATN	Modem deviation setting	Yes
0x16	MCSM2	Main Radio Control State Machine configuration	Yes
0x17	MCSM1	Main Radio Control State Machine configuration	Yes
0x18	MCSM0	Main Radio Control State Machine configuration	Yes
0x19	FOCCFG	Frequency Offset Compensation configuration	Yes
0x1A	BSCFG	Bit Synchronization configuration	Yes
0x1B	AGCTRL2	AGC control	Yes
0x1C	AGCTRL1	AGC control	Yes
0x1D	AGCTRL0	AGC control	Yes
0x1E	WOREVT1	High byte Event 0 timeout	Yes
0x1F	WOREVT0	Low byte Event 0 timeout	Yes
0x20	WORCTRL	Wake On Radio control	Yes
0x21	FREND1	Front end RX configuration	Yes
0x22	FREND0	Front end TX configuration	Yes
0x23	FSCAL3	Frequency synthesizer calibration	Yes
0x24	FSCAL2	Frequency synthesizer calibration	Yes
0x25	FSCAL1	Frequency synthesizer calibration	Yes
0x26	FSCAL0	Frequency synthesizer calibration	Yes
0x27	RCCTRL1	RC oscillator configuration	Yes
0x28	RCCTRL0	RC oscillator configuration	Yes
0x29	FSTEST	Frequency synthesizer calibration control	No
0x2A	PTEST	Production test	No
0x2B	AGCTEST	AGC test	No
0x2C	TEST2	Various test settings	No
0x2D	TEST1	Various test settings	No
0x2E	TEST0	Various test settings	No

Tabla 2.2: Registros de configuración del integrado CC2500

Address	Register	Description	Details on Page Number
0x30 (0xF0)	PARTNUM	<b>CC2500</b> part number	81
0x31 (0xF1)	VERSION	Current version number	81
0x32 (0xF2)	FREQEST	Frequency offset estimate	81
0x33 (0xF3)	LQI	Demodulator estimate for Link Quality	82
0x34 (0xF4)	RSSI	Received signal strength indication	82
0x35 (0xF5)	MARCSTATE	Control state machine state	82
0x36 (0xF6)	WORTIME1	High byte of WOR timer	83
0x37 (0xF7)	WORTIME0	Low byte of WOR timer	83
0x38 (0xF8)	PKTSTATUS	Current GDOx status and packet status	83
0x39 (0xF9)	VCO_VC_DAC	Current setting from PLL calibration module	83
0x3A (0xFA)	TXBYTES	Underflow and number of bytes in the TX FIFO	83
0x3B (0xFB)	RXBYTES	Overflow and number of bytes in the RX FIFO	84
0x3C (0xFC)	RCCTRL1_STATUS	Last RC oscillator calibration result	84
0x3D (0xFD)	RCCTRL0_STATUS	Last RC oscillator calibration result	84

Tabla 2.3: Registros de estado del integrado CC2500

### 2.1.7. Interfaz de datos serie

Todas las transacciones de información con el integrado CC2500 se realizan gracias a una interfaz SPI (*Serial Peripheral Interface*) en la que tiene el rol de esclavo, mientras que el maestro será un microcontrolador. Dicha interfaz consta de 4 líneas:

- SCLK (*Serial Clock*): Es la línea del reloj. Dado que el módulo de radio es un esclavo, este pin será una entrada.
- SI (*Serial Input*): Es la línea de entrada de datos y/o comandos al integrado.
- SO (*Serial Output*): Es la línea de salida de datos del integrado.
- CSn (*Chip Select*): Es la línea de selección de esclavo. Debe ser una entrada en el módulo de radio, y cuando su nivel lógico es bajo el integrado CC2500 puede comenzar la comunicación.

Todas las transacciones sobre la interfaz SPI comienzan con un byte de cabecera conteniendo un bit de lectura/escritura (R/W), un bit de acceso por ráfaga (B, *burst*) y una dirección de 6 bits (A6-A0) que se emplea para acceder a una dirección concreta del módulo de radio.

Un ejemplo de dicho intercambio de información se puede observar en la figura 2.4.

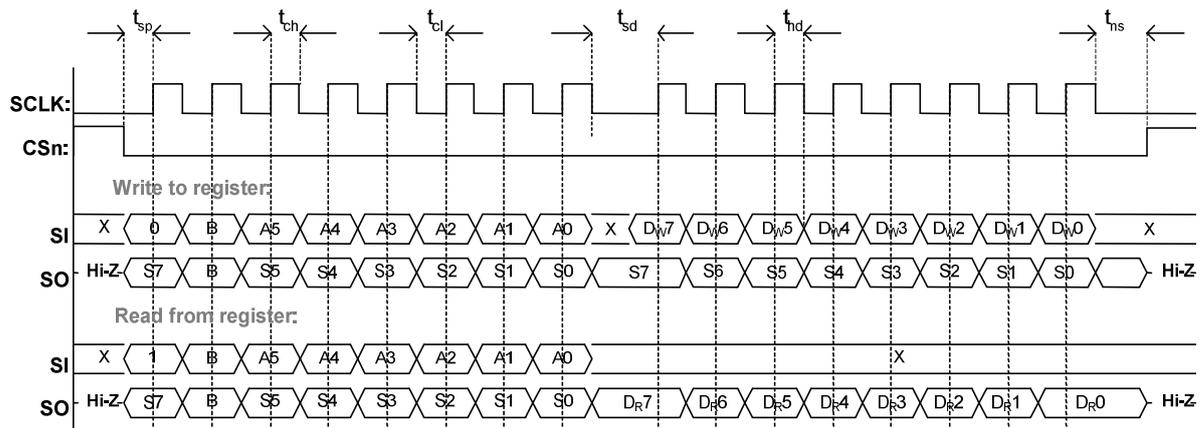


Figura 2.4: Configuración de las operaciones de lectura/escritura a través de la interfaz SPI

El proceso que se debe seguir para poder intercambiar información con el módulo de radio es el siguiente:

- En primer lugar, antes de iniciar cualquier tipo de comunicación, se debe empujar la línea CSn a nivel bajo y esperar hasta que el pin SO vaya a nivel bajo también. Este último hecho indica que el cristal de cuarzo ha alcanzado la frecuencia estable y el integrado CC2500 está preparado para recibir órdenes.
- A continuación se debe colocar el byte de cabecera sobre la línea SI, con el adecuado valor en los bits R/W y B en función de la operación que se quiera realizar, y el valor de la dirección a la que se pretende acceder.
- Una vez enviado dicho byte, si fuera necesario enviar información adicional ésta se puede enviar siempre de byte en byte a continuación de la cabecera. Finalmente, cuando se desea finalizar la comunicación o interrumpirla, se debe poner de nuevo la línea CSn en nivel lógico alto.

Un aspecto relevante que se puede observar en la figura 2.4 es que tanto en la lectura como en la escritura, una vez que se ha enviado el byte de cabecera al módulo de radio, éste envía hacia el microcontrolador un byte de estado. Dicho byte contiene información sobre diferentes señales así como del estado en que se encuentra el CC2500 en ese momento. La descripción detallada de la información de este byte se puede observar en la tabla 2.4.

Volviendo de nuevo al *byte* de cabecera, conviene explicar algo más en detalle en qué consiste el modo ráfaga. Básicamente, lo que dicho modo permite es acceder de manera consecutiva a diferentes direcciones de memoria tanto para leer como para escribir. Así, lo

Bits	Name	Description																											
7	CHIP_RDYn	Stays high until power and crystal have stabilized. Should always be low when using the SPI interface.																											
6:4	STATE[2:0]	Indicates the current main state machine mode <table border="1"> <thead> <tr> <th>Value</th> <th>State</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>IDLE</td> <td>Idle state (Also reported for some transitional states instead of SETTLING or CALIBRATE)</td> </tr> <tr> <td>001</td> <td>RX</td> <td>Receive mode</td> </tr> <tr> <td>010</td> <td>TX</td> <td>Transmit mode</td> </tr> <tr> <td>011</td> <td>FSTXON</td> <td>Frequency synthesizer is on, ready to start transmitting</td> </tr> <tr> <td>100</td> <td>CALIBRATE</td> <td>Frequency synthesizer calibration is running</td> </tr> <tr> <td>101</td> <td>SETTLING</td> <td>PLL is settling</td> </tr> <tr> <td>110</td> <td>RXFIFO_OVERFLOW</td> <td>RX FIFO has overflowed. Read out any useful data, then flush the FIFO with <code>SFRX</code></td> </tr> <tr> <td>111</td> <td>TXFIFO_UNDERFLOW</td> <td>TX FIFO has underflowed. Acknowledge with <code>SFTX</code></td> </tr> </tbody> </table>	Value	State	Description	000	IDLE	Idle state (Also reported for some transitional states instead of SETTLING or CALIBRATE)	001	RX	Receive mode	010	TX	Transmit mode	011	FSTXON	Frequency synthesizer is on, ready to start transmitting	100	CALIBRATE	Frequency synthesizer calibration is running	101	SETTLING	PLL is settling	110	RXFIFO_OVERFLOW	RX FIFO has overflowed. Read out any useful data, then flush the FIFO with <code>SFRX</code>	111	TXFIFO_UNDERFLOW	TX FIFO has underflowed. Acknowledge with <code>SFTX</code>
Value	State	Description																											
000	IDLE	Idle state (Also reported for some transitional states instead of SETTLING or CALIBRATE)																											
001	RX	Receive mode																											
010	TX	Transmit mode																											
011	FSTXON	Frequency synthesizer is on, ready to start transmitting																											
100	CALIBRATE	Frequency synthesizer calibration is running																											
101	SETTLING	PLL is settling																											
110	RXFIFO_OVERFLOW	RX FIFO has overflowed. Read out any useful data, then flush the FIFO with <code>SFRX</code>																											
111	TXFIFO_UNDERFLOW	TX FIFO has underflowed. Acknowledge with <code>SFTX</code>																											
3:0	FIFO_BYTES_AVAILABLE[3:0]	The number of bytes available in the RX FIFO or free bytes in the TX FIFO																											

Tabla 2.4: Resumen del byte de estado

que se debe hacer para habilitar este modo es poner B=1 e indicar la dirección de comienzo en el *byte* de cabecera. Esta forma de intercambio de información tiene la ventaja de que no se precisa enviar un *byte* de cabecera por cada dirección que pretende ser leída o escrita, sino que con uno sólo de esos *byte* podemos escribir o leer todas las posiciones de memoria del integrado CC2500. Debido a la gran ventaja que supone este método en cuanto a ahorro de instrucciones se refiere, este modo de comunicación se ha empleado en el diseño realizado.

Para concluir con esta sección en la tabla 2.5 se muestra el espacio de direcciones para poder acceder a la totalidad de los registros que componen el integrado CC2500. La dirección final a usar se forma añadiendo a la dirección a la izquierda de la tabla, los bits de *burst* y R/W de la parte superior.

### 2.1.8. FIFOs de datos

El módulo de radio contiene dos FIFOs de 64 *bytes* de datos cada una. Tienen la misión de almacenar la información recibida de radiofrecuencia (FIFO de recepción) y la información enviada (FIFO de transmisión). Para acceder a las mismas se hace de igual forma que de si un único registro se tratase, es decir, indicando la dirección de las FIFOs (0x3F) en el *byte* de cabecera.

	Write		Read		
	Single byte	Burst	Single byte	Burst	
	+0x00	+0x40	+0x80	+0xC0	
0x00					IOCFG2
0x01					IOCFG1
0x02					IOCFG0
0x03					FIFOTHR
0x04					SYNC1
0x05					SYNC0
0x06					PKTLEN
0x07					PKTCTRL1
0x08					PKTCTRL0
0x09					ADDR
0x0A					CHANNR
0x0B					FSCTRL1
0x0C					FSCTRL0
0x0D					FREQ2
0x0E					FREQ1
0x0F					FREQ0
0x10					MDMCFG4
0x11					MDMCFG3
0x12					MDMCFG2
0x13					MDMCFG1
0x14					MDMCFG0
0x15					DEVIATN
0x16					MCSM2
0x17					MCSM1
0x18					MCSM0
0x19					FOCCFG
0x1A					BSCFG
0x1B					AGCCTRL2
0x1C					AGCCTRL1
0x1D					AGCCTRL0
0x1E					WOREVT1
0x1F					WOREVT0
0x20					WORCTRL
0x21					FREND1
0x22					FREND0
0x23					FSCAL3
0x24					FSCAL2
0x25					FSCAL1
0x26					FSCAL0
0x27					RCCTRL1
0x28					RCCTRL0
0x29					FTTEST
0x2A					PTEST
0x2B					AGCTEST
0x2C					TEST2
0x2D					TEST1
0x2E					TEST0
0x2F					
0x30	SRES		SRES	PARTNUM	
0x31	SFSTXON		SFSTXON	VERSION	
0x32	SXOFF		SXOFF	FREQUEST	
0x33	SCAL		SCAL	LQI	
0x34	SRX		SRX	RSSI	
0x35	STX		STX	MARCSTATE	
0x36	SIDLE		SIDLE	WORTIME1	
0x37				WORTIME0	
0x38	SWOR		SWOR	PKTSTATUS	
0x39	SPWD		SPWD	VCO_VC_DAC	
0x3A	SFRX		SFRX	TXBYTES	
0x3B	SFTX		SFTX	RXBYTES	
0x3C	SWORRST		SWORRST	RCCTRL1_STATUS	
0x3D	SNOP		SNOP	RCCTRL0_STATUS	
0x3E	PATABLE	PATABLE	PATABLE	PATABLE	
0x3F	TX FIFO	TX FIFO	RX FIFO	RX FIFO	

R/W configuration registers, burst access possible

Command strobes, status registers (read only) and multi byte registers

Tabla 2.5: Espacio de direcciones SPI del integrado CC2500

Un detalle muy a tener en cuenta reside en el hecho de que cuando se escribe a la FIFO de transmisión es responsabilidad del usuario evitar que se produzca su rebasamiento (*overflow*). De igual forma, a la hora de leer datos de la FIFO de recepción es igualmente importante evitar leer una posición vacía de la misma. En ambos casos, si se produjesen los acontecimientos narrados, se produce un error de lectura /escritura que hace inservible los datos.

Para poder llevar la cuenta de cuántos *bytes* libres quedan en la FIFO de transmisión a medida que la vamos llenando y de cuántos *bytes* quedan por leer en la FIFO de recepción a medida que la vamos vaciando, se debe consultar los bits correspondientes en el *byte* de estado que informan de tal hecho. Dicho *byte* se actualiza con cada acceso a la FIFO, de manera que el seguimiento se puede realizar de forma instantánea.

### 2.1.9. Pines de propósito general

El integrado CC2500 cuenta con 2 pines dedicados configurables (GD0 y GD2) y un tercer pin compartido con la función *Serial Output* (GD1) que puede también sacar información útil del estado interno del módulo de radio para control *software*.

En el diseño realizado esta funcionalidad ha resultado muy útil, ya que resulta idónea para la elaboración de interrupciones al microcontrolador. Así, se ha empleado el pin GD2 para provocar una interrupción cuando se reciba un paquete en la FIFO de recepción, y el pin GD0 para leer el valor analógico de temperatura obtenido por el sensor de temperatura que el integrado posee.

### 2.1.10. Formato de paquete

El formato del paquete que se envía y recibe puede ser configurado y consta de los siguientes campos (figura 2.5):

- Preámbulo
- Palabra de sincronización (*Sync Word*)
- Longitud del paquete
- Dirección
- Carga útil
- Código de Redundancia Cíclica (CRC)

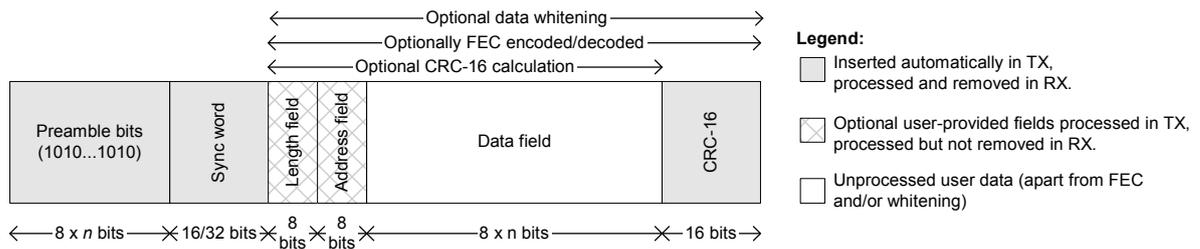


Figura 2.5: Formato de paquete

Comenzando en primer lugar por el preámbulo, éste consiste en un patrón alternativo de unos y ceros que permiten al receptor del paquete sincronizarse con el transmisor y poder así captar la información de forma adecuada. Su longitud en *bytes* es programable y se emite de forma ininterrumpida hasta que se tienen datos disponibles para enviar en la FIFO de transmisión. El valor recomendado es de 4 *bytes*, por lo que ha sido el empleado en el diseño realizado. Una vez procesado en recepción, es eliminado automáticamente.

Respecto a la palabra de sincronización, ésta consta de 2 o 4 *bytes* (configurable) que igualmente se emplea para proveer sincronismo con el paquete entrante. De igual forma, esta palabra de sincronización puede ser empleada como método de filtrado de los paquetes recibidos. Así, en caso de no recibirse un mínimo número de bits que es configurable, el paquete será descartado. En este sentido, para el diseño elaborado se ha optado por emplear un total de 4 *bytes* de palabra de sincronización y un mínimo de 30 bits de los 32 totales para que el paquete sea correctamente procesado. Se ha optado por este alto grado de "seguridad" debido a que el diseño realizado está pensado para llevarse a cabo en entornos industriales y es sabido que dichos entornos son altamente ruidosos desde el punto de vista electromagnético. De esta forma, la posibilidad de errores es relativamente elevada y no nos podemos permitir la posibilidad de trabajar con datos corruptos. Al igual que el preámbulo, tras su procesado es retirada automáticamente.

El *byte* de longitud de paquete ofrece una información diferente según el modo de paquete soportado. Así, existen dos posibles modos:

- Paquetes de longitud fija: Todos los paquetes enviados tienen la misma longitud. La longitud de la carga útil (más adicionalmente el *byte* de dirección, si existe) es lo que debería contener el *byte* de longitud de paquete. No obstante, debido a que su valor es siempre el mismo, no se debe enviar.
- Paquetes de longitud variable: Los paquetes pueden tener diferentes tamaños, y

por tanto, la cantidad de carga útil puede variar. De este modo, el *byte* de longitud deberás contener la longitud de la carga útil más grande que se pretende enviar (más adicionalmente el *byte* de dirección, si existe).

En el diseño realizado se ha optado por la primera opción. Esto es así, porque nuestra intención es la de intercambiar información del estado de los pines de diferentes módulos, lo que supone una carga útil más bien reducida. En este sentido, se ha preferido optar por una carga útil de tamaño fijo, igual a la mayor cantidad de información útil transmisible. Así, en caso de tener menos información se rellenan los *bytes* libres con ceros. Con esto conseguimos eliminar la problemática que subyace al emplear paquetes de longitud variable, sobre todo si éstos superan los 64 *bytes* y no caben enteros en las FIFOs. Con todo esto, al final hemos optado por fijar un tamaño de paquete de 16 *bytes*.

En lo que respecta al *byte* de dirección, éste se emplea para poder comunicarnos con un dispositivo en concreto siempre y cuando habilitemos la opción correspondiente en uno de los registros. De este modo, cuando un dispositivo recibe un paquete, si la dirección de dicho paquete coincide con la que el dispositivo tiene almacenada en un registro interno, el paquete se copia en la FIFO de recepción. En caso contrario, el paquete se descarta. En caso de emplear paquetes de tamaño fijo, como es nuestro caso, este *byte* es el primero que debe colocarse en la FIFO de transmisión.

En nuestro diseño, hemos optado por incorporar esta funcionalidad, ya que nos permite la posibilidad de reducir el procesamiento innecesario de paquetes que no tienen como destino una determinada dirección. No obstante, tenemos el problema de que solamente permite filtrar entre 256 (1 *byte*) posibles direcciones, y eso en el diseño realizado se puede quedar un poco corto. De este modo, se ha decidido emplear este método con el *byte* menos significativo de las verdaderas direcciones que nosotros empleamos y que son de 4 *bytes*. Aunque no nos evita el hecho de tener que comprobar *vía software* la dirección completa, lo cierto es que filtra una buena parte del tráfico total que se puede dar, con el consiguiente ahorro de consumo.

La carga útil, tal y como indica su nombre se refiere a la información concerniente exclusivamente a la aplicación y no a temas de procesado de paquetes. En nuestro caso, dicha parte consta de 6 *bytes*.

Finalmente, en lo que respecta al CRC, éste es de 2 *bytes* y es calculado automáticamente por el transmisor y retirado también de forma automática en recepción. No

obstante, existe la posibilidad de añadir en recepción dos *bytes* adicionales a los ya recibidos, donde entre otras cosas existe un bit que indica si el CRC recibido fue o no fue correcto. Dicho bit puede ser analizado para descartar un paquete en caso de que su valor sea cero.

En el caso del diseño realizado, estos dos *bytes* adicionales son añadidos, pero para determinar si un paquete fue adecuadamente recibido o no hemos empleado un método alternativo [2]. Dicho método, consiste en activar la opción de autolimpieza (*autoflush*) de la FIFO de recepción por la cual la FIFO se vacía en caso de que el CRC sea incorrecto. De esta forma, cuando se recibe un paquete, basta con analizar el valor del número de *bytes* que existen en la FIFO de recepción y que se encuentra en un registro. Si el valor leído es cero, eso indica que la FIFO se ha vaciado y por tanto el CRC fue incorrecto. En caso de que su contenido sea mayor que cero, eso indicará que el CRC fue correcto.

### 2.1.11. Whitening

En las siguientes dos subsecciones se detallan otros métodos alternativos a los ya expuestos en la sección anterior para poder evitar que los datos lleguen corruptos a su destino. En ese sentido, vamos a tratar en primer lugar el método conocido como *Whitening*.

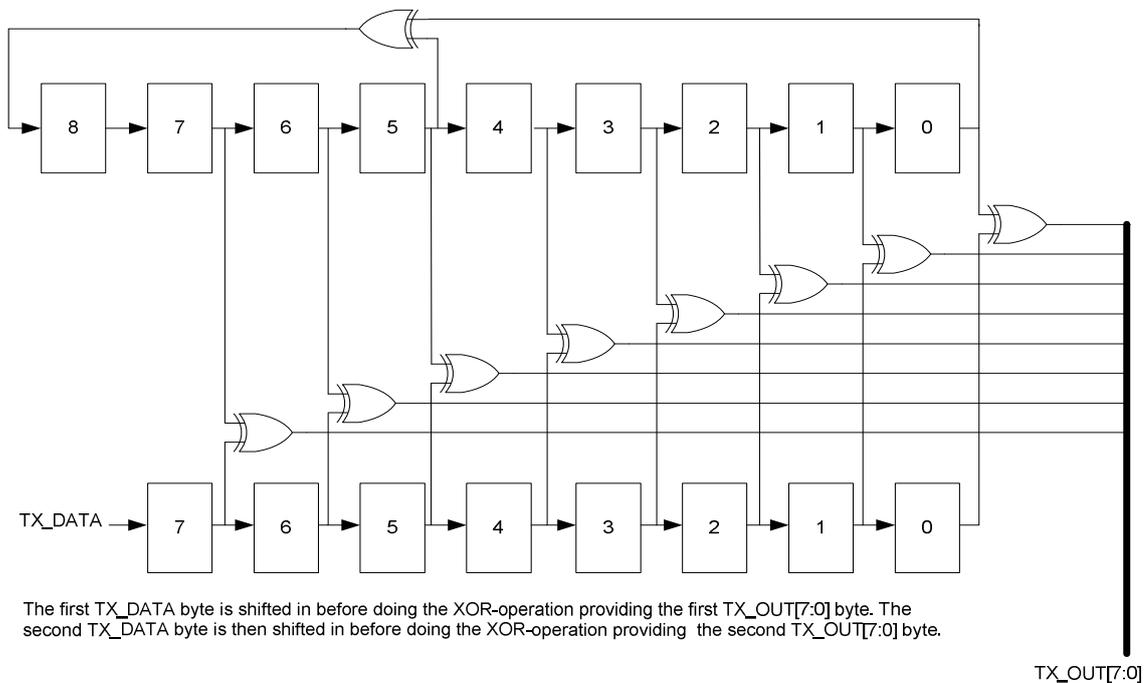


Figura 2.6: Whitening de los datos en modo transmisión

En el mundo real, muchos datos contiene a menudo largas secuencias de unos y ceros. Para evitar que se pierda el sincronismo en estos casos y los datos sean incorrectamente procesados, se puede optar por hacer *whitening* sobre los datos antes de transmitir, y *de-whitening* los datos en recepción.

Este método consiste en hacer, excepto en el preámbulo y la palabra de sincronización, un XOR binario de los datos con una secuencia pseudoaleatoria de 9 bit (PN9) como muestra la figura 2.6.

En recepción, a los datos se les aplica el mismo XOR binario con la misma secuencia pseudoaleatoria. De este modo, el *whitening* es revertido, y los datos originales aparecen en el receptor.

En el diseño realizado se ha optado por incluir este método para mejorar aún más la posibilidad de recibir los paquetes de manera correcta.

### 2.1.12. Corrección de errores

El integrado CC2500 soporta el denominado FEC (*Forward Error Correction*). Este mecanismo es empleado junto con el CRC para reducir la tasa de error de bit (BER, *Bit Error Rate*) cuando el integrado opera cerca del límite de sensibilidad. La redundancia es añadida de tal modo que el receptor pueda recuperar los datos originales en presencia de algunos errores de bits.

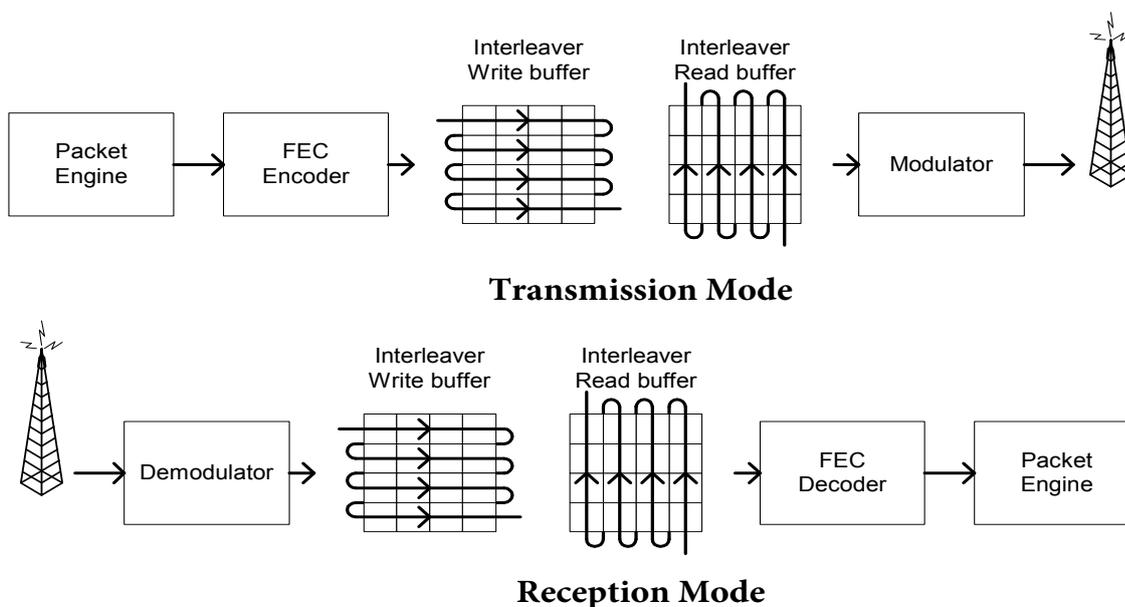


Figura 2.7: Principio de funcionamiento de Interleaving

El esquema adoptado por el módulo de radio respecto a este método consiste en un código convolucional por el cual  $n$  bits son generados basados en  $k$  bits de entrada y los  $m$  más reciente bits.

Por otra parte, junto con el FEC se suele emplear el método denominado *interleaving* como técnica para proteger la información frente a los errores de ráfaga (*burst errors*). Este tipo de errores ocasionales afectan a varios bits seguidos, e invalidan las propiedades correctoras de los CRC. Así, al emplear técnicas de entrelazado los errores de ráfaga se ven distribuidos entre varias palabras, facilitando la labor correctora. El método empleado para realizar este proceso se puede observar en la figura 2.7.

En lo que respecta al proyecto realizado, se ha preferido no incluir estos métodos en el diseño final. Esto es así, porque si bien constituyen un buen método de corrección de errores, también es cierto que incrementan de forma considerable el ancho de banda necesario, con la correspondiente reducción de sensibilidad. De este modo, hemos considerado que el empleo combinado de la técnica del *whitening* junto con el CRC supone ya protección suficiente.

### 2.1.13. Formatos de modulación

El integrado CC2500 soporta formatos de modulación de amplitud, y de desplazamiento de frecuencia y fase. Además, ofrece la posibilidad de que los datos puedan ser codificados con un código de tipo *Manchester*. A continuación se detallan mejor los diferentes tipos de modulación empleados:

- 2-FSK (*Frequency Shift Keying*): Consiste en modular la señal digital generada empleando dos tonos a frecuencias diferentes, uno para cada posible nivel lógico. Su espectro consta de dos sincs centradas en las frecuencias de los tonos. Debido a que la transición entre una frecuencia y otra se realiza de forma brusca (puede verse como si su envolvente fuese un pulso rectangular) los lóbulos secundarios de las sincs se extienden en un rango más o menos amplio de frecuencias.
- MSK (*Minimum Shift Keying*): Este tipo de modulación sigue el mismo principio que la 2-FSK, salvo que la envolvente de los tonos, en vez de ser un pulso rectangular, es un medio ciclo de senoide. Debido a esto, se tiene que las transiciones entre los diferentes pulsos son más suaves, lo que se traslada al dominio de la frecuencia en un ahorro significativo de ancho de banda con respecto a la modulación 2-FSK.
- GFSK (*Gaussian Frequency Shift Keying*): Modulación similar a la MSK pero empleando como forma de pulso un pulso gaussiano. Constituye junto con la MSK, las

modulaciones que poseen una mayor eficiencia espectral.

- OOK (On-Off Keying): Modulación consistente en transmitir portadora a una frecuencia cuando se transmite un "1" lógico, y no transmitir nada cuando se transmite un "0" lógico.

En el diseño realizado, la modulación de amplitud OOK fue descartada de inicio, puesto que como es sabido, el ruido se suma especialmente en amplitud, y debido a que el sistema va a estar inmerso en un ambiente altamente ruidoso, no constituye la mejor opción. Por tanto la decisión se encontraba entre las modulaciones de frecuencia. Abandonando la posibilidad de emplear la modulación 2-FSK, debido a su gran ancho de banda necesario, la decisión se centró entre MSK y GFSK. Finalmente, se optó por la primera por una simple elección de diseño. De igual forma se podía haber escogido la modulación GFSK.

#### **2.1.14. Tasa de transmisión/recepción**

Tan importante como el formato de modulación escogido, lo es también la velocidad de transmisión seleccionada. Así, la tasa de datos empleada en transmisión, o la tasa de datos esperada en recepción (ha de ser la misma), puede ser fácilmente configurada en uno de los registros del módulo de radio.

Las posibles tasas, expresadas en KBaudios (Kbit/s) se pueden observar en la figura 2.6. En el caso del diseño realizado, se ha optado por seleccionar una tasa de 250 KBaudios, buscando un compromiso entre ancho de banda ocupado y velocidad de transmisión. De este modo, una velocidad de transmisión suficientemente alta nos permite reducir el tiempo de emisión (y por tanto el consumo) así como las colisiones con otros sistemas que empleen la misma banda de frecuencias, ya que el tiempo que se tiene ocupado el canal es menor.

#### **2.1.15. Frecuencia de transmisión**

La frecuencia base sobre la que se van a transmitir los diferentes datos es también un aspecto configurable en el integrado CC2500. Dicha frecuencia coincide con la frecuencia central del canal que se emplea para establecer la comunicación. En un principio cabría pensar que con emplear uno de los múltiples canales posibles en la banda de frecuencias de 2.4-2.4835 Ghz es suficiente. Sin embargo, la realidad nos dice que hacer esto puede no resultar una muy buena idea. La razón de este pensamiento se encuentra en que dicha banda de frecuencias es una de las más saturadas actualmente. Esto es así porque se trata de una banda de frecuencias para la que no se requiere licencia alguna y por tanto

Min Data Rate [kBaud]	Typical Data Rate [kBaud]	Max Data Rate [kBaud]	Data Rate Step Size [kBaud]
0.8	1.2/2.4	3.17	0.0062
3.17	4.8	6.35	0.0124
6.35	9.6	12.7	0.0248
12.7	19.6	25.4	0.0496
25.4	38.4	50.8	0.0992
50.8	76.8	101.6	0.1984
101.6	153.6	203.1	0.3967
203.1	250	406.3	0.7935
406.3	500	500	1.5869

Tabla 2.6: Tasas de transmisión posibles y tamaño del paso

es gratuita. Así, son muchos los sistemas de comunicaciones y protocolos que se emiten empleando esta franja del espectro radioeléctrico. Por tanto, pensar en la posibilidad de emplear diversos canales para la comunicación puede hacer que nuestro sistema sea más robusto. De esta forma, tanto la frecuencia base como los canales de transmisión se pueden configurar a través de una serie de registros del módulo de radio.

En el caso del diseño realizado, este hecho no se nos ha pasado inadvertido, más aún teniendo en cuenta que nuestro sistema es de baja potencia de emisión y cualquier señal, por poca potencia que tenga, puede llegar a interferir con nuestro diseño.

En este sentido, para poder buscar los mejores canales posibles para nuestra emisión ha sido necesario determinar qué sistemas son los potencialmente interferentes con el nuestro. De este modo, llegamos a la conclusión de que tanto *Wi-Fi* como *Bluetooth* eran los principales protocolos que operaban en esta banda de frecuencias y que lo hacían con las mayores potencias de emisión. Así pues, nuestro objetivo consistió en averiguar en qué canales concretos dentro de la banda de frecuencia de los 2.4 GHz emitían ambos protocolos para tratar de transmitir nosotros en los huecos disponibles.

Comenzando por *Wi-Fi* (802.11), hay que decir que este protocolo cuenta con diferentes versiones del mismo, cada una de las cuales emiten en diferentes canales y con diferentes anchos de banda que las otras. Por tanto, para poder trabajar hemos escogido los estándares 802.11b y 802.11g que son dos de los más empleados.

Dichos estándares emplean 13 canales posibles en España dentro de la banda de los 2.4 GHz y emiten en cada uno de ellos con 22 MHz de ancho y 5 MHz de separación entre canales [3]. Como se puede observar en la figura 2.7 todos los canales se solapan con sus

Frecuencia (MHz)	Canal
2412	1
2417	2
2422	3
2427	4
2432	5
2437	6
2442	7
2447	8
2452	9
2457	10
2462	11
2467	12
2472	13

Tabla 2.7: Canales Wi-Fi en España para los estándares 802.11b y 802.11g

adyacentes, de modo que para poder comunicarse lo que se tiene que hacer es seleccionar algunos de los no solapados. Así, si se quisiese aprovechar el mayor número de canales posibles, de acuerdo a la tabla 2.7 queda claro que los canales 1, 7 y 13 son los únicos que no se solapan entre sí en una comunicación simultánea. Por tanto, las bandas comprendidas entre 2.423-2.431 GHz y entre 2.453-2.461 GHz podrían ser empleadas en nuestro diseño, siempre y cuando la comunicación vía *Wi-Fi* tenga lugar a través de los canales indicados.

Por otro lado, el otro estándar que nos compete es *Bluetooth*(802.15). En este caso, el ancho de banda y los canales ocupados son idénticos en las diferentes versiones existentes. Así, en el estándar [4] se recoge que existen 79 posibles canales de 1 MHz de ancho de banda y separados una distancia mínima de 1.5 MHz y máxima de 3 MHz (tabla 2.8).

En vista de estos resultados, para nuestro diseño ha sido escogida una frecuencia de 2.430GHz que se encuentra entre los canales 1 y 7 de *Wi-Fi* y en una zona donde si se empleasen de manera consecutiva canales de *Bluetooth* existiría la zona de guarda.

### 2.1.16. Potencia de emisión

El nivel de potencia de emisión de la señal de radiofrecuencia es otro aspecto que el integrado CC2500 nos permite programar. Así, el módulo de radio incorpora internamente una tabla con los posibles valores de potencia que pueden ser empleados (tabla 2.9).

Regulatory range	RF channels
2.400-2.4835 GHz	$f = 2402 + k \text{ MHz}, k = 0, \dots, 78$

Lower guard band	Upper guard band
1.5 MHz	3 MHz

Tabla 2.8: Canales Bluetooth disponibles en el estándar 802.15

Estos valores pueden ir desde -55 dBm o menos, hasta +1dBm de potencia. Además, como cabría esperar, a medida que dicho nivel de potencia va en aumento, también lo hace el consumo de corriente.

En el caso del diseño llevado a cabo, se ha escogido precisamente este valor de +1dBm para poder alcanzar las mayores distancias posibles. En nuestro caso, aunque el consumo de corriente importa, el sistema va a ser alimentado mayormente a través de un puerto USB y por tanto no es tan crítico como si la dependencia fuese de baterías.

Por otra parte, puede parecer, y lo es, que una potencia máxima de +1dBm es sumamente pequeña para la aplicación que se desea implementar. No obstante, debemos recordar que el integrado CC2500 constituye un módulo de bajo nivel de potencia, por lo que valores de este orden de magnitud cabrían ser esperados. Sin embargo, este hecho no constituye ningún problema en caso de que nuestra aplicación requiera extender el rango de alcance. Así, siempre puede ser posible añadir un amplificador de potencia externo insertado entre la antena y el integrado.

### 2.1.17. SmartRF Studio 7

Una vez comprendido el funcionamiento del integrado CC2500, conviene ahora tratar de averiguar como podemos llevar a cabo la programación del mismo, es decir, de sus registros de configuración. Así, como se pudo ver en una subsección anterior, buena parte de dichos registros comprenden aspectos relacionados con el modo en que se lleva a cabo la comunicación vía radio (modulaciones, sintetizadores de frecuencia, control automático de ganancia, frecuencia base, espaciado entre canales, etc.). Debido a esto, para poder programar adecuadamente dichos registros se requiere de un elevado grado de conocimiento de comunicaciones inalámbricas, así como del tiempo suficiente para poder resolver las diferentes ecuaciones que contienen los parámetros que deben ser obtenidos. Y en un gran número de ocasiones, no se dispone de ninguno de los dos. Por tanto, para facilitar la programación de dichos módulos de radio, *Texas Instruments* ha desarrollado

Output Power, Typical, +25°C, 3.0 V [dBm]	PATABLE Value	Current Consumption, Typical [mA]
(-55 or less)	0x00	8.4
-30	0x50	9.9
-28	0x44	9.7
-26	0xC0	10.2
-24	0x84	10.1
-22	0x81	10.0
-20	0x46	10.1
-18	0x93	11.7
-16	0x55	10.8
-14	0x8D	12.2
-12	0xC6	11.1
-10	0x97	12.2
-8	0x6E	14.1
-6	0x7F	15.0
-4	0xA9	16.2
-2	0xBB	17.7
0	0xFE	21.2
+1	0xFF	21.5

Tabla 2.9: Niveles de potencia para varios valores de potencia de emisión

una *suite* denominada *SmartRF Studio*. Actualmente, se llegan por la versión número 7, y puede ser obtenida de manera gratuita desde la página web oficial de *Texas Instruments* en [5].

Las principales características de dicho programa son [6]:

- Soporte para las familias CC11xx, CC12xx, CC24xx y CC25xx.
- Envío y recepción de paquetes entre diferentes nodos.
- Tests para determinar el patrón de radiación de las antenas de los integrados.
- Fácil modo para testar paquetes recibidos.
- Conjunto de recomendados valores típicos para todos los dispositivos soportados.
- Lectura y escritura de manera individual de registros de RF.
- Detallada información sobre los campos de bits de cada registro.

- Guardar/Cargar configuración de los dispositivos desde un archivo.
- Exportar las características de los registros a un formato concreto.
- Comunicación con placas de evaluación a través del puerto USB o el puerto paralelo.
- Soporte de hasta 32 placas de evaluación en un mismo ordenador.

Debido a todas estas características, queda claro por qué este *software* está altamente recomendado para obtener la óptima configuración de los diferentes registros, sobre todo teniendo en cuenta que éstos pierden su contenido después de producirse un reset.

Explicar el funcionamiento del mismo, por otra parte, es una tarea muy sencilla, pues la interfaz gráfica que presenta es muy intuitiva y fácilmente manejable. No obstante, existe un tutorial en línea [7] donde se detalla el funcionamiento de cada una de las partes del programa, así como una serie de ejemplos de aplicación.

El programa consta básicamente de dos partes bien diferenciadas. En una primera, nada más abrir el programa, se muestran las imágenes en miniatura de cada uno de los transceptores de RF que son soportados (figura 2.8). Haciendo doble *click* en el deseado se abre el correspondiente panel de control.



Figura 2.8: Elección de dispositivo en SmartRF Studio 7

En dicho panel de control, tal y como se muestra en la figura 2.9 , se pueden observar varias partes bien diferenciadas:

- Configuraciones típicas (*Typical settings*): En esta parte se recogen algunas de las configuraciones más habituales, proporcionando los valores de todos y cada uno de los registros de configuración.
- Parámetros de RF (*RF Parameters*): En esta parte se pueden configurar de manera manual los aspectos más relevantes relacionados con la comunicación inalámbrica, i.e., la frecuencia base, el número de canales, el espaciado entre canales, el ancho de banda del filtro de recepción, la tasa de transmisión, la frecuencia base del sintetizador de frecuencia, el formato de modulación, la potencia de salida y el *whitening*.
- Testeo de placas de evaluación (*Evaluation Board Test*): En esta parte (parte inferior de la figura 2.9) se pueden configurar diferentes aspectos de testeo y funcionalidad de las placas de evaluación de *Texas Instruments*.

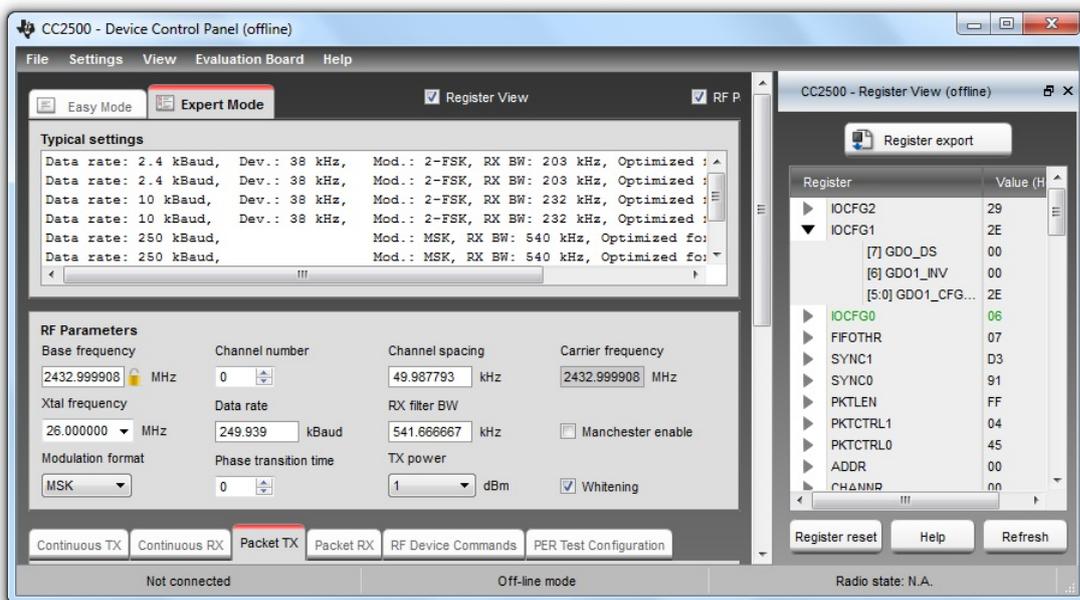


Figura 2.9: Panel de configuración en SmartRF Studio 7

- Vista de registros (*Register View*): En esta parte, se muestran los valores que tienen los registros actualmente, y se actualizan automáticamente cada vez que se realiza un cambio en alguno de ellos. Estos registros son accesibles además, a través de los campos de bits que los conforman.

- Exportación de los registros (*Register export*): Al hacer *click* sobre esta opción se abre una nueva ventana donde se nos permite elegir el formato más adecuado de salida que queremos para los registros (figura 2.10).

En el diseño realizado, se ha empleado este *software* para la correcta programación de los registros de configuración. Así, hemos optado por partir de una de las configuraciones típicas, en concreto la de 250 KBaudios con modulación MSK y optimizada para sensibilidad. Después introdujimos los aspectos que hemos ido considerando a lo largo de las anteriores subsecciones, y finalmente exportamos el conjunto de los registros en formato vector de C. Esto lo hicimos así, para poder escribir el conjunto de ellos de una sola vez empleando el modo *burst*.

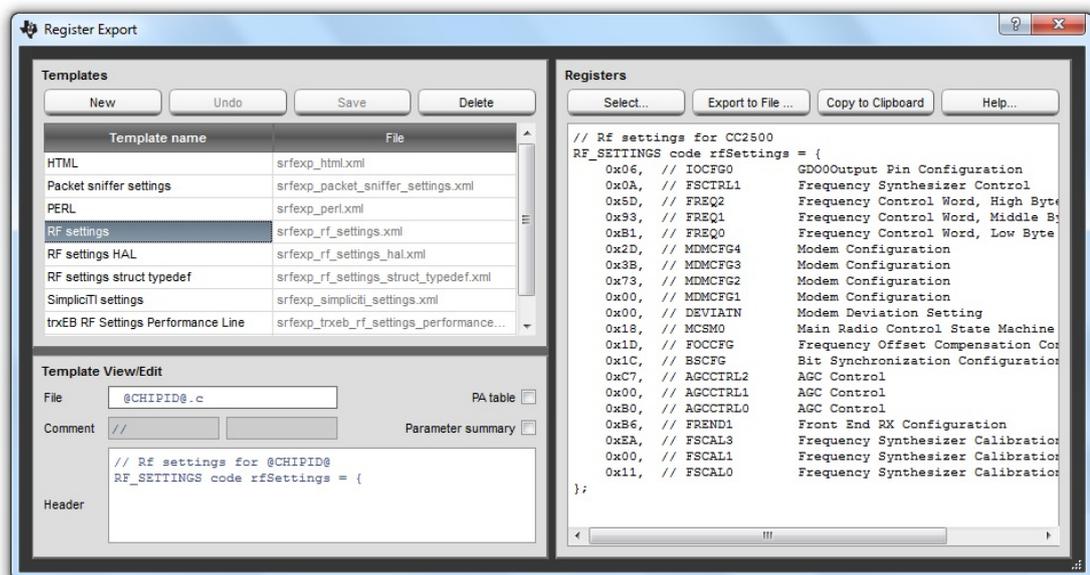


Figura 2.10: Elección de formato de salida en SmartRF Studio 7

## 2.2. LPC11E13

### 2.2.1. Descripción del producto

Los microcontroladores de la familia LPC11Exx (*NXP Semiconductors*) tienen como núcleo central un microprocesador ARM Cortex-M0 y constituyen una familia de microcontroladores de 32 bit de bajo coste, diseñados especialmente para aplicaciones de 8/16 bits y con consumos de potencia muy reducidos. Además, su simple conjunto de instrucciones y direccionamientos de memoria logran reducir el tamaño del código comparado con las existentes arquitecturas de 8/16 bits.

La familia LPC11Exx puede operar a frecuencias de hasta 50 MHz, e incluye hasta 32 kB de memoria *flash*, hasta 8 kB de memoria SRAM de datos, una interfaz I2C *Fast-mode Plus*, una RS-485/EIA-485 USART con soporte de modo síncrono e interfaz *Smart Card*, dos SSP, 4 contadores/temporizadores, un ADC de 10 bits y hasta 54 pines I/O de propósito general.

En las siguientes subsecciones se expondrán con más detalle los periféricos y conjunto de registros que han sido necesarios para el diseño realizado.

### 2.2.2. ARM Cortex-M0

Antes de comenzar a detallar las características específicas con que cuenta el microcontrolador LPC11E13, vamos antes a exponer unas breves nociones acerca del microprocesador ARM Cortex-M0, el núcleo del microcontrolador. Más información acerca del mismo puede encontrarse en [8].

El procesador Cortex-M0 constituye un procesador de 32 bits altamente optimizado, tanto en área como en potencia, que consta de una *pipeline* de 3 etapas y de arquitectura *von Neumann*. Trata de manera excepcional la eficiencia de energía gracias a un pequeño pero poderoso conjunto de instrucciones y a un procesamiento *hardware* incorporado que incluye un multiplicador de un sólo ciclo de reloj.

Por otro lado, el Cortex-M0 implementa la arquitectura denominada ARMv6-M, que está basada en el conjunto de instrucciones *Thumb* de 16 bit pero que incluye también la tecnología *Thumb-2*. Esto otorga al procesador una alta capacidad de densidad de código.

Continuando con la descripción del Cortex-M0, éste cuenta con varios periféricos que pueden observarse en la figura 2.11 y que son:

- Un Controlador de Interrupciones Vectorizadas y Anidables (NVIC, *Nested Vectored Interrupt Controller*) que se encarga de gestionar las interrupciones y excepciones en el sistema y proporcionar una baja latencia de las mismas. Se analizará con más detalle en la subsección 2.2.6.
- Un bloque de control del sistema (SCB, *System Control Block*) que suministra información acerca del sistema, incluyendo datos de configuración, control y notificaciones del sistema de excepciones.
- Una interfaz de depuración que implementa una completa solución *hardware* de depuración, con numerosas opciones de puntos de ruptura (*breakpoint*). Esta interfaz

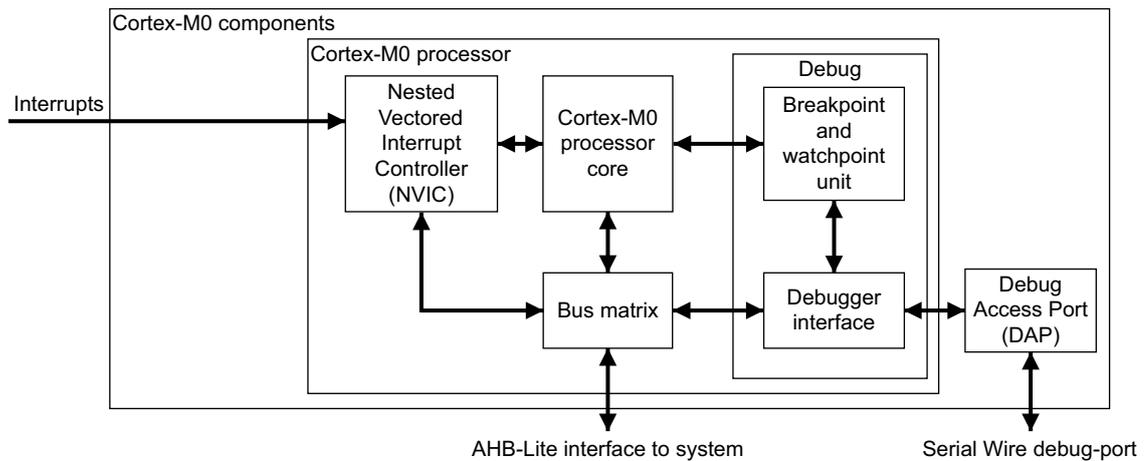


Figura 2.11: Periféricos que componen el microprocesador Cortex-M0

proporciona una amplia visibilidad del procesador, memoria y periféricos a través de un puerto de depuración serie de dos pines (SWD, *Serial Wire Debug*).

- Una interfaz de comunicación con el sistema que proporciona alta velocidad y baja latencia en los accesos a la memoria.
- Un temporizador del sistema (SysTick, *System Timer*) de 24 bits. Esta destinado sobre todo a su empleo junto con sistemas operativos en tiempo real (RTOS, *Real Time Operative System*).

Finalmente, para concluir con esta descripción del modo de funcionamiento de los microprocesadores Cortex-M0 se expone a continuación el conjunto de registros con que cuentan, así como una breve descripción de los mismos.

Estos registros, que se pueden observar en la figura 2.12, son:

- R0-R12: Estos 13 registros de 32 bits son de propósito general para operaciones con datos.
- SP (*Stack Pointer*): Este registro (R13) constituye el puntero de pila, esto es, apunta a la dirección donde se encuentra el tope de la pila (TOS, *Top Of Stack*) en cada momento. La pila que implementa el microprocesador es de tipo *Full Descending*, i.e. cuando un nuevo dato es empujado a la pila, se decrementa el contenido de SP y luego se escribe el dato en la nueva dirección apuntada por SP. Los microprocesadores Cortex-M0 implementan dos pilas, pila principal y pila de procesos, cada una de las cuales tiene copias independientes del puntero de pila, denominados puntero

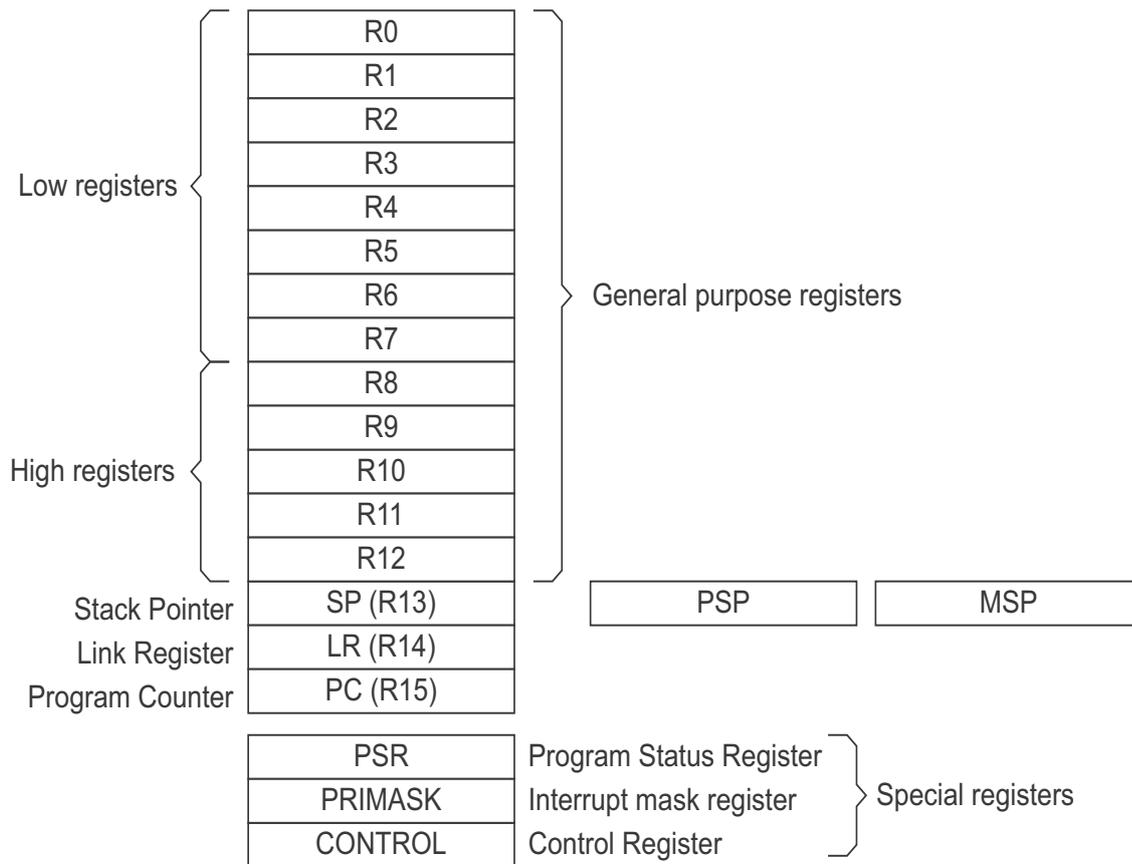


Figura 2.12: Conjunto de registros del microprocesador Cortex-M0

de pila principal (MSP, *Main Stack Pointer*) y puntero de pila de procesos (PSP, *Process Stack Pointer*) respectivamente.

- LR (*Link Register*): Este registro (R14) almacena la información de retorno de las subrutinas, llamadas a función y excepciones.
- PC (*Program Counter*): Este registro (R15) contiene la dirección de la memoria de programa actual en cada momento. Tras un *reset*, el procesador carga el PC con el valor del vector de *reset*, que se encuentra en la dirección 0x00000004.
- PSR (*Program Status Register*): Este registro, combina a su vez a otros 3 registros que son mutuamente excluyentes y que son:
  - APSR (*Application Program Status Register*): Este registro contiene el estado de los flags Negativo, Cero, Acarreo y Desbordamiento de la ejecución de las previas instrucciones.

- IPSR (*Interrupt Program Status Register*): Este registro contiene el número de excepción de la actual Rutina de Servicios de Interrupción (ISR, *Interrupt Service Routine*).
- EPSR (*Execution Program Status Register*): Este registro contiene el estado del bit *Thumb*.
- PRIMASK (*Priority Mask Register*): Este registro permite la habilitación de todas las excepciones con niveles configurables de prioridad.
- CONTROL: Este registro controla qué pila se está usando cuando el procesador está ejecutando una aplicación *software* (Modo *Thread*). Así, permite seleccionar entre los punteros de pila MSP o PSP.

Finalmente, y para concluir con esta subsección, en el apéndice A se pueden encontrar el conjunto de instrucciones que este microprocesador emplea.

### 2.2.3. Características principales

Las principales características del microcontrolador escogido, el LPC11E13FBD48/301, se citan a continuación [9]:

- Sistema
  - Procesador Cortex-M0 corriendo a una frecuencia de hasta 50 MHz.
  - Consta de un Controlador de Interrupciones Vectorizadas y Anidables (NVIC).
  - Interrupciones no enmascarables (NMI, *Non-Maskable Interrupt*) seleccionables desde varias fuentes de entrada.
  - Disponibilidad de un *Tick Timer*.
- Memoria
  - 24 kB de memoria *flash* de programa.
  - 16 kB de memoria ROM para el *bootloader*.
  - 8 kB de memoria SRAM de datos.
  - 2 kB de memoria EEPROM de datos.
  - *Bootloader* via *software* que cuenta con las opciones *In-System Programming* (ISP) e *In-Application Programming* (IAP).

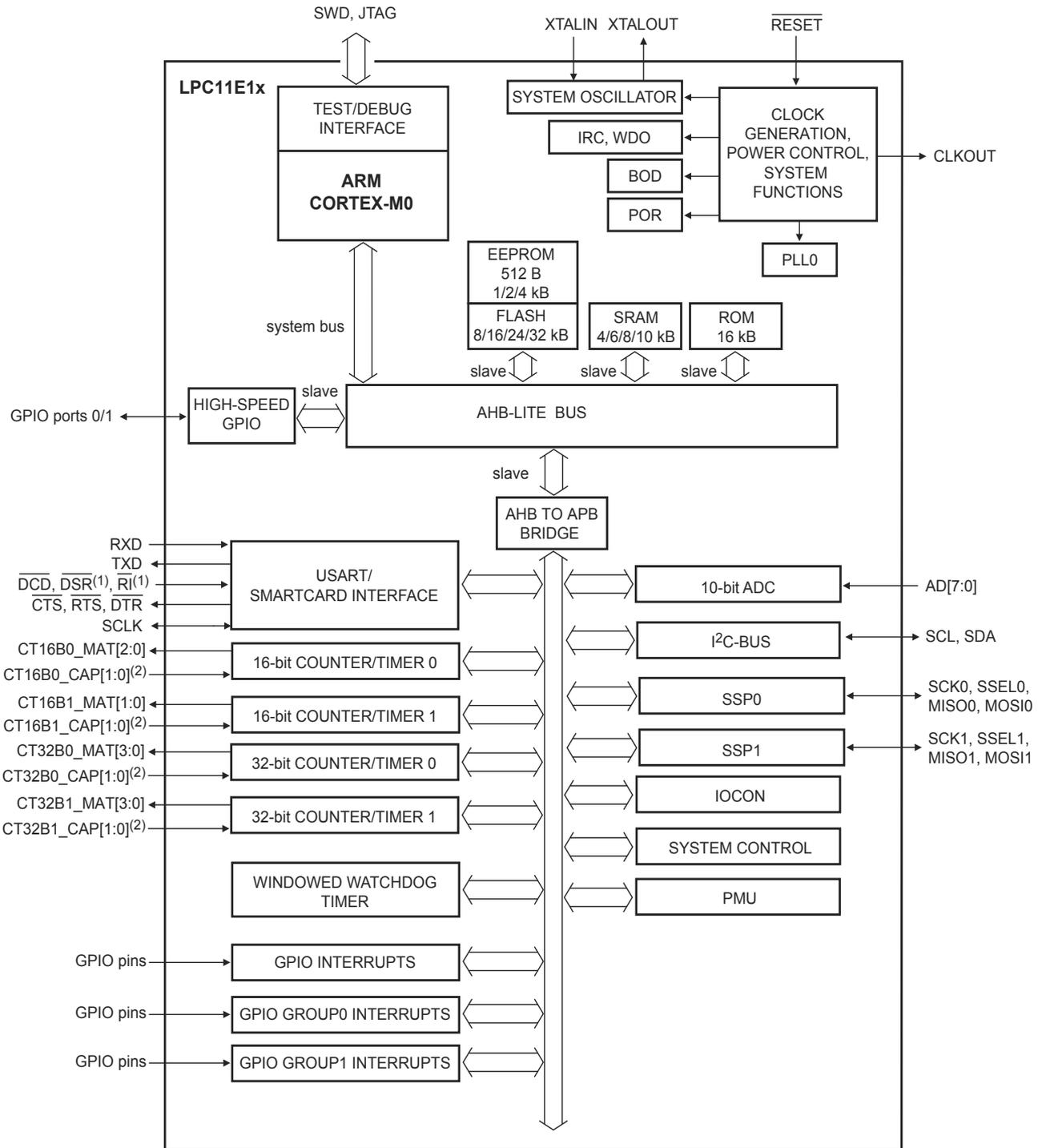
- Periféricos digitales
  - 40 pines de Entradas/Salidas de propósito general (GPIO, *General Purpose Input/Output*) con resistencias configurables de *pull-up* y *pull-down*, modo de repetición y modo de salida en drenador abierto.
  - Hasta 8 pines pueden ser seleccionados como fuente de interrupción por nivel o por flanco.
  - 2 grupos de pines pueden habilitar una interrupción basada en un patrón programable de estados en dichos pines.
  - 4 temporizadores/contadores (2 de 16 bits y 2 de 32 bits) con 4 entradas de captura y 4 registros de comparación.
  - Temporizador de Perro Guardián (WDT, *WatchDog Timer*) programable y con oscilador RC (*Resistor-Capacitor*) interno dedicado.
- Periféricos analógicos
  - Conversor analógico digital de 10 bits con entradas multiplexadas entre 8 pines.
- Interfaces serie
  - USART (*Universal Synchronous Asynchronous Receiver Transmitter*) con generador fraccionario de tasa de baudios, FIFO internas de 16 bytes, una interfaz de control de modem, soporte para modo RS-485/9-bit y modo síncrono, y soporte para interfaz asíncrono de *Smart Card* (ISO 7816-3).
  - 2 módulos SSP (*Synchronous Serial Port*) con capacidades multi-protocolo.
  - Interfaz I2C (*Inter-Integrated Circuit*) con soporte pleno de las características del bus I2C y con *Fast Mode Plus* con una tasa de datos de hasta 1 Mbit/s y múltiples reconocimientos de dirección.
- Generación del reloj
  - Oscilador de cristal externo soportado con frecuencias entre 1 y 25 MHz.
  - Oscilador RC interno de 12 MHz.
  - Oscilador de baja frecuencia para el Perro Guardián.
  - PLL (*Phase Locked Loop*) integrado que permite alcanzar la máxima frecuencia soportada por la CPU (50 MHz) empleando como fuente de reloj el interno o el externo (si incluido).

- Posibilidad de extraer el reloj a través de un pin del microcontrolador.
- Control de consumo
  - 4 modos de bajo consumo disponibles:
    - *Sleep*
    - *Deep-sleep*
    - *Power-down*
    - *Deep-Power-down*
  - Unidad de control de potencia (PMU, *Power Management Unit*) integrada para minimizar el consumo de potencia durante los modos *Sleep*, *Deep-sleep*, *Power-down* y *Deep-Power-down*.
  - El microprocesador se despierta de un modo *Deep-sleep* y *Power-down* via un *Reset*, pines GPIO seleccionados para tal fin o una interrupción del perro guardián.
  - El microprocesador se despierta del modo *Deep-Power-down* usando un pin especial.
  - Disponible funcionalidad *Power On Reset* (POR).
  - Detección de caída de tensión (*Brownout detect*) con 4 separados umbrales para generar interrupciones o un *Reset* forzado.
- Única alimentación a 3.3 V (1.8V a 3.6V).
- Rango de temperatura de -40° C to +85 °C.
- Único encapsulado disponible LQFP48.

Un diagrama de bloques del microcontrolador LPC11E13FBD48/301 y todos sus periféricos se puede observar en la figura 2.13.

#### 2.2.4. Mapa de memoria

La familia de microcontroladores LPC11Exx incorpora distintas regiones de memoria mostradas en la figura 2.14. En dicha figura se puede observar la totalidad del espacio de direcciones desde el punto de vista del programa de usuario que sigue al *Reset*.



(1) Not available on HVQFN33 packages.

(2) CT16B0/1\_CAP1, CT32B1\_CAP1 available on the LQFP64 package only. CT32B0\_CAP1 available on the LQFP64 and LQFP48 packages only.

Figura 2.13: Diagrama de bloques del microcontrolador LPC11E13FBD48/301

Se pueden distinguir dos zonas:

- El área de periféricos AHB (*Advanced High-performance Bus*) que tiene un tamaño de 2 MB y está dividido para permitir hasta 128 periféricos.
- El área de periféricos APB (*Advanced Peripheral Bus*) que tiene un tamaño de 512 kB y está dividido para permitir hasta 32 periféricos.

Todo periférico, cualquiera que sea su tipo, ocupa 16 kB de espacio de memoria. Esto permite simplificar la decodificación de direcciones para cada periférico.

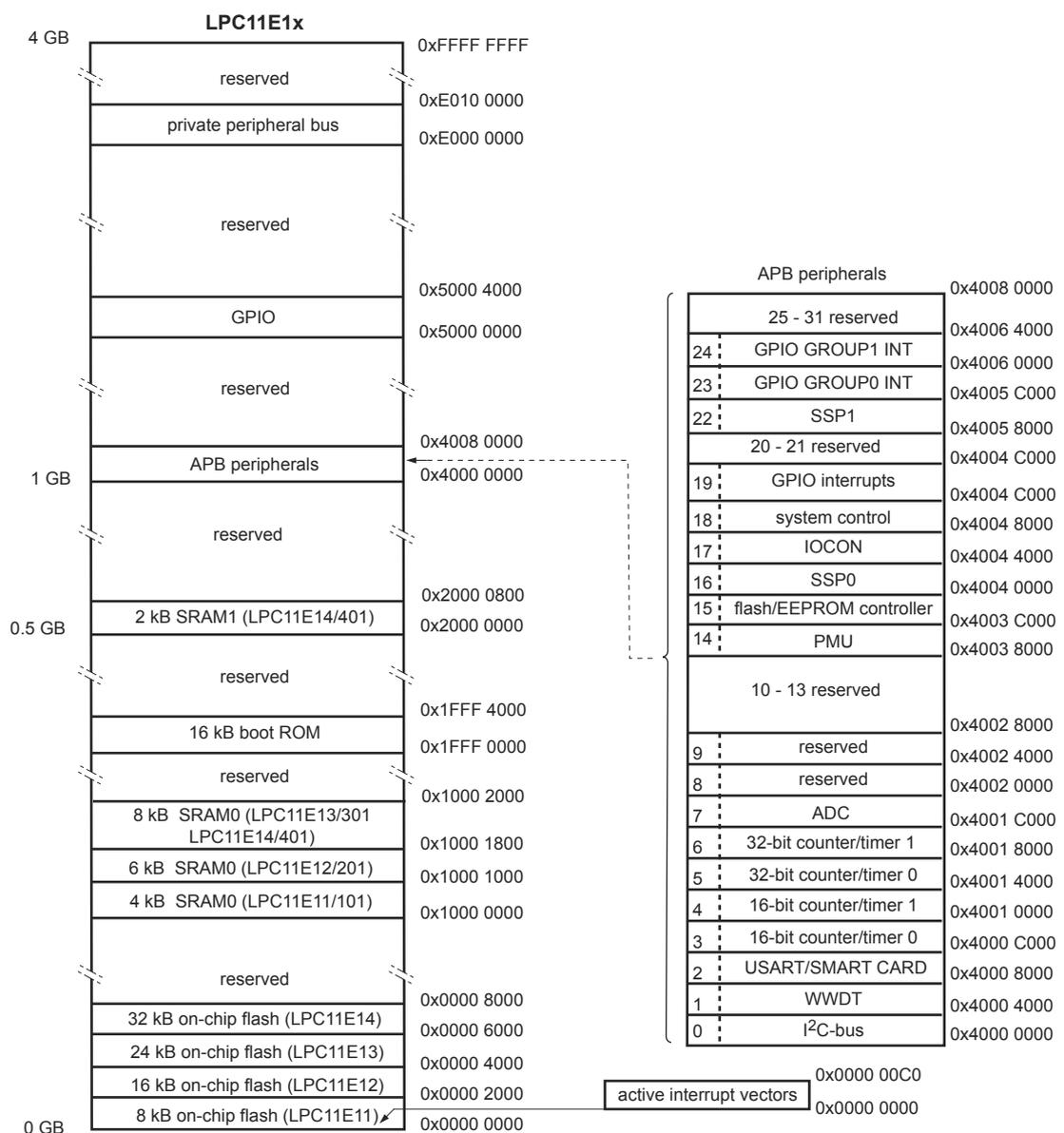


Figura 2.14: Mapa de memoria de la familia LPC11E1x

### 2.2.5. Bloque de Control del Sistema

El módulo denominado Bloque de Control del Sistema (SCB) se encarga del control y la generación de la señal de reloj, así como de algunos aspectos relacionados con la gestión de consumo de potencia. Además, este bloque incluye un registro para remapear las áreas de memoria SRAM, ROM y *flash* del microcontrolador y que no ha sido empleado en nuestro diseño.

A continuación se tratarán con detalle tres de los aspectos más importantes que podemos encontrar en este bloque y que son el sistema de generación del reloj, el *reset* y los modos de bajo consumo disponibles.

#### Sistema de generación del reloj

La figura 2.15 muestra una visión general de la unidad generadora de reloj (CGU, *Clock Generation Unit*). Como se puede apreciar, incluye tres osciladores independientes. Estos son el oscilador del sistema (un cristal de cuarzo externo), un oscilador RC interno de 12 MHz y el oscilador del perro guardián. Cada oscilador puede ser usado para más de un propósito dependiendo de cada aplicación particular.

Tras producirse un *reset*, el LPC11E13 siempre operará desde el oscilador RC interno hasta que dicho oscilador sea conmutado por *software*. Este hecho permite al sistema, y sobre todo al código del *bootloader*, ejecutarse a una frecuencia conocida.

Los principales registros que intervienen en la CGU son los siguientes:

- **SYSPLLCTRL**: Este registro conecta y habilita el sistema PLL y configura los valores de división (PSEL) y multiplicación (MSEL) del PLL. El PLL acepta frecuencias de entrada entre 10 y 25 MHz desde varias fuentes de entrada. Su frecuencia de salida, máxima de 50 MHz, se puede emplear para suministrar el reloj a la CPU, memoria y periféricos.
- **SYSPLLSTAT**: Este registro de sólo lectura (RO, *Read Only*) contiene el bit que informa de si el PLL se encuentra enganchado o no.
- **SYSOSCCTRL**: Este registro permite seleccionar si la entrada del reloj del sistema procede de un cristal de cuarzo o de una señal de reloj externa.
- **WDTOSCCTRL**: Este registro permite configurar la frecuencia del reloj que atacará al perro guardián.
- **SYSPLLCLKSEL**: Este registro permite seleccionar si la señal de reloj de entrada al PLL procede del reloj del sistema o del reloj RC interno (por defecto).

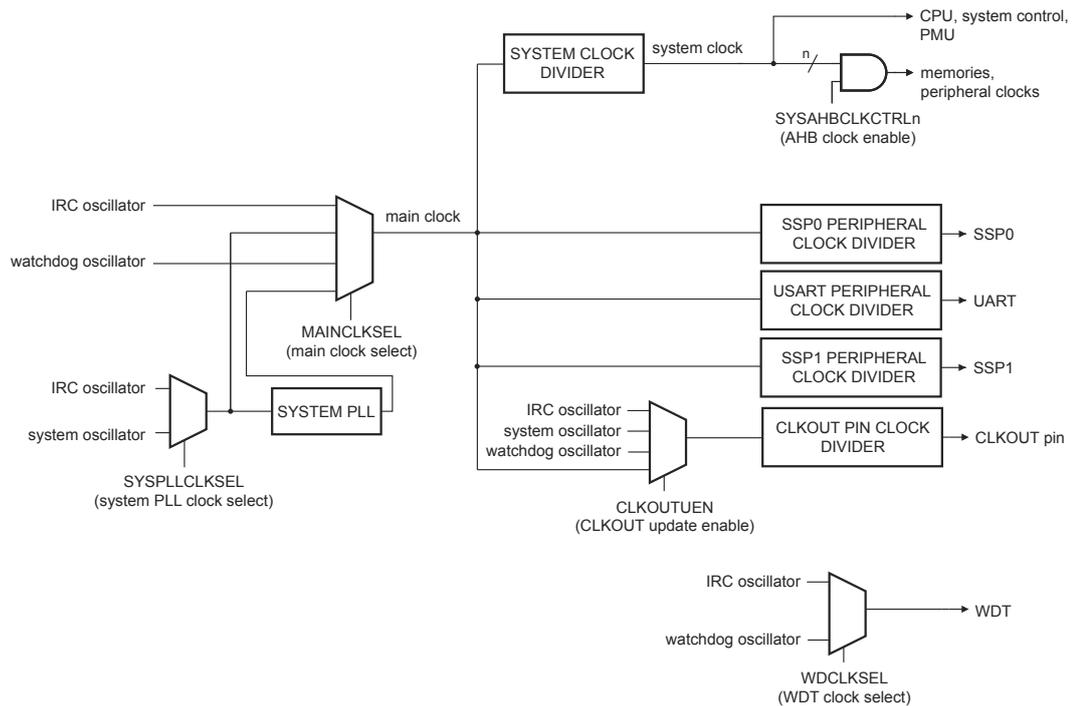


Figura 2.15: Unidad generadora del reloj del LPC111E13

- SYSPCLKUEN: Este registro permite aseverar el cambio realizado en el registro SYSPCLKSEL.
- MAINCLKSEL: Este registro permite seleccionar entre 4 posibilidades qué reloj se suministrará para la CPU, periféricos y memoria. Éstas son: la entrada o salida del PLL, el oscilador RC interno (por defecto) o el oscilador del perro guardián.
- MAINCLKUEN: Este registro permite aseverar el cambio realizado en el registro MAINCLKSEL.
- SYSAHBCLKDIV: Este registro proporciona un factor de división a la señal de reloj seleccionada en MAINCLKSEL
- SYSAHBCLKCTRL: Este registro selecciona a qué periféricos se les será suministrada la señal de reloj resultante de la operación sobre SYSAHBCLKDIV. Aquellos periféricos que por tanto se vayan a emplear, se deben habilitar en este registro.
- PDRUNCFG: Este registro permite despertar tras un *reset*, entre otros, a la CGU que se encuentra inicialmente en modo *Power-down*. Este registro debería ser el último en habilitarse de todos los ya mencionados, ya que se recomienda que la modificación de registros de la CGU tenga lugar con el sistema de reloj inactivo.

- SSP0CLKDIV: Este registro permite configurar la frecuencia a la que funcionará el módulo SSP0 como una fracción de la frecuencia de reloj principal.
- SSP1CLKDIV: Este registro permite configurar la frecuencia a la que funcionará el módulo SSP1 como una fracción de la frecuencia de reloj principal.
- UARTCLKDIV: Este registro permite configurar la frecuencia a la que funcionará el módulo USART como una fracción de la frecuencia de reloj principal.
- CLKOUTSEL: Este registro permite seleccionar qué frecuencia de reloj saldrá a través de la patilla CLKOUT (oscilador RC por defecto).
- CLKOUTUEN: Este registro permite aseverar el cambio realizado en el registro CLKOUTSEL.
- CLKOUTDIV: Este registro proporciona un factor de escala para la señal que salga a través del pin CLKOUT.

Para concluir con este apartado se listan a continuación una miscelánea de registros que permiten configurar una serie de aspectos relevantes en el microcontrolador:

- SYSRSTSTAT: Este registro notifica cuál de las posibles fuentes de *reset* ha sido la que provocó el último *reset* en el sistema.
- NMISRC: Este registro permite seleccionar las interrupciones de un determinado periférico como NMI (la prioridad más alta).
- PINTSEL0-7: Estos registros permiten configurar qué pines de los puertos GPIO se asocian con las ocho posibles fuentes de interrupción. La habilitación definitiva de dichas interrupciones se deberá hacer en la NVIC.
- STARTERP0-1 y PDAWAKECFG: Estos dos registros seleccionan qué pines de los habilitados en los registros PINTSEL0-7 podrán despertar al micro de los diferentes modos de bajo consumo.

## Reset

El *reset* del microcontrolador puede tener lugar de cinco diferentes formas que son:

- Pin  $\overline{RESET}$ : Estando esta patilla habilitada para operar con esta funcionalidad, si se tiene un valor lógico '0'.

- Perro guardián: Estando habilitado, si se produce un *overflow* del temporizador que tiene asociado.
- *Power-On Reset*: Siempre y cuando la tensión de alimentación del microcontrolador no supere cierto nivel.
- *Brown-Out Reset*: Siempre y cuando la tensión de alimentación caiga por debajo de cierto nivel.
- *ARM Software Reset*: Cuando tenga lugar la ejecución de un *reset software*.

Sea cual sea la fuente que provocó el *reset* del sistema, el siguiente proceso es iniciado (figura 2.16):

1. La tensión de alimentación del microcontrolador aumenta progresivamente a medida que pasa el tiempo desde 0V hasta Vdd.
2. Cuando la tensión de alimentación alcanza el valor de 1.8V, el oscilador RC interno se pone en marcha. Después de un tiempo máximo de  $6\mu\text{s}$  proporciona un reloj estable.
3. A continuación la memoria *flash* es alimentada. Esto dura aproximadamente  $100\mu\text{s}$ . Después de este tiempo, la secuencia de inicialización de la memoria *flash* comienza y dura aproximadamente 250 ciclos de reloj.
4. Tras esto, el código de *bootloader* de la memoria ROM comienza. Una vez haya sido ejecutado, se salta a la memoria *flash* para comenzar la ejecución de instrucciones.

Cuando el *reset* interno es retirado, el procesador comienza ejecutando instrucciones desde la dirección 0, que corresponde al vector de *reset* mapeado desde el bloque del *bootloader*. En este punto, todo el microprocesador y los registros de los periféricos han sido ya inicializados a sus valores predeterminados.

### Modos de bajo consumo

En condiciones normales, y siempre tras un *reset*, el microcontrolador se encuentra en estado activo, esto es, el microprocesador Cortex-M0 y las memorias reciben la señal de reloj de la CGU y los periféricos igualmente se encuentran operativos. Sin embargo, en ocasiones puede resultarnos interesante la posibilidad de emplear un modo de bajo consumo con el objetivo de reducir el consumo de potencia del microcontrolador, sobre todo si éste forma parte de un sistema embebido cuyos requisitos de consumo sean muy estrictos.

En este sentido, el LPC11E13 cuenta con cuatro posibles modos de bajo consumo que pasan a detallarse a continuación:

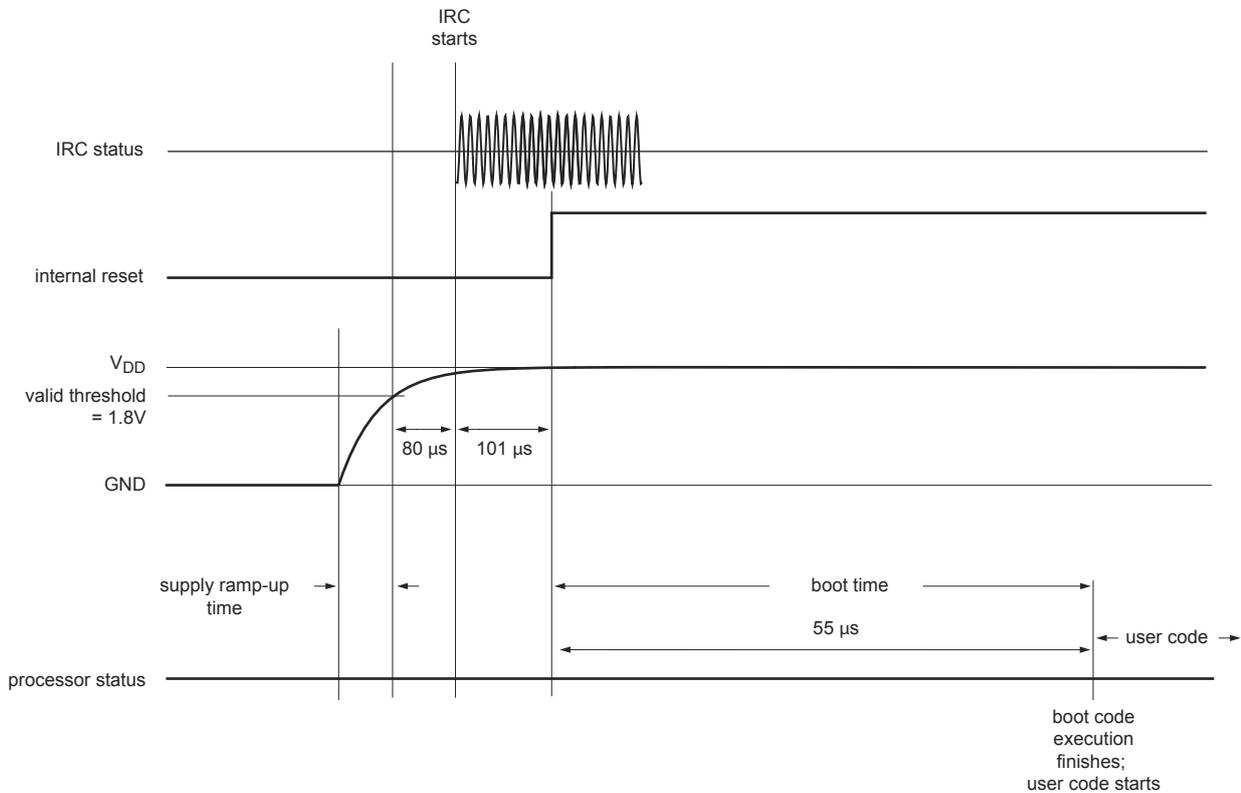


Figura 2.16: Esquema de Reset del microcontrolador LPC11E13

- Modo *Sleep*: En este modo la CPU deja de ejecutar instrucciones hasta que un *reset* o una interrupción ocurra. Todos los periféricos a los que llegaba la señal de reloj siguen funcionando y pueden generar interrupciones. Además, el estado del procesador, los registros de los periféricos, los valores de la memoria SRAM interna y los niveles lógicos de los pines siguen manteniendo su valor.
- Modo *Deep-sleep*: En este modo la CPU deja de ejecutar instrucciones y el reloj principal, y por tanto el reloj de todos los periféricos, se encuentra desactivado. Solamente se encuentran operativos el reloj del perro guardián, si estaba seleccionado, y el oscilador RC, aunque su salida está desactivada. No obstante, el estado del procesador, los registros de los periféricos, los valores de la memoria SRAM interna y los niveles lógicos de los pines siguen manteniendo su valor. Para abandonar este modo interrupciones en el módulo GPIO, perro guardián o BOD deben ocurrir.
- Modo *Power-down*: En este modo la CPU deja de ejecutar instrucciones y el reloj

principal, y por tanto el reloj de todos los periféricos, se encuentra desactivado. Solamente se encuentra operativo el reloj del perro guardián, si estaba seleccionado. El oscilador RC y la memoria *flash* están también desactivados. No obstante, el estado del procesador, los registros de los periféricos, los valores de la memoria SRAM interna y los niveles lógicos de los pines siguen manteniendo su valor. Para abandonar este modo interrupciones en el módulo GPIO, perro guardián o BOD deben ocurrir.

- Modo *Deep Power-down*: En este modo todos los relojes y la alimentación son desactivados excepto para el pin WAKEUP. El estado del procesador, los registros de los periféricos, los valores de la memoria SRAM interna y los niveles lógicos de los pines no retienen su valor. Solamente una pequeña cantidad de datos que puede ser almacenada en un registro del módulo de control de potencia (PMU, *Power Management Unit*) retiene su contenido. La única forma de abandonar este modo de bajo consumo es con un nivel lógico '0' en el pin WAKEUP.

### 2.2.6. Controlador de interrupciones vectorizadas (NVIC)

El módulo NVIC constituye una parte integral del microprocesador Cortex-M0 que se encarga de la gestión de las excepciones y de las interrupciones de los periféricos. Así, soporta hasta 32 interrupciones vectorizadas con 4 niveles de prioridad programables y soporte para interrupciones NMI. Además permite la generación de interrupciones *software*.

En la figura 2.17 se muestra la tabla de vectores de interrupción del microprocesador Cortex-M0. Dicha tabla, que se encuentra situada a partir de la dirección 0x00000000 de la memoria *flash* de programa, contiene el valor de *reset* del puntero de pila (SP, *Stack Pointer*) y la dirección de comienzo de todas las rutinas de excepción/interrupción. El bit menos significativo de cada vector debe ser 1, indicando que las rutinas de excepción están escritas en código *Thumb*.

Los registros que permiten configurar las diferentes posibilidades que ofrece el módulo NVIC se listan a continuación:

- ISER0: Este registro permite habilitar las interrupciones asociadas a los diferentes periféricos así como leer su estado.
- ICER0: Este registro permite deshabilitar las interrupciones asociadas a los diferentes periféricos así como leer su estado.

- ISPR0: Este registro permite cambiar el estado de las interrupciones asociadas a los diferentes periféricos a pendiente, así como leer su estado.
- ICPR0: Este registro permite cambiar el estado de las interrupciones asociadas a los diferentes periféricos a no pendiente, así como leer su estado.
- IABR0: Este registro de sólo lectura se emplea para leer el estado de las diferentes interrupciones asociadas a los periféricos.
- IPR0-7: Estos registros permiten seleccionar cada una de las interrupciones asociadas a los periféricos con hasta 4 diferentes niveles de prioridad.

Exception number	IRQ number	Vector	Offset
47	31	IRQ31	0xBC
.		.	.
.		.	.
.		.	.
18	2	IRQ2	0x48
17	1	IRQ1	0x44
16	0	IRQ0	0x40
15	-1	SysTick	0x3C
14	-2	PendSV	0x38
13		Reserved	
12			
11	-5	SVCall	0x2C
10			
9			
8			
7		Reserved	
6			
5			
4			
3	-13	HardFault	0x10
2	-14	NMI	0x0C
		Reset	0x08
1		Initial SP value	0x04
			0x00

Figura 2.17: Tabla de vectores de interrupción del microcontrolador LPC11E13

Las 32 posibles fuentes de interrupción para los diferentes periféricos, así como su número se listan en la tabla 2.10.

Interrupt Number	Name	Description	Flags
0	PIN_INT0	GPIO pin interrupt 0	-
1	PIN_INT1	GPIO pin interrupt 1	-
2	PIN_INT2	GPIO pin interrupt 2	-
3	PIN_INT3	GPIO pin interrupt 3	-
4	PIN_INT4	GPIO pin interrupt 4	-
5	PIN_INT5	GPIO pin interrupt 5	-
6	PIN_INT6	GPIO pin interrupt 6	-
7	PIN_INT7	GPIO pin interrupt 7	-
8	GINT0	GPIO GROUP0 interrupt	-
9	GINT1	GPIO GROUP1 interrupt	-
10 to 13		-	Reserved
14	SSP1	SSP1 interrupt	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
15	I2C	I2C interrupt	SI (state change)
16	CT16B0	CT16B0 interrupt	Match 0 - 2 Capture 0
17	CT16B1	CT16B1 interrupt	Match 0 - 1 Capture 0
18	CT32B0	CT32B0 interrupt	Match 0 - 3 Capture 0
19	CT32B1	CT32B1 interrupt	Match 0 - 3 Capture 0
20	SSP0	SSP0 interrupt	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
21	USART	USART interrupt	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO) Modem control interrupt
22 to 23	-	-	Reserved
24	ADC	ADC interrupt	A/D Converter end of conversion
25	WWDT	WWDT interrupt	Windowed Watchdog interrupt (WDINT)
26	BOD	BOD interrupt	Brown-out detect
27	FLASH	Flash/EEPROM interrupt	-
28 to 30	-	-	Reserved
31	IOH	IOH interrupt	I/O Handler interrupt

Tabla 2.10: Fuentes de interrupción del microcontrolador LPC111E13

### 2.2.7. Configuración de pines

El microcontrolador LPC11E13 que hemos empleado en nuestro diseño consta de un encapsulado de tipo LQFP48. Se trata de un encapsulado de tipo SMD (*Surface Mount Device*) que consta de 48 pines, 12 a cada lado del mismo (figura 2.18).

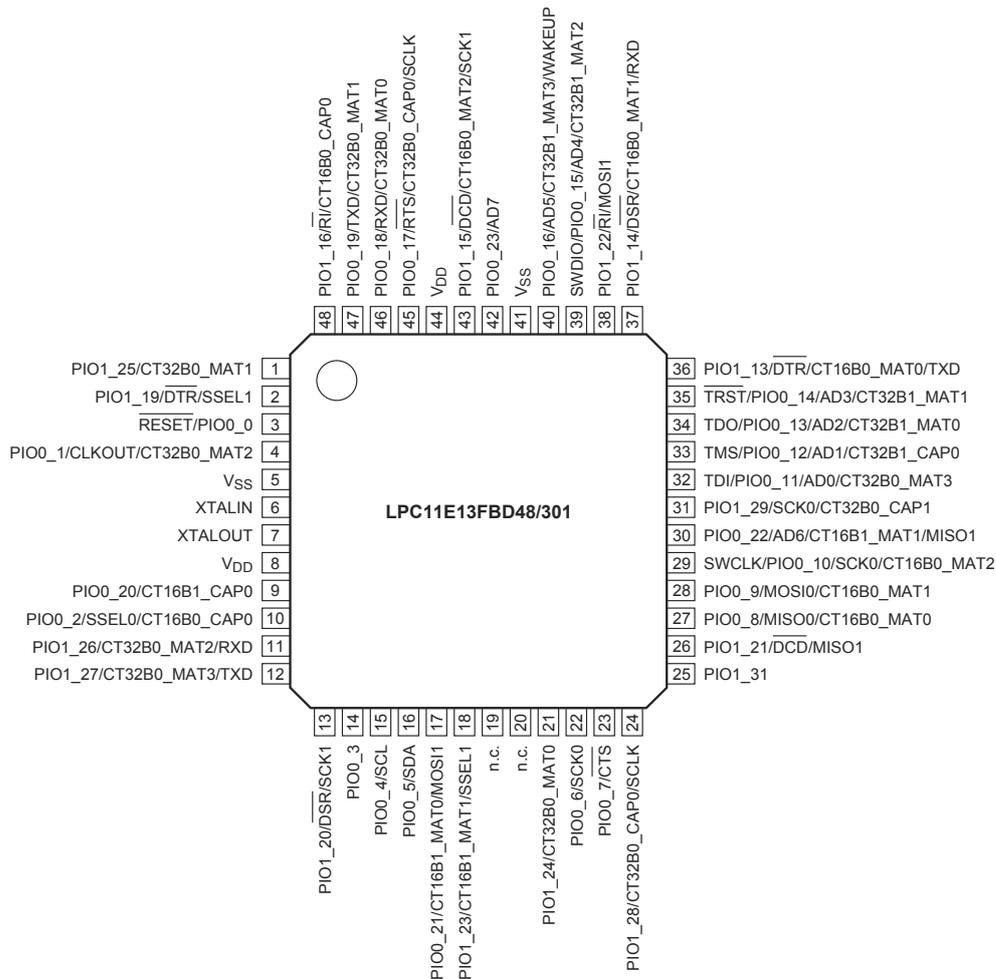


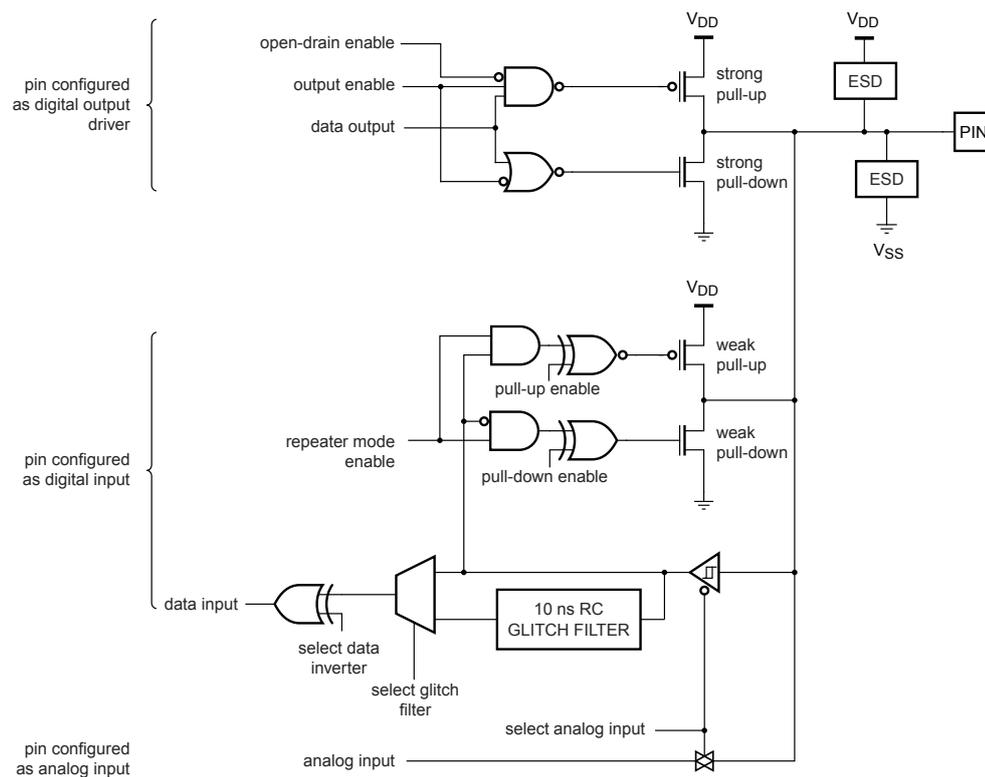
Figura 2.18: Encapsulado del microcontrolador LPC11E13

El conjunto de registros que se emplean para configurar cada uno de estos pines (IOCON) depende del tipo de encapsulado. Esto es así porque la familia de microcontroladores LPC11Exx consta de dispositivos de diferentes números de pines, desde 32 a 64 pines. En nuestro caso concreto, empleamos un encapsulado de 48 pines, y de acuerdo con esto nuestro microcontrolador dispone de dos puertos, a saber:

- Puerto 0: Incluye los pines desde PIO0\_0 hasta PIO0\_23
- Puerto 1: Incluye los siguientes pines:

- Desde PIO1\_13 hasta PIO1\_16
- Desde PIO1\_19 hasta PIO1\_29
- PIO1\_31

Todos estos pines disponen de una configuración similar (figura 2.19) y de unos registros de configuración individuales y donde se pueden configurar diversos aspectos de los mismos. Estas características programables se listan a continuación:



The 10 ns glitch filter is available on selected pins only.

Figura 2.19: Configuración estándar I/O de los pines del microcontrolador LPC11E13

- **Función del pin:** Los bits FUNC en los registros IOCON permiten seleccionar el funcionamiento del pin concreto como GPIO o asociado a un periférico concreto. En caso de seleccionarse la primera opción, será necesario determinar si el pin funcionará como entrada o como salida (se verá en una subsección posterior). En caso de que el pin se encuentre asociado a un periférico, la dirección de dicho pin es controlada automáticamente por el periférico en cuestión.
- **Modo del pin:** Los bits MODE en los registros IOCON permiten seleccionar la posibilidad de configurar los pines con resistencias de *pull-down* o de *pull-up* o en

modo repetición. Por defecto, las resistencias de *pull-up* están habilitadas.

El modo repetición consiste en habilitar la resistencia de *pull-up* si el pin concreto se encuentra en nivel lógico '1' y en habilitar la resistencia de *pull-down* si el pin se encuentra en nivel lógico '0'. Esto permite al pin retener su último valor conocido si es configurado como entrada y dicho pin no es manejado externamente. Es de suma utilidad para evitar que un pin permanezca flotante durante mucho tiempo, sobre todo si no es utilizado con mucha regularidad.

- Histéresis: El bit HYS en los registros IOCON permite seleccionar el *buffer* de entrada para pines digitales con histéresis o sin ella.
- Entrada invertida: El bit INV en los registros IOCON permite invertir la lógica de funcionamiento del pin. De este modo, un nivel lógico HIGH en el pin es invertido a '0', y un nivel lógico LOW es invertido a '1'.
- Drenador abierto: El bit OD en los registros IOCON permite seleccionar un modo de pseudo drenador abierto para pines digitales.
- Filtrado de *glitch*: Algunos pines concretos tienen un bit FILTR en sus registros IOCON que ofrece la posibilidad de activar o no un filtro de 10ns para evitar posibles *glitches*. El pin de *RESET* dispone de uno de 20ns no configurable.

Finalmente cabe decir que aquellos pines que comparten funcionalidad con el conversor analógico digital son más sensibles que el resto. Así, en caso de operar en modo analógico (bit ADMODE en el registro IOCON), la lógica digital del pin es desconectada para obtener un preciso voltaje de entrada para las conversiones. De este modo, los bits HYS, MODE, INV, OD y FILTR no tienen ningún efecto.

### 2.2.8. Entrada/Salida de propósito general (GPIO)

El módulo GPIO abarca 3 funcionalidades separadas que pueden ser configuradas individualmente y que son: la configuración como entrada/salida y la escritura/lectura de los pines, la generación de interrupciones por pines de manera individual, y la generación de interrupciones por pines de manera grupal. En la presente subsección se detallan las dos primeras funcionalidades, ya que son las que se han empleado en nuestro diseño.

Comenzando por las interrupciones que se pueden generar de manera individual en los pines, hay que decir que hasta 8 de éstos pueden ser seleccionados en los registros

PINTSEL0-7 del bloque de control del sistema para generar interrupciones, bien por flanco de subida/bajada o por nivel lógico HIGH/LOW.

Los registros que intervienen en esta funcionalidad se citan a continuación:

- ISEL: Este registro permite seleccionar para cada pin asociado a una interrupción en PINTSEL0-7, si ésta será por flanco o por nivel.
- IENR: Este registro permite habilitar/deshabilitar las interrupciones por flanco (exclusivamente de subida) o habilitar/deshabilitar las interrupciones por nivel.
- SIENR: Este registro permite aseverar lo indicado en IENR. Por tanto, habilita las interrupciones.
- CIENR: Este registro permite de-aseverar lo indicado en IENR. Por tanto, deshabilita las interrupciones.
- IENF: Este registro permite habilitar/deshabilitar las interrupciones por flanco (exclusivamente de bajada) o habilitar las interrupciones por nivel lógico HIGH/LOW.
- SIENF: Este registro permite aseverar lo indicado en IENF en caso de interrupciones por flanco de bajada, o escoger nivel lógico HIGH en caso de interrupciones por nivel.
- CIENF: Este registro permite de-aseverar lo indicado en IENF en caso de interrupciones por flanco de bajada, o escoger nivel lógico LOW en caso de interrupciones por nivel.
- RISE: Este registro permite detectar para los pines configurados como interrupciones por flanco de subida en PINTSEL0-7 si ha sido detectado un flanco de este tipo. Permite además limpiar la notificación en caso de producirse.
- FALL: Este registro permite detectar para los pines configurados como interrupciones por flanco de bajada en PINTSEL0-7 si ha sido detectado un flanco de este tipo. Permite además limpiar la notificación en caso de producirse.
- IST: Este registro permite detectar tanto flancos de subida como de bajada en caso de interrupciones por flanco así como su limpieza. En caso de interrupciones por nivel permite conmutar entre interrupciones por nivel lógico HIGH y LOW.

Por último, la otra funcionalidad que nos resta es la que nos permite configurar cada pin GPIO como entrada/salida y leer su estado, así como modificar el nivel lógico de aquellos

pines que sean configurados como salida.

Los registros que intervienen en este apartado se detallan a continuación:

- B0-31: Estos registros de 8 bits permiten escribir/leer el nivel lógico de un pin del puerto 0 de manera individual.
- B32-63: Estos registros de 8 bits permiten escribir/leer el nivel lógico de un pin del puerto 1 de manera individual.
- W0-31: Estos registros de 32 bits permiten escribir/leer el nivel lógico de un pin del puerto 0 de manera individual.
- W32-63: Estos registros de 32 bits permiten escribir/leer el nivel lógico de un pin del puerto 1 de manera individual.
- DIR0: Este registro de 32 bits permite configurar la dirección de todos los pines del puerto 0.
- DIR1: Este registro de 32 bits permite configurar la dirección de todos los pines del puerto 1.
- MASK0: Este registro de 32 bits permite habilitar la operación de lectura/escritura sobre determinados pines del puerto 0 en el registro MPIN0.
- MASK1: Este registro de 32 bits permite habilitar la operación de lectura/escritura sobre determinados pines del puerto 1 en el registro MPIN1.
- MPIN0: Este registro de 32 bits permite escribir/leer el nivel lógico de todos los pines del puerto 0 que hayan sido habilitados en MASK0.
- MPIN1: Este registro de 32 bits permite escribir/leer el nivel lógico de todos los pines del puerto 1 que hayan sido habilitados en MASK1.
- PIN0: Este registro de 32 bits permite escribir/leer el nivel lógico de todos los pines del puerto 0 sin importar si éstos han sido o no habilitados en MASK0.
- PIN1: Este registro de 32 bits permite escribir/leer el nivel lógico de todos los pines del puerto 1 sin importar si éstos han sido o no habilitados en MASK1.
- SET0: Este registro de 32 bits permite leer el estado de todos los pines del puerto 0, o escribir en ellos solamente el nivel lógico 1.

- SET1: Este registro de 32 bits permite leer el estado de todos los pines del puerto 1, o escribir en ellos solamente el nivel lógico 1.
- CLR0: Este registro de 32 bits de sólo escritura permite escribir en los pines del puerto 0 solamente el nivel lógico 0.
- CLR1: Este registro de 32 bits de sólo escritura permite escribir en los pines del puerto 1 solamente el nivel lógico 0.
- NOT0: Este registro de 32 bits de sólo escritura permite conmutar el nivel lógico de los pines del puerto 0.
- NOT1: Este registro de 32 bits de sólo escritura permite conmutar el nivel lógico de los pines del puerto 1.

A tenor de la gran cantidad de registros de escritura/lectura de que se dispone, queda claro que son múltiples las posibilidades que existen para poder leer/escribir el nivel lógico de un pin. Sin embargo, algunas consideraciones que podrían tenerse en cuenta son las siguientes:

1. Para inicializaciones después de un *reset* conviene trabajar con los registros PIN0-1.
2. Para cambiar el nivel lógico de múltiples pines al mismo tiempo, conviene trabajar con los registros SET0-1 y CLR0-1.
3. Para leer/escribir el nivel lógico de un único pin, conviene trabajar con los registros B0-31, B32-63, W0-31 y W32-63.
4. Para cambiar el nivel lógico de múltiples pines en un entorno como puede ser una máquina de estados, se puede considerar usar los registros NOT0-1.
5. Para tomar una decisión basada en múltiples pines, conviene trabajar con los registros MASK0-1 y MPIN0-1.

### 2.2.9. Temporizadores

El microcontrolador LPC11E13 consta de 4 temporizadores, dos de 16 bits y otros dos de 32 bits, que se pueden emplear tanto para contar tiempo como eventos, así como para modular/demodular señales PWM (*Pulse Width Modulation*) o emplear la modalidad *Free running*.

Las diferentes posibilidades que ofrecen estos temporizadores se listan a continuación:

- 2 canales (temporizadores de 16 bits) o 4 canales (temporizadores de 32 bits) de captura que pueden realizar una copia del valor del temporizador en el momento en que se produce una transición de señal en una entrada concreta.
- Tanto el temporizador como la Pre-escala pueden ser configurados para ser borrados en un determinado evento de captura. Esto permite fácilmente medir la anchura de pulsos borrando el temporizador al detectar un flanco entrante y capturando el valor del temporizador al detectar el flanco saliente.
- 4 registros de coincidencia (*match*) que permiten:
  - Continua operación con generación opcional de interrupción al coincidir.
  - Parada del temporizador al coincidir con generación opcional de interrupción.
  - Reseteo del temporizador al coincidir con generación opcional de interrupción.
- 2 salidas (temporizadores de 16 bits) o 4 salidas (temporizadores de 32 bits) correspondientes a los registros de coincidencia con las siguientes capacidades:
  - Poner nivel lógico LOW al coincidir
  - Poner nivel lógico HIGH al coincidir
  - Conmutar al coincidir
  - No hacer nada al coincidir
- Para cada temporizador, hasta 4 registros de coincidencia pueden ser configurados como PWM.

Un esquema del funcionamiento de los temporizadores del LPC11E13 se puede observar en la figura 2.20.

A continuación, se procede a detallar los registros principales de los temporizadores que intervienen en la capacidad de éstos para contar tiempo. No se detallará pues, el funcionamiento como PWM y como captura, ya que se trata de funcionalidades que no se han desarrollado en nuestro diseño.

- IR: Este registro permite notificar las interrupciones que tienen lugar en los temporizadores debido a captura o coincidencia.
- TCR: Este registro permite habilitar el contador y la Pre-escala para poder contar.

- TC: Este registro se incrementa cada vez que el registro que cuenta el valor de la Pre-escala (PC) alcanza su máximo valor fijado. Es el temporizador en sí mismo.
- PR: Este registro almacena el valor de la Pre-escala.

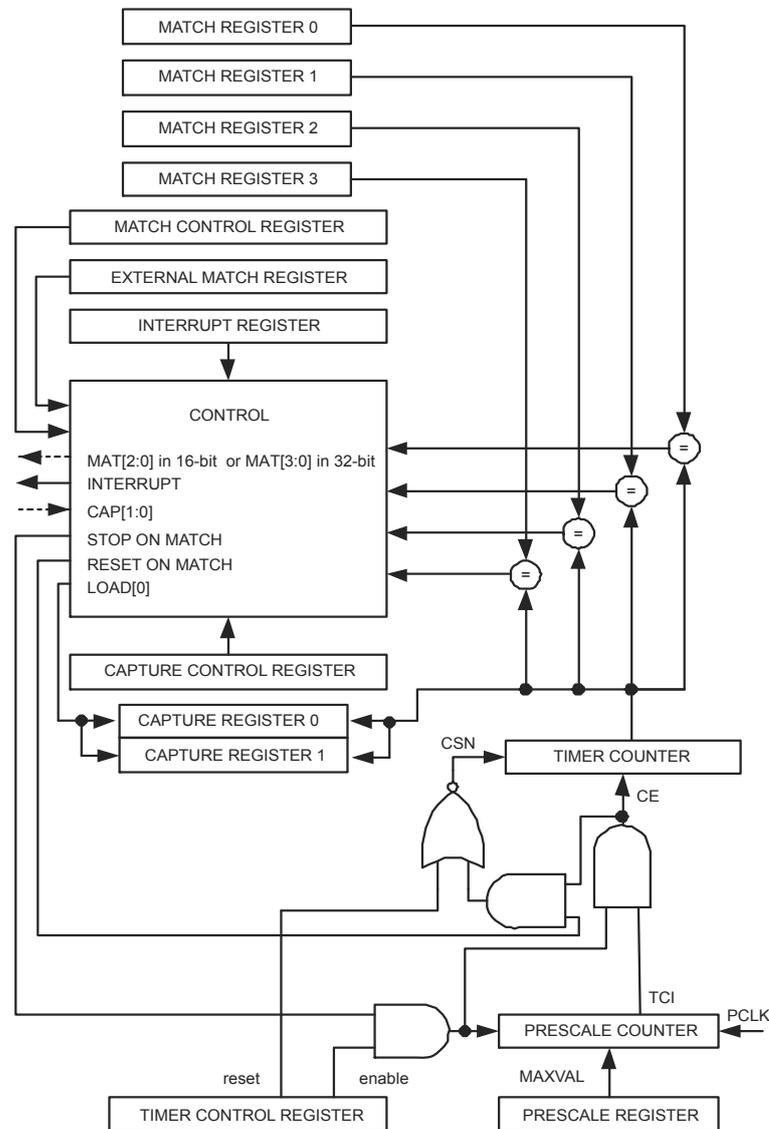


Figura 2.20: Diagrama de bloques de los temporizadores

- PC: Este registro se incrementa con cada ciclo de reloj hasta alcanzar el valor almacenado en PR más 1. En ese momento vuelve otra vez a contar desde el principio.
- MR0-3: Estos registros comparan automáticamente el valor que tienen almacenado con el valor del temporizador en cada instante. En el momento en que se produce una coincidencia se ejecutan las acciones indicadas en el registro MCR.

- MCR: Este registro controla lo que ocurre al existir una coincidencia en los registros MR0-3.
- CTCR: Este registro permite seleccionar si el temporizador contará eventos o tiempo.

Así pues, vistos estos registros, la fórmula que indica cómo el temporizador cuenta tiempo es:

$$T_{\text{temporizado}} = TC * [PR + 1] * t_{\text{peripheralclock}} \quad (2.1)$$

### 2.2.10. Interfaz de periféricos serie (SPI)

Los módulos SPI (SSP0 y SSP1) de que dispone el microcontrolador LPC11E13 son compatible con el protocolo clásico SPI de *Motorola*, pero también pueden funcionar sobre buses *4-wire* de *Texas Instruments* y *National Semiconductor Microwire*. Además, pueden interactuar con múltiples maestros y esclavos sobre el bus, aunque solamente un maestro y un esclavo pueden comunicarse durante una transferencia de datos. Estas transferencias, con tramas de datos que pueden ir desde 4 a 16 bits, son en un principio de tipo *full-duplex*, esto es, con tramas viajando desde el maestro al esclavo y viceversa. No obstante, en la práctica, la mayor parte de la información importante viaja solamente en un sentido. Finalmente, las tramas, una vez llegan a su destino, son almacenadas en sendas FIFO con capacidad para 8 tramas tanto en transmisión como recepción.

Los registros que se emplean para la configuración de este periférico se citan a continuación:

- CR0: Este registro permite configurar todos los aspectos relacionados con la transferencia de los datos. Así, permite fijar el número de bits de la trama a transmitir, el protocolo concreto, la polaridad y fase del reloj, y la Pre-escala de este último.
- CR1: Este registro permite determinar quién será el maestro/esclavo en la comunicación, la habilitación de la comunicación SPI, así como del modo *loop back*.
- DR: Este registro es donde se debe escribir/leer el dato transmitido/leído.
- SR: Este registro de sólo lectura permite comprobar el estado de la comunicación SPI: si se ha recibido o no un dato, si se ha transmitido o no un dato, etc.
- CPSR: Este registro, junto con la Pre-escala en el registro CR1, conforman la Pre-escala que se aplica al reloj que ataca este periférico con el objetivo de fijar la tasa de transmisión a la que tiene lugar la comunicación SPI.

- IMSC: Este registro permite fijar qué eventos producidos en este periférico generarán una interrupción.
- RIS: Este registro de sólo lectura contiene los *flags* de interrupción para las condiciones fijadas en el registro IMSC, tanto si fueron habilitadas como si no.
- MIS: Este registro de sólo lectura contiene los *flags* de interrupción para las condiciones fijadas en el registro IMSC, solamente si fueron habilitadas en este registro.
- ICR : Este registro permite borrar los *flags* de interrupción notificados tanto en RIS como en MIS.

Una vez detallados los registros que intervienen en la configuración del periférico, pasamos ahora a detallar el funcionamiento del protocolo empleado para la comunicación con el módulo de radio, no sin antes exponer la fórmula que permite determinar la tasa de transmisión de los datos a través de la interfaz SPI; que es:

$$SPI_{clock} = \frac{\frac{MAINCLKSEL}{SSPxCLKDIV}}{CPSR * [SCR(CR0) + 1]} \quad (2.2)$$

El protocolo empleado ha sido el formato SPI de *Motorola*. Dicho formato cuenta con una interfaz de 4 líneas que son:

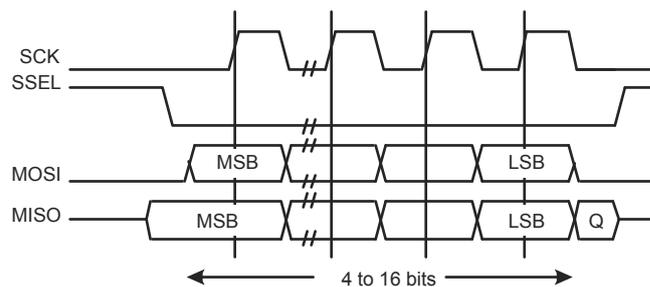
- SCK (*Serial Clock*): Esta línea es por la que se transmite el reloj en sincronismo con los datos. Es transmitido por el maestro y recibido por el esclavo. Durante la transferencia de un dato se transmite como tal, pero cuando cesa la transmisión se abandona en estado inactivo. El valor de dicho estado es configurable en el registro CR0 entre estado lógico HIGH o LOW.
- MOSI (*Master Output Slave Input*): Esta línea transfiere datos desde el maestro hacia el esclavo. Por tanto, es una línea de entrada en el esclavo y una línea de salida en el maestro.
- MISO (*Master Input Slave Output*): Esta línea transfiere datos desde el esclavo hacia el maestro cuando el nivel lógico en SSEL fijado en un registro se cumple. Por tanto, es una línea de entrada en el maestro y una línea de salida en el esclavo.
- SSEL (Slave Select): Esta línea es controlada por el maestro para determinar el esclavo con el que se desea comunicar en cada momento. Al tratarse de una línea dedicada, será necesaria una de éstas por cada esclavo que se encuentre en el bus.

La principal característica del formato SPI de transmisión es que el estado inactivo y fase de la señal de reloj son programables a través de los bits CPOL y CPHA dentro del registro CR0.

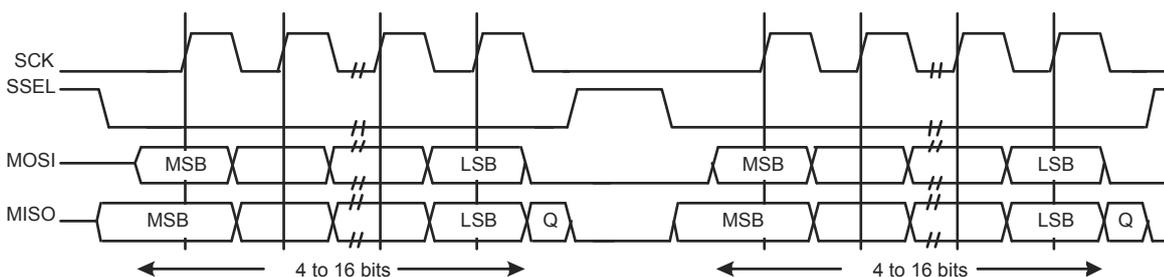
Cuando el bit de control CPOL es puesto en nivel lógico LOW, el estado inactivo de la señal de reloj (cuando no tiene lugar una transferencia) será LOW también. De forma contraria ocurre cuando CPOL tiene un nivel lógico HIGH.

Por otro lado, el bit de control CPHA permite seleccionar el flanco de reloj con el que serán capturados los datos y que permitirá el cambio de los mismos. Así, cuando CPHA está en nivel lógico LOW, el dato será capturado en la primera transición del reloj. Por el contrario, si CPHA está en nivel lógico HIGH, el dato será capturado en la segunda transición del reloj.

En el caso del diseño realizado, la configuración escogida ha sido CPOL=0 y CPHA=0. Esto es así porque es la forma de recibir y procesar las órdenes y datos del módulo de radio CC2500. Un ejemplo tanto de transferencia simple como continua de datos, se puede apreciar en la figura 2.21.



a. Single transfer with CPOL=0 and CPHA=0



b. Continuous transfer with CPOL=0 and CPHA=0

Figura 2.21: Transferencia SPI con formato CPOL=0 y CPHA=0

En esta configuración, durante los periodos de inactividad, la señal de reloj es forzada a LOW, la línea SSEL a HIGH, y las líneas MOSI y MISO a alta impedancia.

Cuando el módulo SPI es habilitado y existe un dato válido en la FIFO de transmisión, el

comienzo de la transmisión es indicado por poner el maestro la línea SSEL a nivel lógico LOW. Esto causa que el esclavo se habilite.

Medio ciclo de reloj después, el dato del maestro es transferido al pin MOSI y el dato del esclavo al pin MISO. Ahora que tanto maestro como esclavo tienen sus datos listos para ser enviados, el maestro habilita la línea SCK.

Los datos son transferidos simultáneamente (de MSB a LSB) del maestro al esclavo y viceversa, capturándose en el flanco de subida y propagándose en los flancos de bajada. En el caso de tratarse de una transferencia simple, una vez que todos los bits de un dato han sido transferidos, la línea SSEL es devuelta al estado lógico HIGH y el resto de líneas vuelven a su estado de inactividad.

Por otro lado, si la transferencia es múltiple, el maestro debe colocar la línea SSEL en nivel lógico HIGH entre cada dato transmitido. Esto es así para que el esclavo pueda habilitar la escritura de datos cada vez que recibe uno nuevo, o si no solamente permitiría la escritura de un único dato.

### 2.2.11. Interfaz USART

El módulo USART de que dispone el microcontrolador LPC11E13 es uno de los más completos que pueden encontrarse en el mercado (figura 2.22). Así, dispone de FIFOs de transmisión y recepción de 16 *bytes* que pueden generar interrupciones al recibir 1, 4, 8 y 14 *bytes*. Además, dispone de flujos de control *hardware* y *software* (RTS/CTS) y de un generador de baudios integrados. Soporta también diferentes modos de transmisión, como pueden ser: modo síncrono, RS-485/EIA-485, ISO 7816-3 para la comunicación con tarjetas inteligentes (*Smart Cards*) e IrDA.

A continuación se describen los principales registros que conforman este periférico, especialmente aquellos relacionados con el protocolo asíncrono clásico propio de las UARTs, ya que ha sido el empleado en el diseño. Éstos son:

- RBR: Este registro permite leer el carácter más 'viejo' recibido que se encuentra en la FIFO de recepción. Para ello, el bit DLAB en el registro LCR debe ser '0'.
- THR: Este registro permite escribir el carácter más 'nuevo' en la FIFO de transmisión. Para ello, el bit DLAB en el registro LCR debe ser '0'.
- DLL: Este registro contiene la parte menos significativa de la Pre-escala que se aplica a la señal de reloj que ataca el periférico, para poder obtener la tasa de baudios deseada. Para poder configurarlo, el bit DLAB en el registro LCR debe ser '1'.

- DLM: Este registro contiene la parte más significativa de la Pre-escala que se aplica a la señal de reloj que ataca el periférico, para poder obtener la tasa de baudios deseada. Para poder configurarlo, el bit DLAB en el registro LCR debe ser '1'.

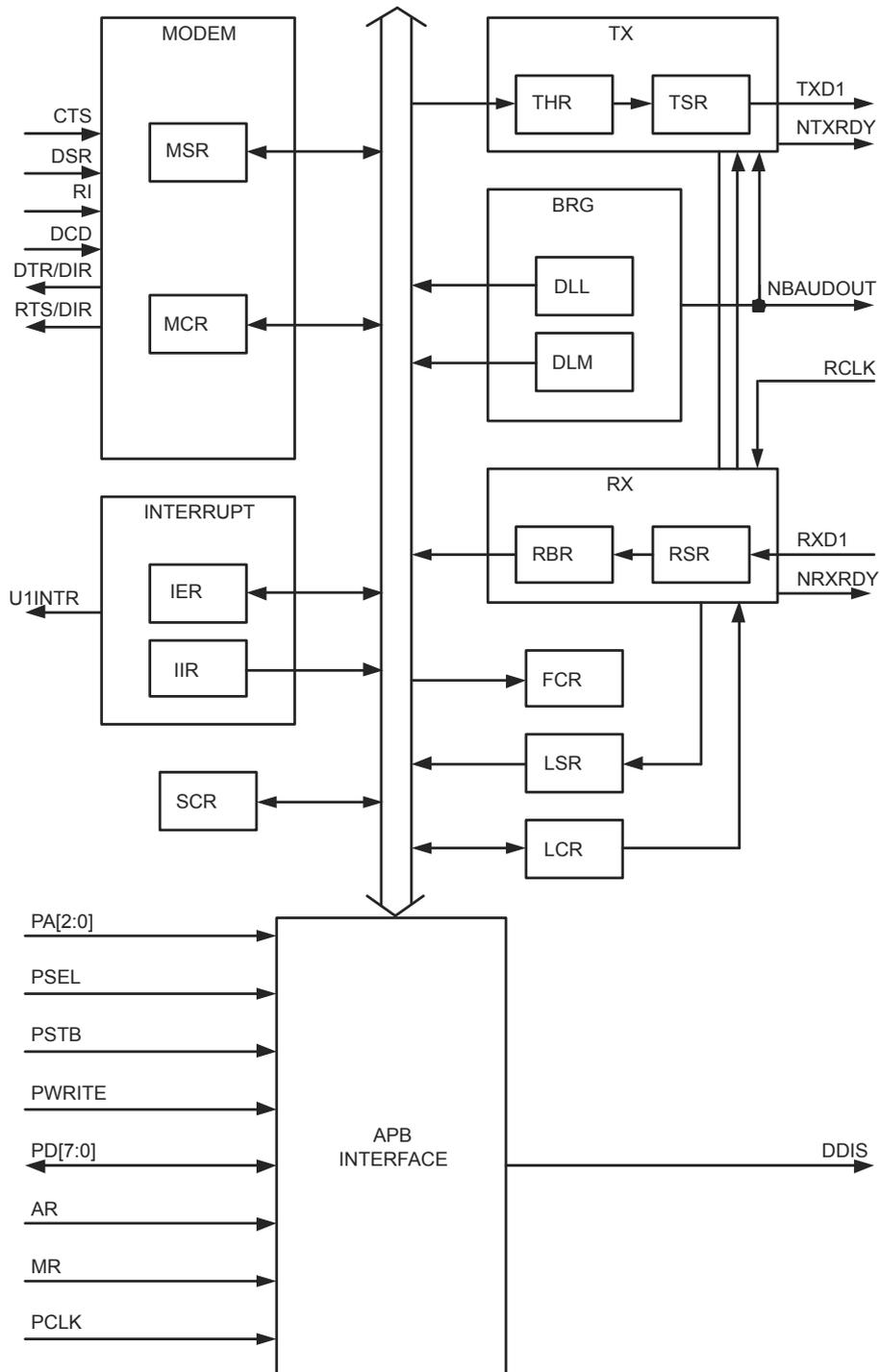


Figura 2.22: Diagrama de bloques del módulo USART

- FDR: Este registro contiene otra Pre-escala adicional a la suministrada por los registros DLL y DLM y se emplea para conseguir tasas de baudios fraccionadas. Se compone de los bits DivAddVal y MulVal.
- IER: Este registro permite habilitar/deshabilitar las diferentes fuentes de interrupción posibles que puede generar la USART. Para ello, el bit DLAB en el registro LCR debe ser '0'.
- IIR: Este registro de sólo lectura denota la prioridad y fuente de una interrupción pendiente.
- FCR: Este registro de sólo escritura permite habilitar/deshabilitar las FIFOs de transmisión y recepción. Además, permite limpiar su contenido e indicar con cuantos caracteres recibidos se debe activar la interrupción por recepción.
- LCR: Este registro permite configurar el formato de la trama a transmitir, i.e. el número de bits a transmitir, el número de bits de Stop y el tipo de paridad. Contiene además el bit DLAB.
- MCR: Este registro permite habilitar el modo *loop-back* y controlar las señales de salida de modem.
- LSR: Este registro de sólo lectura provee información acerca del estado de la transmisión y recepción de la USART.
- MSR: Este registro de sólo lectura provee información sobre el estado de las señales de entrada a la USART.
- ACR: Este registro permite controlar el proceso de medida de la tasa de baudios de una señal de datos entrante.
- TER: Este registro permite habilitar/deshabilitar el módulo USART (habilitado por defecto).

Una vez vistos los registros que componen este módulo, conviene aclarar que la tasa de baudios a la que tiene lugar las transmisiones en la USART sigue la fórmula siguiente:

$$UART_{baudrate} = \frac{\frac{MAINCLKSEL}{UARTCLKDIV}}{16 * [256 * DLM + DLL] * (1 + \frac{DivAddVal}{MulVal})} \quad (2.3)$$

$$1 \leq MulVal \leq 15$$

$$0 \leq DivAddVal \leq 14$$

$DivAddVal < MulVal$

El protocolo asíncrono por excelencia que implementa la USART, y que ha sido desarrollado en el diseño se puede observar en la figura 2.23. Consiste en la transmisión serie de palabras de datos multi-bit en intervalos indeterminados donde cada palabra tiene un número fijo de bits y un tiempo de bit constante. La transmisión siempre tiene lugar comenzando con el bit LSB y finalizando con el MSB.

Tanto emisor como receptor disponen de un reloj para medir el tiempo de bit (conocido por ambos), pero pueden no coincidir exactamente o estar desfasados. Por este motivo, es necesaria una resincronización periódica. El primer bit de cada dato (bit de *Start*) será pues un bit de sincronización. Para finalizar la transferencia de un dato se señala con un bit de stop que puede ser de 1, 1.5 o 2 bits de duración. Además, opcionalmente se puede añadir un bit de paridad entre el bit MSB y el bit de stop para comprobar si la transmisión fue correcta.

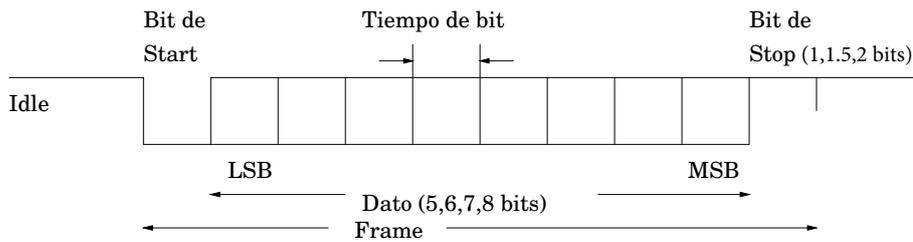


Figura 2.23: Transmisión del módulo USART

## Capítulo 3

# Implementación hardware del proyecto

*Beware of programmers carrying soldering irons.*

~ Anónimo ~

En el presente capítulo se pretende abordar el desarrollo *hardware* del proyecto. Para ello, en las siguientes secciones se tratará qué componentes han sido los seleccionados para cumplir con los requisitos expuestos en el capítulo anterior. Además, se analizarán los diversos esquemáticos producidos, donde pueden observarse las conexiones eléctricas de los diferentes elementos, así como el listado de materiales (BOM, *Bill Of Materials*). Finalmente, se abordará el diseño de la placa de circuito impreso producida (PCB, *Printed Circuit Board*) donde el rutado de las pistas y el *placement* de los componentes serán cuestiones importantes para poder desarrollar un sistema que no presente, en la medida de lo posible, problemas de compatibilidad electromagnética.

### 3.1. Elección de los componentes

A continuación paso a comentar el por qué de la elección de los cuatro circuitos integrados que forman parte del diseño realizado:

#### 3.1.1. Microcontrolador LPC11E13

Las principales premisas en cuanto a la elección del microcontrolador eran básicamente dos: bajo coste y bajo consumo. En este sentido, el microcontrolador seleccionado cumple con ambas características.

Para su elección se inició una búsqueda de microcontroladores que emplearan una CPU

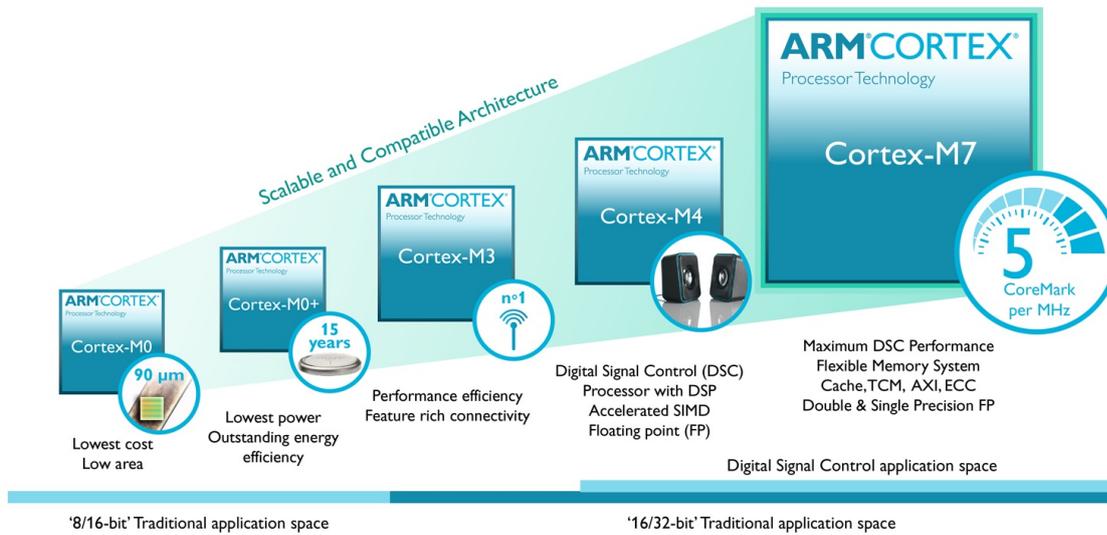


Figura 3.1: Familia de microprocesadores ARM Cortex-M

de ARM, puesto que éstas son las mejores en cuanto a consumo de potencia al estar optimizadas para la implementación de aplicaciones embebidas. Dentro de las muchas familias existentes de arquitecturas en ARM, se optó por la familia Cortex-M (figura 3.1), pues como se indica en su página web [10] se trata de la familia ideal para aplicaciones que requieran de dispositivos embebidos fáciles de usar, que no necesiten de grandes velocidades de proceso y consumo de recursos, y que precisen de un bajo coste y eficiencia energética. Finalmente, dentro de esta familia se optó por un Cortex-M0, pues es más que suficiente para la aplicación que se pretende desarrollar.

Así pues, consultando en la página web del proveedor seleccionado para la compra de componentes, *Farnell* [11], un microcontrolador que emplease dicho microprocesador, se encontró el LPC11E13 del fabricante *NXP* y el cual cuenta con todos los periféricos necesarios para el desarrollo de la aplicación.

### 3.1.2. Módulo de radio ANAREN

Aunque el módulo tranceptor de radio no constituye una elección de nuestro diseño, ya que en las especificaciones se fijó que debía emplearse el circuito integrado CC2500 de *Texas Instruments*, lo cierto es que la manipulación del mismo, debido a sus reducidas dimensiones, lo hacen muy complicado de montar sobre la PCB. Esto es así, por que al tratarse de un prototipo todos los componentes se van a montar a mano. Además, por otra parte, se requeriría del diseño de una antena empleando una pista de cobre, donde habría que ajustar la impedancia de la misma y realizar un estudio de su patrón

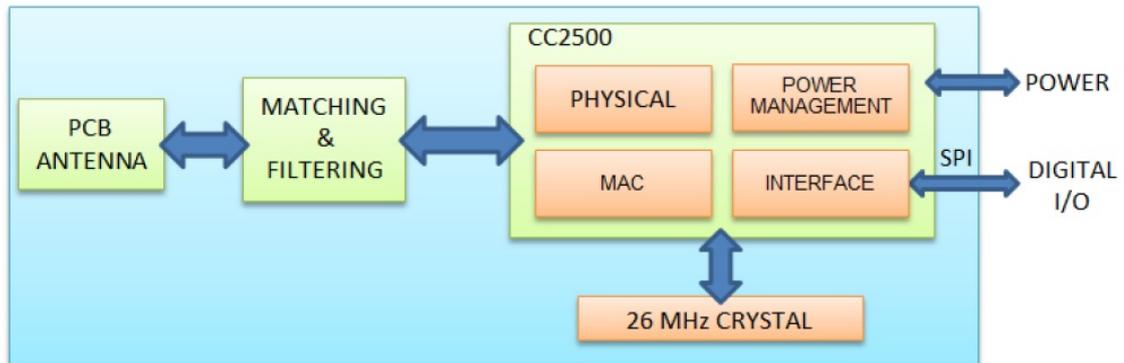


Figura 3.2: Diagrama de bloques del módulo de radio Anaren A2500R24A

de radiación para comprobar que emite en la frecuencia adecuada. Todo ello complicaría el diseño a realizar.

Por todo lo dicho, lo que se ha hecho es buscar un módulo de radio que implemente en un mismo circuito integrado todas las prestaciones necesarias para poder realizar una comunicación via radio. En este sentido, el circuito integrado escogido ha sido el A2500R24A del fabricante *Anaren*. Este módulo, como se puede observar en la figura 3.2 cuenta con un cristal de cuarzo de 26 MHz integrado, un regulador de tensión, el propio circuito C2500, y una red de impedancias que cumple la función de filtrar y adaptar la impedancia de salida del C2500 a la antena integrada que tiene forma de diapasón. Esta antena ha sido diseñada para proveer una alta eficiencia y un patrón de radiación próximo a la omnidireccionalidad. Esta característica junto con el hecho de que el módulo opera en la banda de frecuencia no licenciada de 2.4 GHz, convierten a este dispositivo en el ideal para el desarrollo de aplicaciones que requieren de conectividad via radio sin tener que tratar con caras antenas de radiofrecuencia y normativas de regulación sobre la zona del espectro radioeléctrico empleada. Además, dicho módulo cumple con las normativas ETSI de emisiones electromagnéticas que tienen vigencia en Europa y que hacen que cumpla con la legislación vigente.

### 3.1.3. Convertidor USB-UART FTDI

Para poder comunicar la placa PCB donde se ha de montar el módulo de radio con el PC donde reside la aplicación que gestiona las órdenes que se mandan, es necesario disponer de una interfaz de comunicaciones en el diseño ha realizar para poder comunicar ambos elementos. Para poder llevar a cabo esto, se podría pensar en emplear el periférico USART del microcontrolador por un lado, y por otro el puerto serie de un PC. Sin

embargo, esta solución presenta diversos inconvenientes, a saber:

- Sería necesario el empleo de algún circuito convertidor de niveles para poder ajustar los niveles de tensión de salida del microcontrolador y los procedentes del PC, e.g. un circuito MAX232 que permite obtener niveles de tensión compatibles con la normativa de comunicaciones RS232, y los elementos pasivos necesarios. Esto supone un incremento de los circuitos a emplear y por tanto un coste mayor.
- No solo necesitamos en nuestro diseño poder comunicar microcontrolador y PC, sino que además tenemos que poder programar la memoria *flash* de dicho microcontrolador, lo cual no se consigue con el módulo USART solamente.
- Hoy en día, con el gran avance tecnológico que se ha alcanzado, son cada vez menos los ordenadores de sobremesa que incorporan un puerto serie físico, llegando incluso a desaparecer casi en su totalidad en los ordenadores portátiles. Es por ello, que una solución basada en un puerto serie podría llegar a estar obsoleta en relativamente poco tiempo.

Por todo lo dicho, la solución buscada ha consistido en emplear un circuito integrado que convierta el protocolo USB en un protocolo serie compatible con UART. El CI utilizado ha sido el FTDI232RL del fabricante *FTDI Chip*, el cual simplifica los diseños basados en conversión USB a serie y reduce el empleo de componentes externos al incluir de manera integrada todo lo necesario para poder realizar dicha conversión. Sus principales componentes son (figura 3.3):

- Memoria EEPROM interna: Se emplea para almacenar la configuración concreta del convertidor, incluyendo la de los pines CBUS. Dicha configuración puede ser cambiada empleando una utilidad *software* denominada MPROG, que puede ser descargada desde la página web del proveedor [12].
- Transceptor USB: Se encarga de proveer la interfaz física USB 2.0 al cable USB.
- Motor de Interfaz Serie: Se encarga de convertir los datos USB de formato paralelo a serie y viceversa. Se encarga además de emplear la técnica de *bit stuffing* y la generación del código de redundancia cíclica de tipo CRC15/CRC16 sobre los datos enviados, así como la correcta recepción de este último en recepción.
- Motor de Protocolo USB: Se encarga de llevar a cabo el control de flujo sobre los datos USB, así como generar las peticiones y órdenes llevadas a cabo por el *USB host controller*.



A la hora de escoger un regulador de tensión para una aplicación como la nuestra, que requiere de un bajo consumo, sobre todo si el sistema está conectado a una batería, dos son los parámetros principales en los que nos debemos de fijar a la hora de decantarnos por uno u otro, a saber:

- *Dropout Voltage*: Este voltaje se refiere a la diferencia de tensiones ,entre la salida y la entrada, más pequeña para poder mantener la regulación. Por ejemplo, si en un regulador, para obtener a la salida una tensión de 3.3V se necesita tener como mínimo a la entrada 3.4V, eso significa que se tiene un *dropout voltage* de 100mV. Esta tensión, por tanto, constituye el voltaje que cae a través de las diferentes uniones en los transistores y demás dispositivos de estado sólido que conforman el regulador.
- *Quiescent Current*: Esta corriente se refiere a la diferencia entre la corriente de entrada al regulador y la de salida. Por tanto, constituye la mínima corriente necesaria para poder polarizar adecuadamente los diferentes elementos que componen el regulador.

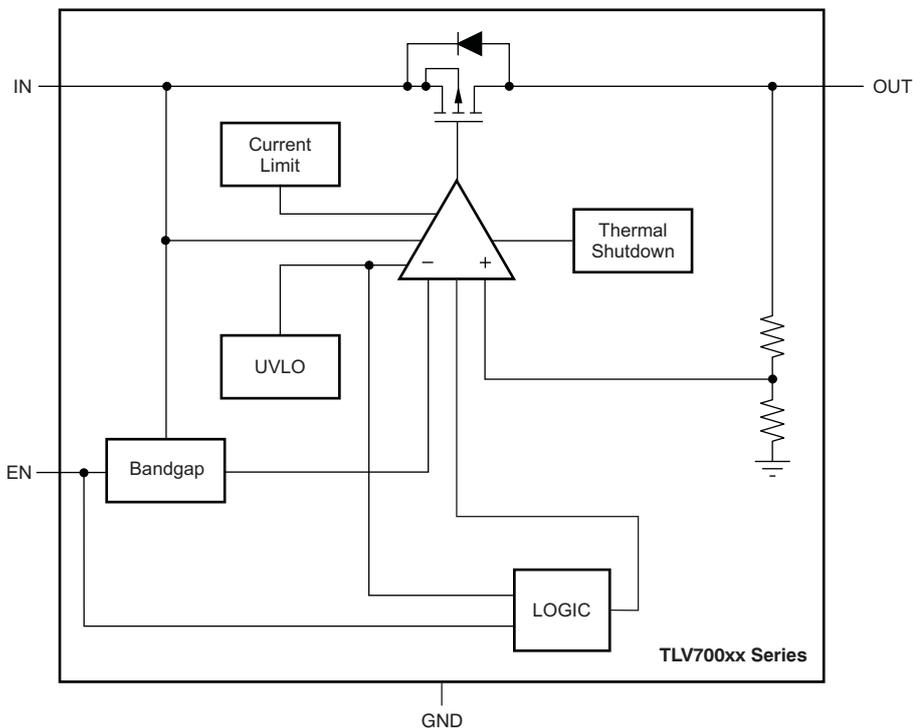


Figura 3.4: Diagrama de bloques del regulador de tensión TLV70033

Como queda claro de las definiciones anteriores, tanto el *dropout voltage* como la *quiescent current* constituyen una potencia disipada exclusivamente por el mero hecho de disponer

de un regulador y que no aporta nada a su salida. Por este motivo, en la elección del regulador de tensión para nuestra aplicación se ha escogido uno que tiene unos valores muy bajos de ambos parámetros.

El regulador de tensión escogido ha sido el TLV70033 del fabricante *Texas Instruments*, el cual presenta un valor de *dropout voltage* de unos 100mV y un valor de *quiescent current* de unos 31 $\mu$ A, valores muy por debajo del de otras familias de reguladores. Con esto conseguimos disminuir en la medida de lo posible el consumo de potencia en el regulador, y que el consumo real tenga lugar solamente a través del microcontrolador y el módulo de radio.

Un diagrama de bloques del regulador de tensión escogido se puede observar en la figura 3.4.

## 3.2. Elaboración de los esquemas electrónicos

En la presente sección se pretende explicar el conexionado realizado entre los diferentes elementos que conforman la aplicación, para lo cual se abordará el sistema total en tres partes bien diferenciadas que se detallarán en las siguientes subsecciones. Todos los esquemáticos producidos han sido elaborados con la herramienta Proteus ISIS [13] de la empresa *Labcenter Electronics*. Los esquemáticos completos se pueden consultar en el apéndice B.

### 3.2.1. Red de conversión USB-UART

En la figura 3.5 se puede observar el esquemático de esta primera parte del diseño. En ella se puede observar como se ha realizado la conexión entre el conector USB mini-B y el circuito integrado FTDI.

A continuación se detalla el por qué de las conexiones y elementos empleados:

- Las líneas de datos que salen del conector USB así como las de alimentación y tierra se han conectado a las líneas equivalentes en el integrado FT232. De esta forma, el PC podrá conectarse con el microcontrolador y el módulo de radio para comunicarse.
- La resistencia R1 y el condensador C1 constituyen un filtro que previene de posibles descargas electrostáticas que puedan producirse sobre la parte metálica del conector USB mini-B. De este modo, cualquier descarga se dirige a tierra rápidamente.

- Los condensadores C3 a C7 constituyen condensadores de desacoplo de la alimentación del integrado FT232. Estos condensadores tienen como objetivo el de desviar los pulsos de corriente que puedan producirse debido a la conmutación de la lógica digital que constituye el integrado hacia tierra. Con ello se consigue que no se produzca una caída de tensión en el carril de la alimentación, lo cual podría hacer que la tensión efectiva que llegase al integrado fuese inferior a la necesaria para poder hacer que el CI funcione adecuadamente.
- El condensador C2 junto con la bobina de choke L1 constituyen un filtro que tiene como objetivo evitar que posibles interferencias electromagnéticas procedentes del FT232 se dirijan al conector USB. A su vez, también permite que posibles descargas electrostáticas que se dirijan al FT232 se atenúen antes de llegar a él.

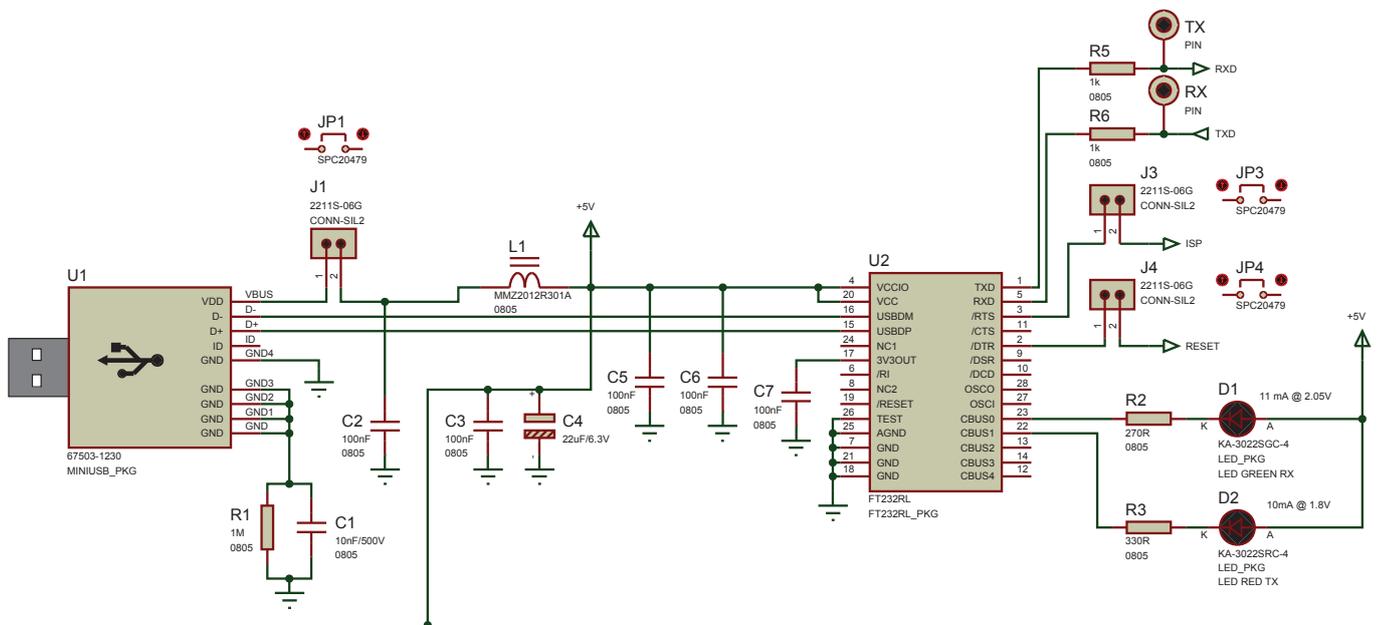


Figura 3.5: Red de alimentación y circuito de conversión USB-UART

- El *jumper* J1 tiene como objetivo facilitar la medida del consumo de corriente total que tiene lugar en el conjunto de la PCB. Así, retirando el conector JP1 y conectando en su lugar un amperímetro en serie podemos medir fácilmente el consumo de corriente de nuestra placa.
- Las líneas de transmisión (TXD) y recepción (RXD) del integrado FT232 se han conectado a través de dos resistencias a los respectivos pines de transmisión y recepción del microcontrolador, permitiendo así comunicar la interfaz serie del integrado

FT232 con el periférico UART del microcontrolador. Las resistencias se han situado para evitar posibles cortocircuitos en caso de que uno de los pines se encuentre en estado lógico alto y el otro en estado lógico bajo. Además se han añadido dos puntos de testeo en dichas líneas para permitir su depuración en caso de que sea necesario.

- Los *jumpers* J3 y J4 se emplean para conectar los terminales  $\overline{RTS}$  y  $\overline{DTR}$  del integrado FT232 con los pines de programación en el sistema (ISP, *In-System Programming*) y  $\overline{RESET}$  respectivamente. Estas dos líneas permiten que un programa se pueda almacenar en la memoria *flash* del microcontrolador. Para ello el FT232 mantiene la línea de ISP en nivel lógico bajo al tiempo que provoca el reset del microcontrolador. Esto indica al *bootloader* del microcontrolador que lo que reciba a través del puerto serie lo debe almacenar en *flash*. Pues bien, esto lo hará el FT232 cada vez que reciba datos procedentes del puerto USB. De esta manera, los conectores JP3 y JP4 sirven para ser retirados en el momento en que se grabe el programa en el microcontrolador para evitar que se resetee cada vez que reciba datos.
- Finalmente, los diodos Led D1 (verde) y D2 (rojo) se emplean para señalar la transmisión y recepción de datos a través del puerto serie, a modo de depuración para comprobar que efectivamente tiene lugar la comunicación.

### 3.2.2. Red de regulación de tensión

En la figura 3.6 se puede observar cómo se ha llevado a cabo el conexionado del regulador de tensión en el diseño realizado. Básicamente se tiene que:

- Los *jumpers* J5 y J6 se pueden emplear, si así se desea, para alimentar el módulo de radio y el microcontrolador desde una batería en vez de la conexión al puerto USB. Para ello, retirando el conector JP2 y situando los bornes de la batería en los correspondientes extremos del *jumper* J6 esto se puede conseguir. Esta conexión tendrá sentido realizarla sobre los módulos de radio esclavos cuando debido a su situación no se disponga de una toma USB cercana.
- Los condensadores C8 y C9 son necesarios para ayudar a mejorar el comportamiento del regulador de tensión. Así, el condensador C8 mejora la respuesta transitoria, el rechazo del ruido y el rizado a la entrada del regulador, mientras que el condensador C9 permite estabilizar el nivel de tensión a la salida del mismo.

- Finalmente, el diodo LED D3 (naranja) se emplea como un diodo testigo de que efectivamente está llegando tensión a la salida del regulador. Dado que representa un diodo que consume potencia de manera continua, se ha escogido uno de bajo consumo de corriente. No obstante, en caso de realizar una conexión de la placa a una batería, convendría no montar dicho componente para alargar en la medida de lo posible la vida útil de la misma.

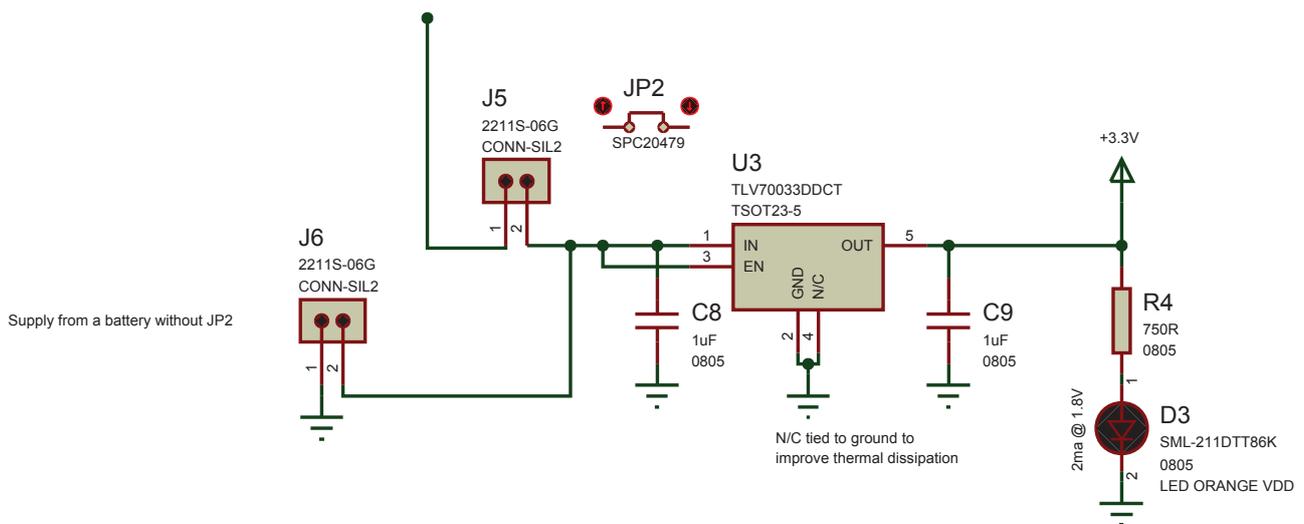


Figura 3.6: Red de regulación de tensión en el diseño realizado

### 3.2.3. Red del microcontrolador y módulo de radio

Finalmente, el último bloque que compone el diseño lo constituye la comunicación entre el microcontrolador con el módulo de radio y con los dispositivos externos a controlar. En la figura 3.7 se pueden observar las conexiones realizadas y componentes empleados que son:

- Los condensadores C10 y C11 se emplean como desacoplo de las alimentaciones que posee el microcontrolador con la función que ya se comentó anteriormente.
- El pulsador B2, el diodo D4, la resistencia R9 y el condensador C12 constituyen el circuito de *reset* manual del microcontrolador. Así, el diodo se emplea a modo de protección para evitar que se supere la tensión de 3.3V en el pin de *reset*, y tanto la resistencia de 1MΩ como el condensador de 100nF se encargan de fijar una constante de tiempo  $\tau$  de unos 100ms que permita garantizar un *reset* adecuado del microcontrolador cada vez que se pulsa el botón B2.

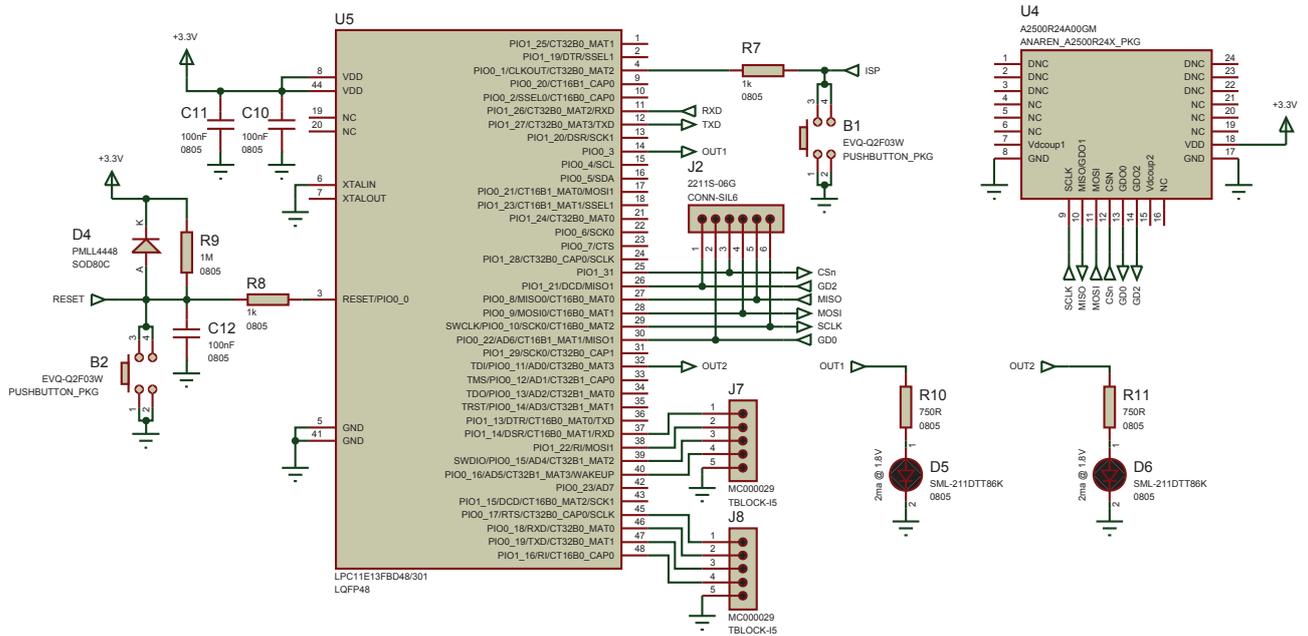


Figura 3.7: Red de comunicación externa y via radio

- El pulsador B1 se emplea para situar un nivel lógico bajo a la entrada del pin 4 del microcontrolador, que corresponde a la entrada ISP del mismo. De este modo, se pretende que se pueda programar su memoria *flash* de otro modo alternativo al de emplear el convertidor FT232. Para ello, será necesario pulsar y mantener pulsado este botón, al tiempo que se activa el circuito de *reset* (pulsador B2).
  - Las conexiones existentes en el conector J2 se emplean a modo de puntos de testeo sobre las líneas que conectan el microcontrolador con el módulo de radio. Así, se permite una fácil depuración del puerto SPI como de los dos pines de salida que posee el integrado ANAREN.
  - Los diodos D5 y D6 se emplean de igual modo como elementos de depuración, permitiéndonos saber si los pines a los que se conectan tienen los niveles lógicos fijados.
  - Finalmente los conectores de bloques J7 y J8 serán los encargados de establecer los niveles lógicos adecuados en función de las órdenes recibidas via radio. De este modo, permitirán dotar de tensión a los elementos externos (relés, etc.) que se conecten a sus entradas.
- Estos bloques solamente tendrán sentido en el caso de los módulos esclavo, pues el módulo maestro nunca recibirá órdenes de ningún tipo.

### 3.2.4. Listado de materiales

#### Bill Of Materials For Sistema De Comunicación Via Radio

**Design Title** : Sistema de Comunicación via Radio  
**Author** : Javier Santos Romo  
**Revision** : 1  
**Design Created** : miércoles, 24 de diciembre de 2014  
**Design Last Modified** : sábado, 28 de febrero de 2015  
**Total Parts In Design** : 51

#### 11 Resistors

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
2	R1, R9	1M	0805	9332413	Farnell
1	R2	270R	0805	9332936	Farnell
1	R3	330R	0805	9333037	Farnell
3	R4, R10, R11	750R	0805	9333525	Farnell
4	R5-R8	1k	0805	9332383	Farnell

#### 12 Capacitors

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
1	C1	10nF/500V	0805	1886013	Farnell
8	C2, C3, C5-C7, C10-C12	100nF	0805	1759144	Farnell
1	C4	22uF/6.3V	C_ELEC_PKG	1973274	Farnell
2	C8, C9	1uF	0805	1759429	Farnell

#### 5 Integrated Circuits

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
1	U1	67503-1230	MINIUSB_PKG	2313554	Farnell
1	U2	FT232RL	FT232RL_PKG	1146032	Farnell
1	U3	TLV70033DDCT	TSOT23-5	1778230	Farnell
1	U4	A2500R24A00GM	ANAREN_A2500R24X_PKG	2096127	Farnell
1	U5	LPC11E13FBD48/301	LQFP48	771-LPC11E13FBD48301	Mouser Electronics

#### 6 Diodes

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
1	D1	KA-3022SGC-4	LED_PKG	1142615	Farnell
1	D2	KA-3022SRC-4	LED_PKG	1142617	Farnell
3	D3, D5, D6	SML-211DTT86K	0805	1685055	Farnell
1	D4	PMLL4448	SOD80C	1097279	Farnell

#### 2 Miscellaneous

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
2	RX, TX	PIN	PIN		

#### 8 Connectors

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
5	J1, J3-J6	2211S-06G	CONN-SIL2	1593415	Farnell
1	J2	2211S-06G	CONN-SIL6	1593415	Farnell
2	J7, J8	MC000029	TBLOCK-15	2007998	Farnell

#### 1 Inductors

<u>Quantity</u>	<u>References</u>	<u>Value</u>	<u>Package</u>	<u>Reference Code</u>	<u>Provider</u>
1	L1	MMZ2012R301A	0805	1669724	Farnell

2 Buttons					
Quantity	References	Value	Package	Reference Code	Provider
2	B1, B2	EVQ-Q2F03W	PUSHBUTTON_PKG	2460018	Farnell
4 Jumpers					
Quantity	References	Value	Package	Reference Code	Provider
4	JP1-JP4	SPC20479	NULL	2396301	Farnell

sábado, 28 de febrero de 2015 14:14:41

Tabla 3.1: Listado de materiales empleados en la aplicación

Una vez analizados todos los componentes que forman parte de nuestro proyecto, una lista con todos ellos, así como sus proveedores y códigos de referencia, se puede observar en el BOM de la tabla 3.1.

### 3.3. Layout de la PCB

En esta sección se pretende describir cómo se ha llevado a cabo el rutado y diseño de la PCB de nuestra aplicación. Para ello, en las siguientes subsecciones se detallarán los diferentes procedimientos y *guidelines* seguidos para lograr con éxito el propósito fijado. Además, para su obtención nos hemos servido de la herramienta Proteus ARES de la empresa *Labcenter Electronics* [14]. Las diferentes capas de *layouts* obtenidas con esta herramienta se pueden consultar en el apéndice C.

#### 3.3.1. Requerimientos en el diseño de la PCB

Antes de comenzar propiamente con el rutado y *placement* de los componentes, es necesario conocer las imposiciones que impone, sobre cómo han de ser ciertos aspectos del diseño, el fabricante al cual va a encargársele la producción de la PCB. Así, en nuestro caso, el fabricante escogido ha sido Itead Studio [15]. Dicha elección se fundamenta en la gran calidad del producto final elaborado junto con el bajo coste asociado al mismo (10 placas de 10cm x 10cm a doble cara por unos 30\$).

Entre las restricciones más importantes que hay respetar para que la fabricación tenga lugar sin ningún inconveniente se encuentran las siguientes:

- Las dimensiones de la PCB en ningún caso podrán ser superiores a 10cm x 10cm.
- La mínima anchura posible de las pistas de cobre que recorran el circuito es de 6th (0.15mm).

- La mínima separación posible entre dos pistas de cobre que recorran el circuito es de  $6t$ (0.15mm).
- La mínima distancia de separación entre cualquier pista de cobre que recorra el circuito y el borde de la placa es de  $12t$ (0.3mm).
- El menor diámetro posible para cualquier vía o agujero presente en el circuito es de  $12t$ (0.3mm).
- La mínima distancia de separación entre cualquier pista de cobre que recorra el circuito y un agujero o vía es de  $10t$ (0.25mm).
- La mínima distancia de separación entre cualquier agujero o vía presente en el circuito y el borde de la placa es de  $16t$ (0.4mm).
- El mínimo espesor de las líneas de serigrafía (*silk*) presentes en el circuito es de  $6t$ (0.15mm).

### 3.3.2. Rutado y placement de los componentes

Una vez tenidos presentes los aspectos de diseño de la subsección anterior, pudimos comenzar a elaborar nuestra PCB. Para ello, las directrices que seguimos fueron las siguientes:

- Fijamos un espesor para las pistas de cobre encargadas de llevar la alimentación a los diferentes circuitos integrados de exactamente el doble del utilizado para el resto de señales. En concreto, establecimos un ancho de  $20t$ (0.5mm) para las alimentaciones y de  $10t$ (0.25mm) para el resto de pistas. Con esto conseguimos reducir la impedancia de aquellas líneas que transportan más corriente evitando así caídas de tensión y calentamiento.
- Fijamos un diámetro interior de las vías de  $15t$  y exterior de  $30t$  para favorecer el intercambio de las señales entre las diferentes caras de la placa.
- Las pistas nunca se trazan en el momento de un cambio de dirección formando ángulos rectos, sino que se emplean siempre ángulos obtusos. De esta forma se consigue tener una impedancia de línea lo más homogénea posible y menos sensible a posibles ruidos.

- Se ha dispuesto de dos planos de masa, uno por cada cara, para minimizar en la medida de lo posible la impedancia del camino de retorno de las señales. De esta forma, conseguiremos minimizar el posible acoplamiento por impedancia común que pudiese tener lugar. Además, al recorrer el camino de menor impedancia, las posibles señales de alta frecuencia describen superficies menores, con lo que evitamos posibles emisiones electromagnéticas con la formación de antenas parásitas de tipo dipolo magnético.

Por otra parte, empleando dos planos de masa en vez de uno, y repartiendo adecuadamente las pistas a lo largo de la PCB, conseguimos que en ningún momento se produzca una rotura del plano de masa que obligue a las corrientes de retorno a circular por caminos largos, con las consecuencias enunciadas anteriormente, así como a que todas ellas lo hagan por una zona muy estrecha, formándose antenas de parche parásitas.

- Los condensadores de desacoplo se colocarán lo más cerca posible de los pines correspondientes a los que deban conectarse, y se unirán a masa a través de una vía. Con esto conseguimos que los posibles pulsos de corriente que se generen retornen a masa recorriendo un camino muy corto y de baja impedancia.
- Todas las conexiones a masa de los diferentes componentes se realizarán a través de contactos de tipo *relief*. Con esto conseguiremos mantener el punto de soldadura a una elevada temperatura, puesto que el calor se concentra exclusivamente en el pad y no se calienta todo el plano de masa. De esta forma, el soldado de los diferentes elementos a la placa se consigue de una manera mucho más fácil.
- El módulo de radio se situará en un borde de la placa y lo más alejado posible del resto de elementos del circuito. Con esto disminuirémos las posibles interferencias que puedan aparecer debido a las corrientes sobre la placa. De igual forma, se dispondrá de numerosas vías que conecten los dos planos de masa en las inmediaciones al módulo de radio, así como la ausencia de los mismos justo en la zona donde se encuentra la antena, para mejorar en la medida de lo posible la recepción de señales via radio.
- Las líneas de conexión de equipos externos a la placa se han situado en la parte superior de la misma, y de nuevo alejadas lo suficiente del resto de dispositivos para que los posibles equipos de alta potencia conectados directa o indirectamente a ellas no ocasionen interferencias electromagnéticas.

- El rutado se comenzará por las líneas más críticas, i.e. las líneas de comunicación entre el USB y la UART, y las que conectan el módulo SPI del microcontrolador con la radio, puesto que se tratan de las de más alta velocidad.
- Se situarán 4 agujeros en cada una de las esquinas de las placas para favorecer su fijación en una posible caja contenedora.
- Se dotará de serigrafía suficiente a la placa para poder proceder posteriormente a la colocación de los componentes sobre la misma con facilidad.

Siguiendo pues estos criterios, se han elaborado los *layouts* de ambas caras de la placa que pueden consultarse en el apéndice C.

### 3.3.3. Pedido al fabricante

Finalmente, y para concluir con este capítulo, voy a comentar de forma muy breve el pedido concreto realizado a Itead Studio, i.e. las características de las placas que se han mandado fabricar. Éstas son:

- La anchura de la PCB se ha solicitado de 1.2mm por ser esta la anchura estándar de la mayor parte de las placas disponibles en el mercado.
- Se ha escogido como acabado superficial uno de tipo HASL (*Hot Air Solder Levelling*) que consiste en sumergir la PCB en un baño de soldadura de manera que todas las partes expuestas de la misma se cubren de cobre. Posteriormente, el exceso de cobre es eliminado sometiendo a la placa a un chorro de aire caliente a presión.
- La espesura de la capa de cobre se ha escogido de 1 onza ( $35\mu\text{m}$ ) al ser de nuevo este espesor un estándar de la industria.

Finalmente, una vez estos aspectos fueron seleccionados, se adjuntaron los ficheros Gerber correspondientes a cada una de las capas que pueden observarse en el apéndice C y que contienen toda la información necesaria para poder fabricar la PCB. Básicamente describen el conjunto de movimientos que la máquina de control numérico que se encarga de trazar las pistas y colocar los componentes debe dar para poder realizar el diseño pedido. El estándar común hoy en día es el conocido como RS-274X.

Con todo esto preparado, se tramitó el pedido que en poco menos de 3 semanas fue recibido.

# Capítulo 4

## Diseño software del proyecto

*If debugging is the process of removing bugs,  
then programming must be the process of putting them in.  
~ Edsger W. Dijkstra ~*

En el presente capítulo se va a detallar los pasos seguidos en la elaboración del *software* necesario para la comunicación tanto entre los módulos de radio como entre estos últimos y el PC. Procederemos en primer lugar a detallar la problemática del protocolo necesario para la comunicación fiable entre los módulos de radio, así como los formatos de tramas empleados. A continuación atacaremos el problema de otorgar a cada módulo de radio una dirección única que les permita ser direccionados sin ambigüedades, además de la posibilidad de implementar algún modo de bajo consumo para aumentar la vida útil de las baterías a las que podrán ir conectados. Finalmente detallaremos la aplicación para PC desarrollada para establecer las comunicaciones.

### 4.1. Protocolo empleado en la comunicación

Regresando de nuevo al problema que pretendemos resolver de la comunicación via radio, es necesario retomar la topología de red contemplada en la figura 1.1. Así, se pretende comunicar un módulo de radio conectado a un PC, que llamaremos de aquí en adelante módulo maestro, y el cual recibirá las peticiones del usuario a través de una aplicación, con diferentes módulos receptores (módulos esclavo) que se encontraran situados en diferentes lugares y controlarán los dispositivos conectados a sus entradas.

El empleo de los nombres maestro y esclavo no es indiferente. Así, hablamos de un único módulo maestro porque dada la topología de red disponible, solamente existe un único módulo de radio conectado al PC, y por tanto, ha de ser el encargado de iniciar todas y cada una de las comunicaciones con el resto de módulos, que únicamente reciben órdenes

y las acatan, por lo que hablamos de múltiples esclavos.

Sin embargo, para poder acometer con éxito las comunicaciones, no basta sólo con establecer quién ordena y quién recibe las peticiones, también es necesario establecer un protocolo que permita distinguir inequívocamente a cada módulo presente en la red así como garantizar la fiabilidad de las comunicaciones.

En este sentido, a la hora de escoger el protocolo que mejor se adapta a nuestras necesidades conviene hacer un listado de las mismas. Así:

- Toda orden emitida por el maestro ha de ser asentida por el receptor, i.e. nuestra comunicación ha de ser fiable puesto que hemos de estar seguros que las órdenes son acatadas.
- La petición de una orden únicamente tiene lugar entre el módulo maestro y uno de los módulos esclavo, i.e. solamente intervienen dos entidades, transmisor y receptor, en cada comunicación.
- La comunicación es por tanto *half-dúplex*, i.e. un módulo esclavo solamente envía una trama como respuesta a una orden.
- Debido al tipo de órdenes que se podrán enviar (siguiente subsección) únicamente será necesario el envío de una trama por cada orden, i.e. no será necesario segmentar la respuesta en varias tramas.
- Cuando el módulo maestro envía una orden ha de esperar al asentimiento del módulo esclavo antes de proceder a enviar otra orden. Se ha decidido implementar esta característica porque debido al tipo de aplicación que se pretende diseñar, no se contempla la posibilidad de un uso intensivo de las comunicaciones.
- Ante la posibilidad de errores en las comunicaciones, ya sea por la recepción de tramas corruptas o porque las tramas no lleguen a su destino, la lógica de control de los mismos será llevada a cabo por el módulo maestro, liberando así de carga a los módulos receptores.

Vistas pues nuestras necesidades, el protocolo que cumple con todas y cada una de las características expuestas anteriormente sería uno de tipo ARQ (*Automatic Repeat reQuest*, Solicitud de repetición automática) de parada y espera. El modo en que este protocolo permite resolver los problemas anteriormente mencionados de tramas perdidas y corruptas se puede observar en la figura 4.1. Así, se pueden dar cuatro posibles casos:

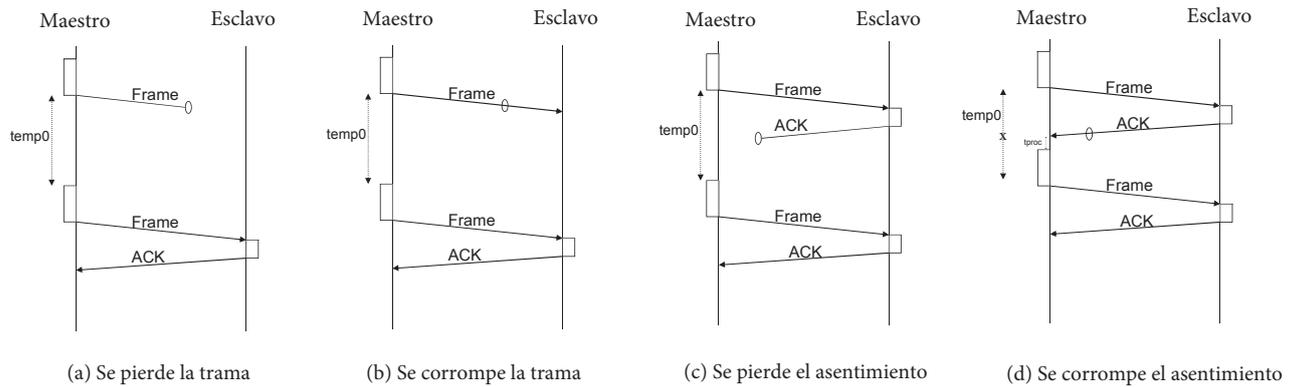


Figura 4.1: Solución de errores en ARQ de parada y espera

- Si se pierde la trama con la orden del módulo maestro, bien por atenuación o por interferencia con otras señales, se volverá a reenviar la trama tras vencer un temporizador asociado en el maestro. En nuestra aplicación dicho temporizador ha sido fijado a  $30 \mu s$ .
- Si se corrompe la trama con la orden del módulo maestro debido a interferencias con otras señales, el módulo receptor descartará la trama recibida y no hará nada más. Tras vencer el temporizador asociado en el maestro se reintentará la comunicación de nuevo.
- Si se pierde la trama con el asentimiento del módulo receptor, bien por atenuación o por interferencia con otras señales, se volverá a reenviar la trama tras vencer un temporizador asociado en el maestro.
- Si se corrompe la trama con el asentimiento del módulo receptor debido a interferencias con otras señales, el módulo maestro descartará la trama recibida y reenviará de nuevo la trama inmediatamente.

Finalmente, cabe decir que en caso de que se produzcan muchas retransmisiones debido a interferencias con otras señales o a que los módulos se encuentren a mucha distancia entre sí, este protocolo se quedaría bloqueado. Así pues, para evitarlo se ha fijado un máximo de 10 retransmisiones consecutivas en el envío de una trama que en caso de producirse hace que el módulo maestro transmita al PC un mensaje de módulo esclavo inalcanzable. La implementación de este protocolo en el módulo maestro se puede contemplar en el apéndice D.

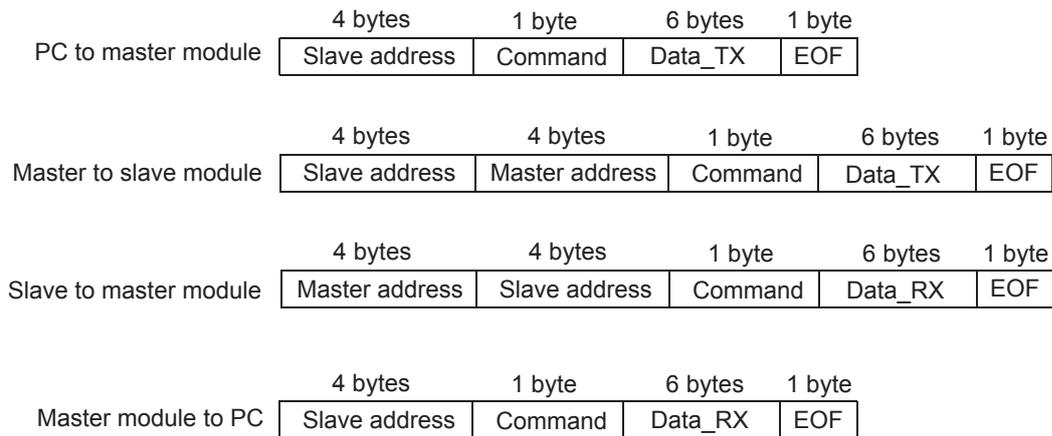


Figura 4.2: Formato de las tramas intercambiadas

#### 4.1.1. Formatos de trama empleados

Una vez comprendido el protocolo empleado en la comunicación, vamos a continuación a detallar el formato de las tramas intercambiadas entre los módulos, y entre éstos y el PC. Todo el conjunto de tramas se puede observar en la figura 4.2. Comenzando por la trama que la aplicación de PC pasa al módulo maestro, ésta comienza con la dirección del esclavo al que se pretende enviar la orden. Al constar esta dirección de 4 *bytes* esto nos permite direccionar hasta 4.294.967.296 módulos esclavos, número muy superior al necesario en la realidad. A continuación se transmite un campo de 1 *byte* que indica la orden concreta que se va a solicitar al esclavo. Hasta 256 órdenes diferentes se podrían llegar a procesar. Después viene un campo de datos de 6 *bytes* que se emplea como complemento de la orden solicitada en caso de que sea necesario. Finalmente se envía un *byte* que indica el final de la trama.

La trama que es enviada desde el módulo maestro hacia el PC es idéntica a la ya comentada, salvo que en el campo de datos se encuentra la respuesta enviada desde el módulo esclavo.

En estas dos tramas intercambiadas entre el PC y el módulo maestro no se ha implementado ningún tipo de CRC o de *checksum*, ni de marcaje de las mismas, puesto que la comunicación entre ambos tiene lugar a través del protocolo USB desde el puerto correspondiente en el ordenador hasta el convertidor FTDI USB a serie en el módulo maestro, y ya dicho protocolo se encarga de incorporar los mecanismos de corrección de errores correspondientes.

Por otra parte, en la comunicación que tiene lugar entre los módulos de radio el for-

<b>Command</b>	<b>Data_TX</b>	<b>Slave address</b>	<b>Data_RX</b>
<b>READPINS</b>	No use	Unicast	State of pins
<b>SETPIN</b>	Pin to set	Unicast	Pin set
<b>RESETPIN</b>	Pin to reset	Unicast	Pin reset
<b>BROADCAST</b>	No use	Broadcast	Slave address
<b>RECOVERCONFIG</b>	State of pins	Unicast	Config. recovered
<b>TEMPERATURE_SENSOR</b>	No use	Unicast	Temperature

Tabla 4.1: Comandos implementados en la aplicación

mato de trama varía ligeramente con respecto al anterior. Así, en la trama enviada desde el módulo maestro hacia el módulo esclavo se envía al igual que antes la dirección del esclavo en primer lugar; sin embargo, antes del campo de comando se envía la dirección propia del módulo maestro. Esto es así, para que el módulo receptor, una vez reciba la trama sepa después a quién debe dirigir la respuesta.

La trama de respuesta intercambiada entre el módulo receptor y el maestro es similar a la anterior pero con los campos de direcciones intercambiados. Esto es así, porque como ya se comentó en el capítulo donde se detallaba el integrado CC2500, el módulo de radio filtra los mensajes en función de la comparación entre el primer *byte* recibido de la trama y el *byte* más significativo de la dirección dada al módulo. Por tanto para que cada módulo reciba los mensajes dirigidos a ellos es necesario que el primer *byte* de la trama corresponda al *byte* MSB del receptor de la misma.

Finalmente, al igual que ocurría en las tramas intercambiadas entre el PC y el módulo maestro, no es necesario incluir aquí ningún tipo de método de comprobación de errores puesto que dichas tramas van encapsuladas dentro del campo de datos del paquete final enviado (figura 2.5).

Para concluir con esta sección, en la tabla 4.1 se recogen las órdenes que han sido implementadas en el proyecto desarrollado.

## 4.2. Implementación de la lógica del esclavo

Recapitulando un momento, hasta ahora hemos elaborado un módulo maestro encargado de la comunicación con el esclavo y de implementar la lógica del protocolo descrito anteriormente para sobreponernos ante posibles errores. Del módulo esclavo solamente se ha comentado que responde ante las peticiones del maestro y nada más. Sin embargo, existen otros aspectos que atañen a los módulos esclavos y que vamos a analizar en esta

sección. Así, son dos los principales problemas a los que nos enfrentamos, a saber:

- ¿Cómo podemos otorgar direcciones de manera unívoca a los diferentes módulos de una manera sencilla? Y una vez solucionada esta cuestión, ¿cómo podemos hacer para solicitar la dirección de un módulo que desconocemos?
- ¿Cómo podemos conseguir reducir al máximo el consumo de los módulos con el objetivo de alargar la vida útil de las baterías de las que se alimentan?

En las siguientes subsecciones analizaremos estas cuestiones.

### 4.2.1. El problema de la dirección

A la hora de otorgar direcciones a los módulos esclavos existen diferentes posibilidades. Una de ellas, la primera que se nos puede venir a la mente, consiste en introducir dichas direcciones de manera manual en los módulos a la hora de programarlos. Este método, si bien puede resultar sencillo, solamente es válido cuando el número de módulos es relativamente pequeño. En caso contrario, el tiempo dedicado a cambiar de manera manual todas y cada una de las direcciones de los diferentes módulos podría llegar a ser elevado, e incluso existiría el riesgo de repetir alguna de las direcciones.

De este modo, y aunque para nuestro proyecto no se contempla un número elevado de módulos receptores, se va a proceder a otorgar las direcciones de otra forma. Para ello, nos podríamos plantear la siguiente pregunta: ¿existe algún evento que cambie constantemente dentro del ordenador, de manera que podamos usar dicha situación para otorgar las direcciones? La respuesta es que sí, y el evento es el tiempo. Así pues, emplearemos el tiempo como base para elaborar nuestras direcciones.

En este sentido, la función *time* de la librería de C *time.h* nos permite obtener el tiempo en segundos transcurridos desde el inicio de la época UNIX (medianoche del 1 de enero de 1970). Se trata simplemente de un temporizador de 32 bits que se incrementa en una unidad cada vez que transcurre 1 segundo. Si empleásemos estos 32 bits (4 *bytes*) como direcciones de los módulos esclavo podríamos otorgarles direcciones diferentes cada segundo transcurrido durante aproximadamente 136 años. Esta es la filosofía implementada en el proyecto.

Sin embargo, para poder llevar a cabo esta tarea de manera automática, es necesario conocer la ubicación (dirección de memoria) exacta de la variable de nuestro programa que contiene la dirección del módulo, la cual puede cambiar de posición cada vez que se compila el código. Por tanto, para evitar que esto ocurra es necesario servirnos de las herramientas que nos proporciona el compilador cruzado empleado en el desarrollo del

proyecto (YAGARTO). Así, y como se puede observar en el código del módulo esclavo (apéndice E), incluyendo la primitiva `__attribute__((section(".vectors")))` conseguimos que la dirección del módulo permanezca en una posición fija tras la tabla de vectores de interrupción. Esto es así, porque esta directiva del compilador permite fijar las posiciones de memoria que ocuparán determinados segmentos de código junto con la información proporcionada en el *linker*.

Tras realizar este último paso, solamente queda comprobar la dirección de memoria donde se ha ubicado la variable que almacenará la dirección en el fichero *codierom.bin* y modificarla de acuerdo al valor leído de tiempo en el PC .

El código del programa que se encarga de esta última tarea se puede observar en el listado 4.1, y para lograr su ejecución siempre que se programe un nuevo módulo, se ha incluido dicho programa en la orden correspondiente del fichero *Makefile* antes de invocar al *flasher*.

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <time.h>
4 #include <stdlib.h>
5 #include <string.h>
6 #include <errno.h>
7
8 void DecToHexStr(int dec, char *str)
9 {
10     sprintf(str, "%X", dec);
11 }
12
13 void AsciiToInt(unsigned char *cadena)
14 {
15     while(*cadena)
16     {
17         if((*cadena)>=48) && ((*cadena)<=57)
18             *cadena = (*cadena)-48;
19         else
20             *cadena = (*cadena)-55;
21         cadena++;
22     }
23 }
24
25 int main()
26 {
27     typedef unsigned int    uint32;
28     typedef unsigned char   uint8;
29
30     uint32 fd,size,size2;
31     uint8 buffer[32768]={0};           //24KB of flash + 8KB of RAM (
        LPC11E13FBD48/301)
```

```

32
33     uint8 sample[8]={0};                //Store a provisional value
34     uint8 final_address[4]={0};        //Store the address given to
        the device
35     uint32 seconds;                    //Store time without lost
36
37     seconds = time(NULL);              //Time in seconds from January
        1, 1970
38
39     //printf("Seconds since January 1, 1970 = %d\n", seconds);
40
41     DecToHexStr(seconds, sample);      //Convert seconds to
        hexadecimal string and store in sample
42
43     //printf ("Address Device: %s\n",sample);
44
45     AsciiToInt (sample);              //Convert seconds in
        hexadecimal string to seconds in hexadecimal numbers
46
47     //Interchange positions to see MSB changes everytime (CC2500 filter
        addresses due to MSB byte)
48
49     final_address[3] = sample[0]*16 + sample[1];
50     final_address[2] = sample[2]*16 + sample[3];
51     final_address[1] = sample[4]*16 + sample[5];
52     final_address[0] = sample[6]*16 + sample[7];
53
54     printf ("\nAddress Device: %02X%02X%02X%02X \n\n",final_address[0],
        final_address[1],final_address[2],final_address[3]);
55
56
57     //////////// FILE HANDLE ////////////
58
59     fd = open("coderom.bin",O_RDONLY); //Open file to read
60
61     if(fd == -1)
62     {
63         printf("Error opening file\n");
64         printf("%s\n", strerror(errno));
65         return 0;
66     }
67
68     size = read(fd, buffer, sizeof(buffer)); //Read file
69
70     if(size == -1)
71     {
72         printf("Error reading file\n");
73         printf("%s\n", strerror(errno));
74         return 0;
75     }
76
77     close(fd);                        //Close file
78

```

```
79     if(fd == -1)
80     {
81         printf("Error closing file\n");
82         printf("%s\n", strerror(errno));
83         return 0;
84     }
85
86     buffer[0xC9] = final_address[0];           //MSB byte ADDR Register in
87         CC2500
88
89     //Vector with address
90
91     buffer[0xEF] = final_address[0];
92     buffer[0xF0] = final_address[1];
93     buffer[0xF1] = final_address[2];
94     buffer[0xF2] = final_address[3];
95
96     fd = open("coderom.bin", O_WRONLY);       //Open file to write
97
98     if(fd == -1)
99     {
100         printf("Error opening file\n");
101         printf("%s\n", strerror(errno));
102         return 0;
103     }
104
105     size2 = write(fd, buffer, size);          //Write file
106
107     if(size2 == -1)
108     {
109         printf("Error writing file\n");
110         printf("%s\n", strerror(errno));
111         return 0;
112     }
113
114     close(fd);                                //Close file
115
116     if(fd == -1)
117     {
118         printf("Error closing file\n");
119         printf("%s\n", strerror(errno));
120         return 0;
121     }
122
123     return 0;
124 }
```

Código 4.1: Programa encargado de proporcionar la dirección a un módulo

Una vez que dispongamos de nuestros módulos adecuadamente programados y dotados cada uno de ellos de una dirección diferente, es el momento de solventar el otro problema que presentamos anteriormente. Así, tenemos que encontrar la manera de que

el módulo maestro, encargado de las comunicaciones, pueda obtener las direcciones de dichos esclavos para poder transmitirles las órdenes pertinentes.

Atendiendo a lo expuesto en el Capítulo 2, el transceptor de radio CC2500 solamente atiende a los mensajes que comienzan con la dirección propia interna con la que ha sido configurado, y también a aquellos que empiezan con la dirección de *broadcast* 0x00. Por este motivo, para poder conocer la dirección de un módulo será necesario preguntarle la misma con un mensaje enviado por difusión.

Sin embargo, si se realizase esto sin ningún otro tipo de control, todos los módulos esclavos presentes en el radio de alcance del transmisor responderán a dicho mensaje a la vez, con las consiguientes interferencias. Por todo ello, se ha implementado un mecanismo de control que consiste en que solamente el esclavo que haya sido seleccionado previamente al envío del mensaje de difusión será el que responda a dicho mensaje indicando su dirección. Esta selección ha sido implementada mediante la pulsación de un botón presente en la PCB del módulo esclavo, que lo que permite es que durante los siguientes 15 segundos a la pulsación el módulo quedará habilitado para recibir órdenes preguntando por su dirección. Pasado ese tiempo, volverá al estado de no respuesta.

Por tanto, a la vista de la solución presentada, el modo de proceder para poder conocer la dirección de todos los módulos consistirá en ir seleccionándolos de uno en uno, y enviar una petición de difusión preguntando por su dirección en un tiempo no superior a 15 segundos.

El código correspondiente a dicha lógica de selección se puede consultar en el apéndice E.

#### 4.2.2. El problema del consumo

Como ya se hubo contemplado en el diseño *hardware* de los módulos de radio en el Capítulo 3, es posible conectar dichos módulos a una batería para que funcionen de manera autónoma, lo cual es bastante útil si pensamos que en un entorno industrial no siempre será accesible una fuente de suministro de potencia que alimente nuestro módulo receptor.

Así pues, si tenemos en cuenta esta posibilidad, una premisa de este tipo de sistemas es la reducción del consumo para permitir así una mayor autonomía de los mismos.

En el caso que nos compete, si nuestros módulos de radio permaneciesen en estado de recepción todo el tiempo, tendrían un consumo de entre 15.7mA y 18.8mA estando, como han sido programados, optimizados para sensibilidad. Si el consumo de nuestro módulo

se debiera exclusivamente al módulo de radio en estado de recepción (que no es así), y suponiendo una batería típica de 1800mAh de capacidad, esto supone una vida útil de unos 4 días y 8 horas. Si bien esta cifra pudiera parecer aceptable, lo cierto es que es posible aumentarla si empleamos algún tipo de modo de bajo consumo.

Así pues, en un primer momento se pensó en una utilidad opcional que incluye los integrados CC2500 y que parecía idónea para nuestro propósito denominada *WOR*, *Wake On Radio*. Consiste básicamente en despertar periódicamente al integrado de su estado *Sleep* y escuchar el medio por si llegan tramas, todo ello sin necesidad de interacción con el microcontrolador, sino a través de la configuración de una serie de registros internos. Sin embargo, al consultar la documentación relativa a esta funcionalidad, nos encontramos con que la nota de errores del integrado [2] detallaba hasta 3 errores, alguno de ellos de silicio, que le afectaban, siendo las soluciones propuestas alternativas (*workaround*) un tanto complejas y difíciles de implementar. Debido a este percance, se decidió implementar un sistema similar al que presentaba esta funcionalidad pero gestionado desde el microcontrolador.

En la figura 4.3 se pueden observar los pasos de cómo se ha conseguido implementar. Básicamente, lo que se lleva a cabo, con la ayuda de un temporizador es la conmutación periódica del integrado CC2500 entre los estados *Sleep* y Recepción. Se ha decidido que tras 300ms de inactividad, la radio pase a escuchar el medio durante 100ms. Esto garantiza que la radio se encuentre en modo recepción 2 veces por segundo, lo que será más que aceptable para la frecuencia de órdenes que pueda recibir y también para minimizar la posibilidad de que no se reciba un mensaje por encontrarse la radio inactiva.

Además de esto, al microcontrolador también se le ha dotado de un modo de bajo consumo que permite que la CPU deje de ejecutar instrucciones si bien los periféricos integrados siguen funcionando. Con esto conseguimos dormir también al microcontrolador cuando el transceptor CC2500 pase al modo *Sleep*, reduciendo así el consumo total de la PCB.

Retomando el ejemplo comentado anteriormente de que la radio se alimente desde una batería de 1800mAh, ahora el consumo que se tiene, de acuerdo a los periodos de actividad e inactividad comentados anteriormente, y suponiendo que no se transmite ningún mensaje es de 4.3mA, lo cual nos otorga una vida útil de unos 17 días y 9 horas, i.e. casi 4 veces más duración que la que se consigue sin emplear ningún tipo de modo de bajo consumo.

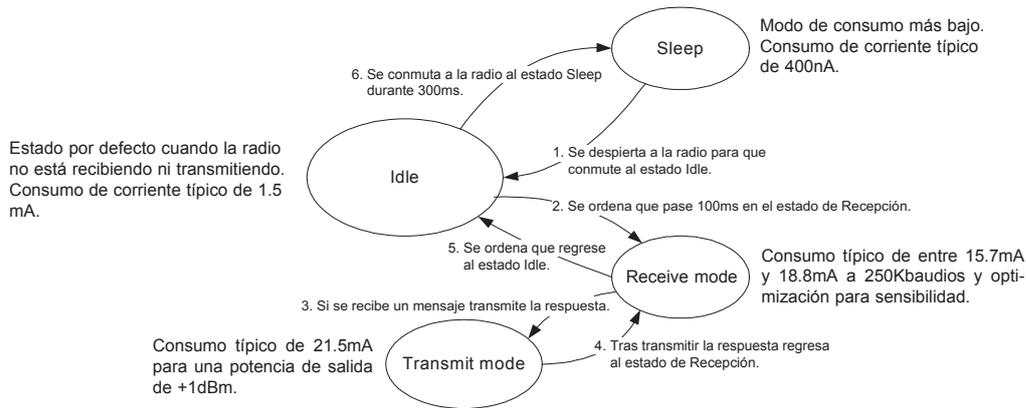


Figura 4.3: WOR gestionado por el microcontrolador

### 4.3. Programación de los módulos

Una vez que el *firmware* de los diferentes módulos ha sido elaborado, tan solo queda grabarlo en la memoria *flash* de los microcontroladores para que no se pierda su contenido cuando no se disponga de alimentación. Para ello nos hemos servido de la utilidad de libre distribución *lpc21isp* [16] que se trata de un programador ISP portable para las familias de microcontroladores LPC1000 y LPC2000 del fabricante NXP. Básicamente se encarga de indicar al *bootloader* del microcontrolador que a través del puerto serie se va a hacer llegar a la memoria *flash* el *firmware* elaborado.

No obstante, en el código fuente de dicho programa no se contempla la posibilidad de programar el modelo de microcontrolador elegido (LPC11E13FBD48/301), por lo que ha sido necesario añadir la siguiente línea de código en el fichero *lpcprog.c* para lograr su programación:

```

1 // id, name of product, flash size, ram size, total number of sector,
2 // max copy size, sector table, chip variant
3 { 0x296A102B, "11E13.../301", 24, 8, 6, 4096, SectorTable_17xx,
  CHIP_VARIANT_LPC11XX },

```

Código 4.2: Parámetros de configuración del microcontrolador LPC11E13FBD48/301

Dicha línea proporciona el número de identificación del modelo de microcontrolador empleado, el nombre del producto, los tamaños de las memorias *flash* y *RAM* en KB, el número de sectores en que se divide la memoria *flash* y el tamaño de los mismos en KB, así como la variante del *chip* a la que pertenece el microcontrolador.

El fichero *Makefile* desde el que se ordena la programación de los microcontroladores se lista a continuación. En él se puede observar como antes de la invocación del programador se llama al programa que se encarga de otorgar una dirección a los módulos.

```
1 TOOL = arm-none-eabi
2 CC   = $(TOOL)-gcc
3 CP   = $(TOOL)-objcopy
4 BT2  = ../bootloader/bt2
5 ISP  = ../lpc2lisp/lpc2lisp
6
7 CFLAGS = -w -Os -mcpu=cortex-m0 -mthumb -nostartfiles
8
9 def:   code.bin
10      $(BT2) -l code.bin
11
12 burn: coderom.bin
13      ./putaddress
14      $(ISP) -control -bin $< /dev/ttyUSB0 115200 12000
15
16 all:  code.bin coderom.bin term6b
17
18 code.elf: main.c init.c printf.c linker_scriptRAM.ld Makefile
19          $(CC) $(CFLAGS) -Wl,-Tlinker_scriptRAM.ld -Wl,-Map=a.map -o $@ init.c
20          main.c
21          $(TOOL)-size $@
22
23 coderom.elf: main.c init.c printf.c linker_script.ld Makefile
24             $(CC) $(CFLAGS) -Wl,-Tlinker_script.ld -Wl,-Map=a.map -o $@ init.c main
25             .c
26             $(TOOL)-size $@
27
28 code.bin: code.elf
29           $(CP) -O binary $< $@
30
31 coderom.bin: coderom.elf
32             $(CP) -O binary $< $@
33
34 term6b: term6b.c
35         gcc -w -O2 -o $@ $<
```

Código 4.3: Fichero Makefile empleado en la programación

## 4.4. Programa para PC

Para concluir con el presente capítulo dedicado a la implementación *software* del proyecto voy a tratar el último aspecto que nos queda y que consiste en la realización de la aplicación para PC. Así, una vez que nuestros módulos maestro y esclavos han sido

convenientemente programados, es el momento de realizar una interfaz de usuario que permita a cualquier operador comunicarse con el módulo maestro para que éste pueda solicitar las órdenes pertinentes a los diferentes esclavos.

Dado que se prevee que los posibles usuarios potenciales de la aplicación no tengan un conocimiento profundo en el manejo de este tipo de *software*, se ha optado por elaborar una GUI que responda a tres claras premisas, a saber:

1. Portable
2. Sencilla
3. Intuitiva

Para poder lograr estos objetivos, y tras una búsqueda de posibles IDEs, se decidió que la mejor manera de implementar dicha interfaz fuese con la herramienta *wxDev-C++*, detallada en la siguiente subsección.

#### 4.4.1. wxDev-C++

*wxDev-C++* [17] es un entorno integrado de desarrollo basado en el popular *Dev-C++*. Así, constituye un entorno ideal para la elaboración de proyectos al contar entre otras muchas características con la capacidad de crear árboles de directorios, creación de esqueletos de funciones, y muchos otros aspectos que resultan de gran ayuda a la hora de programar aplicaciones tanto en lenguaje C como C++. En la imagen 4.4 se puede observar una captura de dicho IDE.

Sin embargo, la característica más importante de este IDE, y que lo distingue de otros, es la incorporación de las API *wxWidgets*. Éstas son unas bibliotecas multiplataforma y libres para el desarrollo de interfaces gráficas programadas en lenguaje C++. Además, al estar basadas en las bibliotecas ya existentes en el sistema (nativas), se integran de forma óptima y resultan muy portables entre distintos sistemas operativos. En el caso del proyecto realizado, la aplicación ha sido implementada bajo un entorno *Windows*, pero gracias al empleo del compilador *MinGW* la aplicación en su totalidad es prácticamente portable entre sistemas operativos *Windows* y *Linux* con apenas ligeros cambios.

La manera de generar la interfaz gráfica es muy sencilla. Tan solo hay que ir seleccionando el elemento que necesitemos en cada instante, e.g. una caja de texto, un menú, una caja de selección, etc. y colocarla en la zona de la ventana que creamos oportuna. Este paradigma de arrastrar y soltar resulta francamente sencillo y permite con muy poco esfuerzo conseguir una interfaz gráfica en relativamente poco tiempo.

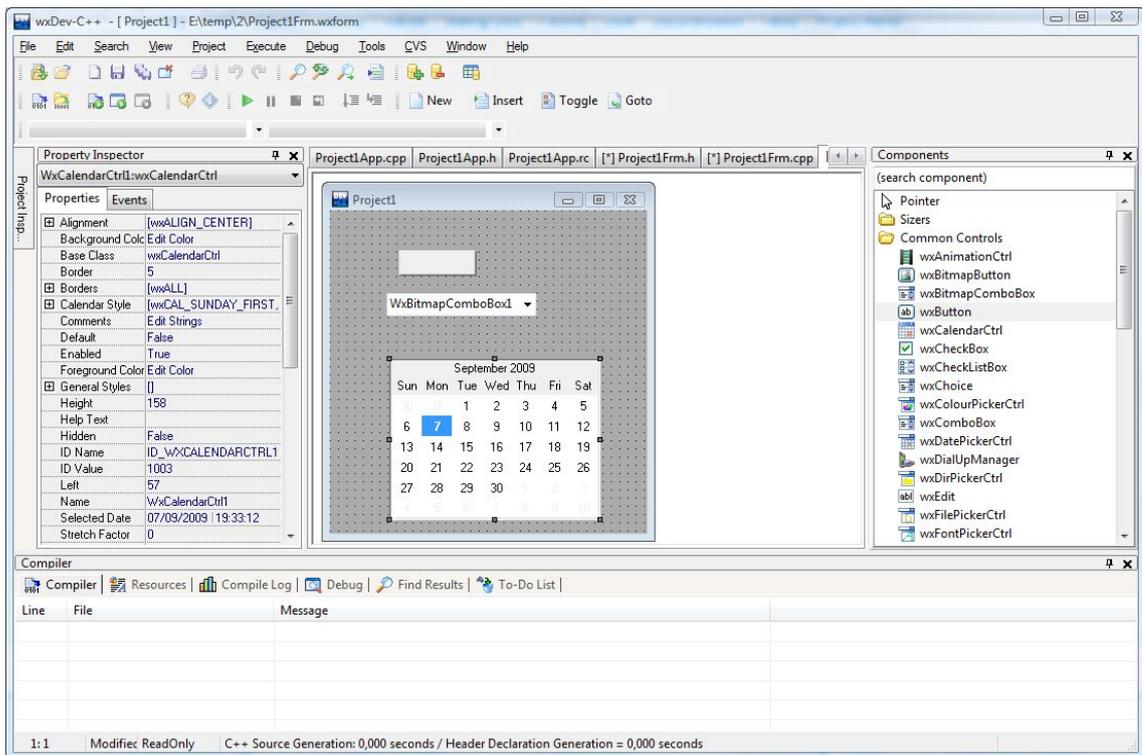


Figura 4.4: Captura de pantalla del IDE wxDev-C++

Por otra parte, la programación de la aplicación en sí, i.e. que se tiene que realizar cuando se realiza una acción (seleccionar, escoger, hacer *click*) sobre cada uno de los elementos colocados sobre la ventana de trabajo, resulta también sencilla. Así, lo que se emplea es un sistema orientado a eventos que permite que cada vez que se realice una acción sobre uno de los elementos se 'dispare' el procedimiento asociado al mismo. De este modo no existe un 'main' como tal, sino que existen un conjunto de funciones independientes que son ejecutadas cada vez que se interactúa de algún modo sobre la interfaz gráfica.

Siguiendo pues todas estas indicaciones, se ha elaborado para nuestra aplicación la interfaz gráfica que puede observarse en la imagen 4.5. El código generado para obtener la funcionalidad deseada se puede contemplar en el anexo F, mientras que el manual de instrucciones de la misma se analiza en la siguiente subsección.

#### 4.4.2. Manual de usuario

En la presente subsección se pretende explicar cómo se ha de manejar la aplicación para PC con el objetivo de lograr comunicarnos con el módulo de radio maestro.

En primer lugar, podemos dividir la interfaz gráfica diseñada en las tres partes que se

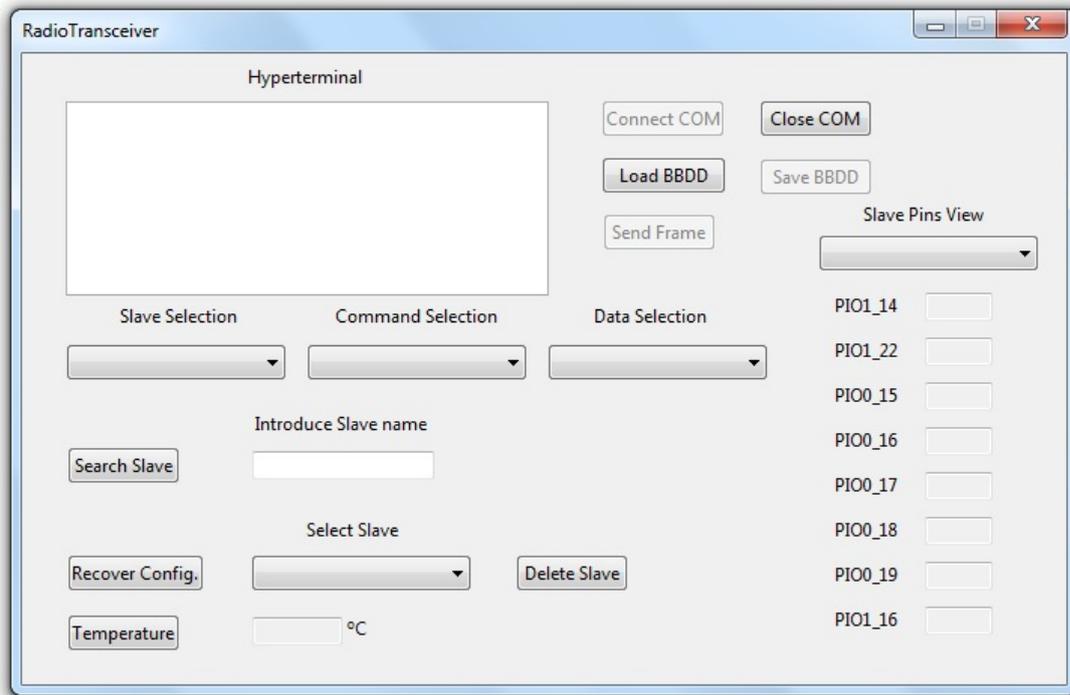


Figura 4.5: Captura de pantalla de la interfaz gráfica desarrollada

pueden apreciar en la imagen 4.6. Éstas son:

- Zona de depuración: En este cuadro de diálogo se pueden observar todos los mensajes de depuración generados por la aplicación, así como las respuestas procedentes del módulo de radio maestro a cada una de las órdenes emitidas. Supondrá una herramienta muy útil para comprobar posibles errores en la comunicación.
- Comunicaciones con el puerto serie y gestión de BBDD: Esta zona permite iniciar y finalizar las comunicaciones con el puerto serie correspondiente al módulo de radio maestro, así como cargar y guardar la base de datos donde se aloja la información de todos y cada uno de los módulos de radio esclavos que se han vinculado.
- Administración y órdenes: Esta zona permite intercambiar todos los mensajes analizados en la sección primera de este capítulo, así como la posibilidad de eliminar de la base de datos todo aquel módulo esclavo que ya no se precise.

Para comenzar a usar la aplicación será necesario realizar tres pasos de manera consecutiva:

1. Conectar el módulo de radio maestro a uno de los puertos USB del ordenador.

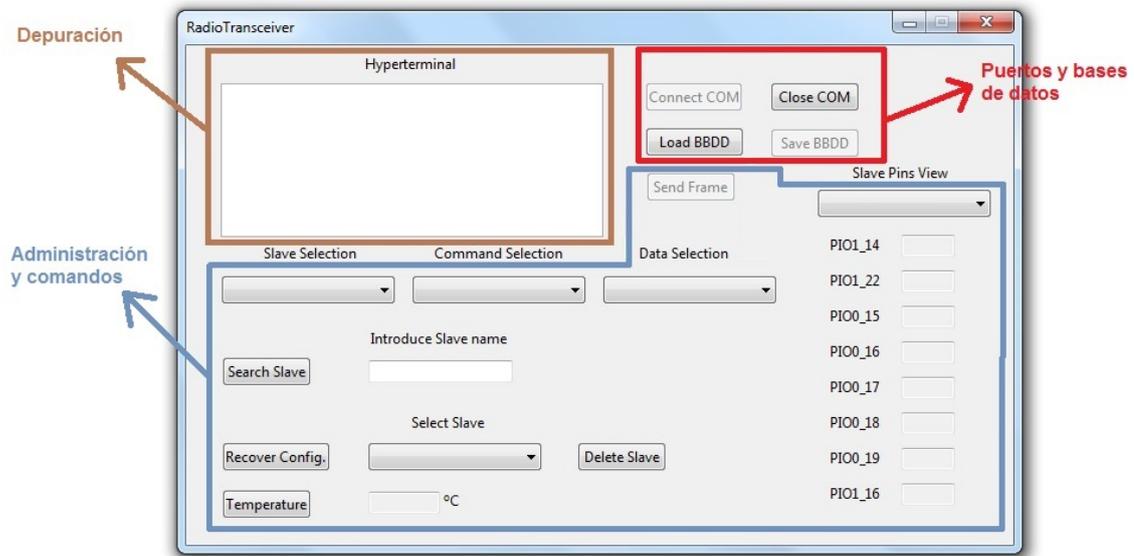


Figura 4.6: Partes de la interfaz gráfica desarrollada

2. Pulsar el botón *Connect COM* para localizar de manera automática dicho módulo de radio. Si la conexión con el mismo fue exitosa, el mensaje *Puerto serie encontrado* así como la velocidad de conexión aparecerán en el *hyperterminal*.
3. Pulsar el botón *Load BBDD* para cargar la configuración de todos los módulos esclavos almacenados y poder así comunicarnos con los mismos.

Si en alguno de los pasos anteriores ocurrió algún problema (no se detectó el módulo, etc.) la aplicación no nos permitirá realizar ninguna tarea más. Esto se podrá comprobar porque los botones correspondientes a las acciones que en un momento dado no se nos permiten realizar permanecen desactivados y no se puede interactuar con ellos.

Una vez la conexión con el módulo ya se haya realizado, el usuario puede proceder a realizar cualquiera de las acciones que han sido descritas en el comienzo de este capítulo. Así:

- Los *combobox* etiquetados como *Slave Selection*, *Command Selection* y *Data Selection* permiten seleccionar el nombre del esclavo, nombre de la orden y número de salida respectivamente. Las órdenes que se nos permite seleccionar son las de *SET PIN* y *RESET PIN*, mientras que el número de salidas disponibles es de 4. Una vez hayamos hecho una selección en todos y cada uno de estos *combobox*, el botón

*Send Frame* se activará y con una pulsación sobre el mismo enviaremos la orden solicitada.

- El cuadro de texto *Introduce Slave Name* junto con el botón *Search Slave* permiten realizar la orden *BROADCAST*. Así, tras introducir el nombre con el que deseamos que se registre el esclavo que vayamos a incorporar al sistema en dicho cuadro de texto, pulsando el botón descrito podremos iniciar la búsqueda de esclavos previamente seleccionados y en caso de que exista alguno incorporarlo al sistema.
- El *combobox Select Slave* nos permite seleccionar el esclavo presente en la base de datos sobre el que deseamos que se realice alguna de las siguientes acciones:
  - *TEMPERATURE SENSOR*: Tras la selección de un esclavo y la pulsación del botón *Temperature* el módulo procederá a solicitar la temperatura al esclavo correspondiente y la visualizará en el cuadro provisto para tal efecto.
  - *RECOVER CONFIG*: Con un esclavo seleccionado y tras pulsar el botón *Recover Config*, se solicitará al esclavo que recupere la configuración de sus pines de salida que están almacenado en la base de datos.
  - Tras la selección de un esclavo y la pulsación del botón *Delete Slave* se eliminará de la base de datos el esclavo correspondiente.
- El *combobox Slave Pins View* permite seleccionar el esclavo del que deseamos conocer el estado actual de sus pines tanto de salida como de entrada, i.e. ejecutar el comando *READ PINS*. Una vez recibido el mensaje, los ocho pines listados en la parte inferior derecha de la ventana mostrarán el valor *SET* en caso de que valgan '1' ó *RESET* en caso de que valgan '0'. De igual forma, cada vez que se ejecute una orden que altere el estado de alguno de estos pines, los valores de éstos también se actualizarán.

Finalmente, una vez hayamos terminado de realizar todas las acciones que consideremos oportunas, podremos abandonar la aplicación a través del símbolo 'X' de la ventana, si bien se pueden hacer dos últimas acciones, a saber:

1. Pulsando el botón *Save BBDD* podremos guardar el estado del sistema tal y como lo dejamos al abandonar la aplicación, por si en una posterior conexión deseamos continuar donde lo dejamos.
2. Pulsando el botón *Close COM* cerraremos la conexión con el módulo de radio maestro y podremos desconectar la placa de forma segura.

# Capítulo 5

## Resultados experimentales

*No amount of experimentation can ever prove me right;  
a single experiment can prove me wrong.  
~ Albert Einstein ~*

Una vez realizado tanto el diseño *hardware* como *software* del proyecto es el momento de proceder a montar la PCB y probar su correcto funcionamiento. De este modo, en este capítulo se abordarán aspectos tales como errores en el diseño *hardware*, pruebas experimentales de cobertura y alcance, y mediciones de consumo real de potencia.

### 5.1. Diseño realizado

El diseño completo realizado, con todos los componentes electrónicos ya soldados sobre la PCB, y por ambas caras se puede contemplar en las imágenes 5.1 y 5.2.

El proceso de soldadura de los diferentes componentes no ha supuesto ningún problema, si bien los integrados de pequeño tamaño son difíciles de manipular y la soldaduras de sus pines ha requerido de la ayuda de un microscopio. Por todo lo demás, lo único que ha consumido realmente tiempo ha sido la verificación, tras cada componente soldado, de su correcta conexión a las pistas de la PCB. De este modo, nos aseguramos que en caso de detectarse un problema tras el montaje, éste no es debido a una conexión flotante. Para dicha verificación nos hemos servido simplemente de un polímetro en modo cortocircuito. Sin embargo, a la vista de las anteriores imágenes, se pueden observar dos aspectos en los que sí se han encontrado problemas, a saber:

- El módulo de radio ANAREN ha sido conectado a la PCB a través de una serie de finos cables. El principal problema con el que contábamos desde el principio del diseño ha sido que dicho integrado solamente se comercializa con un encapsulado de tipo QFN (*Quad Flat No-leads*), i.e. con los pines sin sobresalir por fuera del



encapsulado. Si bien este tipo de encapsulados no suponen ningún problema para su soldadura industrial, lo cierto es que para la soldadura manual suponen un auténtico reto. Así, aunque en un principio se pensó que podía ser posible su soldadura si se lograba calentar los *pads* lo suficiente, lo cierto es que resultó casi imposible. Por este motivo, se decidió recurrir a la solución cableada que se observa en la imagen 5.1. Si bien no es una solución muy estética, si es plenamente funcional, pues apenas supone un incremento de impedancia debido a los cables.

- La necesidad de un par de cables para solucionar un problema de rutado. Una vez se hubo montado la placa en su totalidad, ésta se conectó a través de un cable USB a un PC para una primera prueba de conexionado. Se pudo comprobar como el ordenador reconocía el dispositivo FTDI, así como los LEDs de testigo de alimentación se iluminaban. Todo parecía ir bien, hasta que llegó el momento de la programación del microcontrolador. Así, cuando se ejecutaba el comando *make* no se establecía conexión alguna con el mismo. Pensando que se trataba de un problema de conexiones flotantes, se revisaron de nuevo todas y cada una de ellas entre el FTDI, que el ordenador reconocía, y el microcontrolador. Sin embargo, éstas eran correctas. Cambiando de posible causa del problema, se revisó el *datasheet* del microcontrolador por si la conexión con los pines de la UART no era la correcta. Y allí se encontró la solución al problema. Resultó que aun siendo correctos los pines del microcontrolador que empleaban la UART, ésta no era la que se empleaba por defecto en la programación ISP. De este modo, la solución encontrada fue rutar dos cables desde la salida del FTDI a los pines de la UART por defecto. Una vez hecho esto, la programación del microcontrolador pudo realizarse correctamente.

Un último aspecto a comentar aquí es que, aunque no se dispuso del diseño completo hasta un tiempo después de iniciado el proyecto, las pruebas experimentales del *software* que se fueron elaborando mientras, se pudieron realizar gracias a una solución temporal. Dicha solución consistió en el empleo de una *Education Board* basada en el microcontrolador LPC2103 del fabricante *Embedded Artists* [18]. De este modo, conectando el módulo de radio ANAREN al puerto de expansión de dicha placa, se pudieron hacer pruebas *software* de comunicación con el módulo de radio hasta que se dispuso del diseño propio. En la imagen 5.3 se puede observar este conexionado.

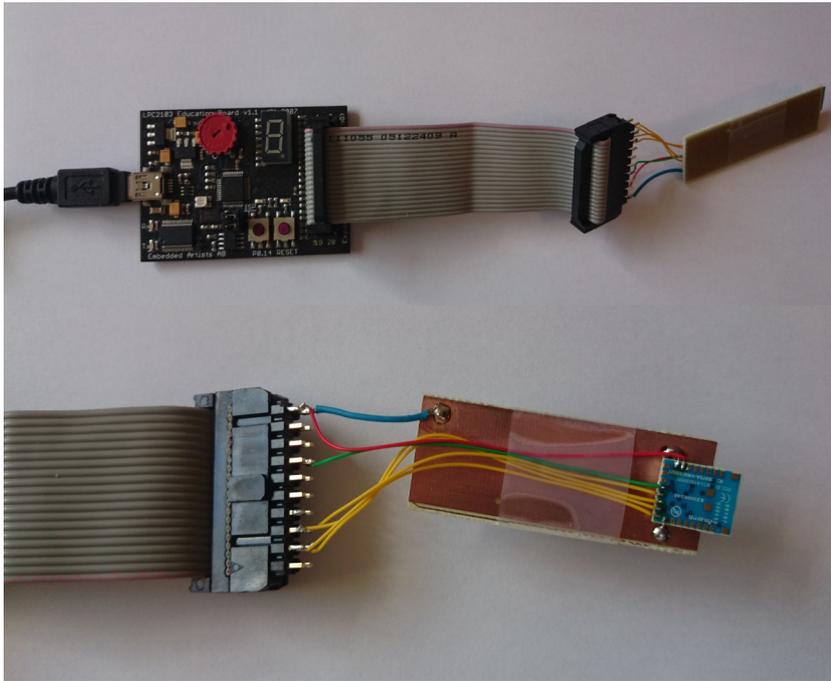


Figura 5.3: Módulo de radio ANAREN conectado a la placa *Education Board*

## 5.2. Consumo de potencia

En el momento en que se elaboraron los esquemáticos del proyecto (apéndice B) se realizó un cálculo aproximado del consumo de potencia estimado total que tendrían las PCB. No obstante, este tipo de cálculos suponen en la mayor parte de los casos una cota superior, ya que se elabora con los máximos valores de consumo de los dispositivos. Por este motivo, se hace necesario realizar la medición experimental del resultado exacto de consumo que tiene nuestro proyecto. Para ello, basta con realizar una medición del consumo de corriente de nuestra placa retirando el *jumper* que se encuentra en serie con la alimentación y colocando el multímetro en su lugar.

De este modo, podemos distinguir dos valores diferentes:

- En el módulo maestro, que no implementa el modo de bajo consumo analizado en el capítulo 4 ya que se alimenta siempre desde un PC, se obtuvo un consumo constante de 32.8mA (imagen 5.4). Este valor es bastante próximo al analizado teóricamente teniendo en cuenta que dicha medida fue tomada estando la radio en el estado de recepción y con todos los diodos LED apagados.
- En el módulo esclavo, al implementarse el modo de bajo consumo comentado anteriormente, se obtuvo una medida fluctuante, tal y como cabría esperar. Así, midien-

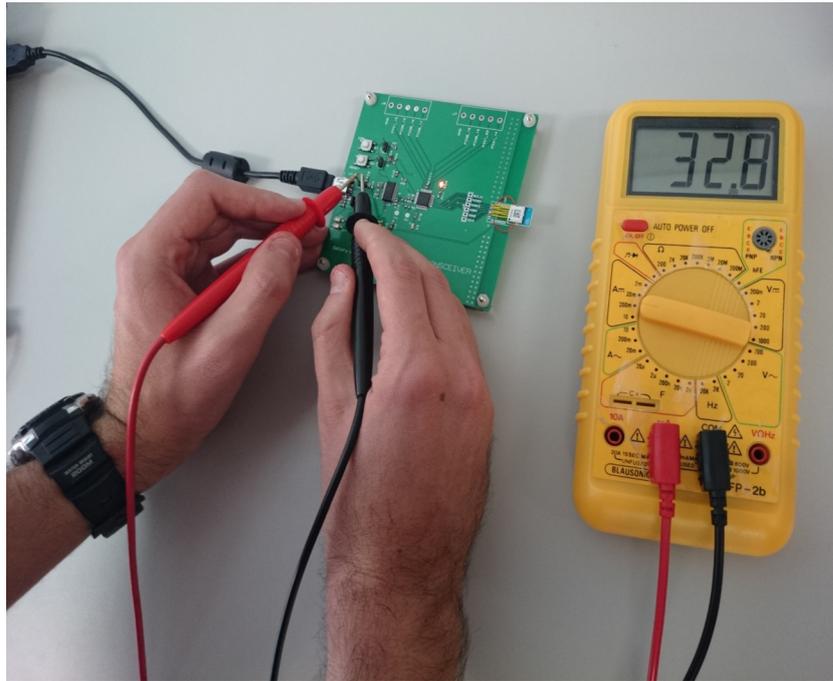


Figura 5.4: Consumo experimental del módulo de radio maestro

do dicho consumo en el estado de recepción y con el diodo testigo de la alimentación encendido se obtuvo un valor mínimo de 17.3mA y uno máximo de 41.3mA. Teniendo en cuenta que la primera medida se tiene 300ms de cada 400ms y que la segunda medida se tiene 100ms de cada 400ms, todo ello de acuerdo a lo explicado en la sección 4.2.2, se tiene un consumo promedio total de 23.3mA. Dicho consumo podría ser aún menor si alimentásemos el sistema desde una batería y no alimentásemos el integrado FTDI.

### 5.3. Rango de cobertura

En la presente sección se pretende demostrar de manera empírica el alcance máximo que posee nuestro sistema para el cual la comunicación obtenida es bastante fluida, i.e. el número de errores obtenido es aceptable y no supone una gran pérdida de tiempo. Las pruebas han sido realizadas en una sala sin ningún tipo de obstáculos entre los módulos emisor y receptor. La potencia de transmisión se ha fijado al máximo posible, i.e. +1dBm, que para una tasa de transmisión de 250kBaudios presenta una sensibilidad en recepción de -87dBm. Teniendo en cuenta estas premisas de partida, los resultados que se han obtenido se pueden observar en la tabla 5.1. A la vista de los resultados obtenidos podemos concluir que la mejor distancia para garantizar un buen funcionamiento de las

<b>Distancia medida</b>	<b>Potencia en recepción promedio</b>	<b>Calidad de la señal recibida</b>	<b>Número de retransmisiones promedio</b>
1,6 metros	-61dBm	Muy buena	< 5
3,2 metros	-65dBm	Buena	< 10
6,5 metros	-75dBm	Regular	10 - 40
8 metros	-83dBm	Mala	30 - 80
9,5 metros	-89dBm	Muy mala	> 100

Tabla 5.1: Cobertura y número de errores a +1dBm de potencia de tx y 250kBaudios para órdenes individuales

comunicaciones es en torno a los 6 metros.

Por otra parte, es necesario considerar que debido a la característica de bajo consumo con que se ha dotado al módulo esclavo, no todas las retransmisiones que figuran en el punto anterior son debidas a errores en el CRC o la atenuación de la señal, sino que algunas de ellas tienen lugar debido a que en ese momento el módulo puede encontrarse dormido y por tanto no responde a un mensaje procedente del transmisor. Sin embargo, teniendo en cuenta que el módulo esclavo permanece dormido una media de 250ms cada segundo, i.e. un cuarto de segundo, las retransmisiones debido a este aspecto no serán muy elevadas.

# Capítulo 6

## Presupuesto económico

*There are so many things more important than money,  
but they cost too much...  
~ Groucho Marx ~*

Tras la finalización del proyecto, y una vez se ha comprobado que se dispone de un prototipo plenamente funcional, es el momento de elaborar un presupuesto económico en el que se recojan todos los gastos en que se ha incurrido. En el presente capítulo se calculan dichos gastos y se muestran desglosados en función de su carácter.

### 6.1. Costes materiales

En primer lugar, en la tabla 6.1 se recoge el gasto que ha supuesto la adquisición de los diferentes componentes de que consta el proyecto realizado. Así, en ella se han incluido los precios de los elementos necesarios para la fabricación de dos módulos de radio, uno maestro y otro esclavo. El motivo por el cual para algunos elementos el número de unidades presupuestadas es superior a la necesaria se debe a que el proveedor de los mismos no suministra un pedido inferior a tal cantidad. De este modo se puede obtener así un precio final más ajustado al real.

Por otra parte, dentro de los gastos materiales debemos incluir también el coste asociado al envío y fabricación de las placas de circuito impreso, así como el gasto relativo al empleo de los materiales e instrumentación de la mesa de soldadura. En este último aspecto, cabría recoger el empleo de estaño y *flux*, poco importante, así como el consumo eléctrico de los soldadores y demás equipo electrónico (osciloscopio, multímetro) empleado en la verificación del montaje de las PCBs. La tabla 6.2 recoge estos gastos.

COSTES DE LOS COMPONENTES ELECTRÓNICOS						
Tipo	Nombre del Componente	Proveedor	Ref. Prov.	Coste Unitario	Cantidad	Coste Total
Regulador de tensión	TEXAS INSTRUMENTS TLV70033DDCT IC, LDO, 200MA, 3.3V, 5SOT23	Farnell	1778230	0,693 €	2	1,39 €
Mini USB - B	MOLEX 67503-1230 CONNECTOR, USB MINI B, RCPT, 5POS, SMT	Farnell	2313554	0,862 €	2	1,72 €
Microcontrolador	NXP LPC11E13FBD48/301 MCU, 32BIT, BRAZO CORTEX-M0, 48LQFP	Farnell	2115646	2,09 €	2	4,18 €
Módulo de radio	ANAREN A2500R24A00GM MÓDULO DE RADIO, TXRX, CC2500, INT ANT	Farnell	2096127	12,38 €	2	24,76 €
Conector USB - UART	FTDI FT232RL IC, USB A UART, SMD, 28SSOP	Farnell	1146032	5,33 €	2	10,66 €
Jumpers	MULTICOMP SPC20479 SHUNT JUMPER, 2POS, CODE STRIP LINE PLUG	Farnell	2396301	0,09 €	8	0,72 €
Pulsador	PANASONIC EVQ-Q2F03W SWITCH, TACTILE, SPST-NO, 20mA, 15VDC, SMD	Farnell	2460018	0,34 €	4	1,36 €
Diodos	KINGBRIGHT KA-3022SRC-4.5SF LED, SMD, ULTRA RED	Farnell	1142617	0,51 €	5	2,55 €
	KINGBRIGHT KA-3022SGC-4.5SF LED, SMD, ULTRA GREEN	Farnell	1142615	0,29 €	5	1,43 €
	ROHM SML-211DTT86K LED, 0805, LO-CUR. ORG	Farnell	1685055	0,71 €	6	4,27 €
	NXP PMLL4448 DIODO, ALTA VELOCIDAD, SOD-80C	Farnell	1097279	0,035 €	10	0,35 €
Terminal Blocks	MULTICOMP MC000029 TERMINAL BLOCK, WIRE TO BRD, 5POS, 12AWG	Farnell	2007998	1,270 €	4	5,08 €
Pin Headers	MULTICOMP 2211S-06G MACHO, THT, VERTICAL, 2,54 MM, 6 VÍAS	Farnell	1593415	0,19 €	10	1,85 €
Resistencias	MULTICOMP MC01W080511M RESISTENCIA, 1M, 0.1W, 1%, 0805	Farnell	9332413	0,041 €	50	2,04 €
	MULTICOMP MC01W08051270R RESISTENCIA, 270R, 0.1W, 1%, 0805	Farnell	9332936	0,040 €	50	1,99 €
	MULTICOMP MC01W08051330R RESISTENCIA, 330R, 0.1W, 1%, 0805	Farnell	9333037	0,041 €	50	2,04 €
	MULTICOMP MC01W08051750R RESISTENCIA, 750R, 0.1W, 1%, 0805	Farnell	9333525	0,041 €	50	2,06 €
	MULTICOMP MC01W080511K RESISTENCIA, 1K, 0.1W, 1%, 0805	Farnell	9332383	0,039 €	50	1,95 €
Condensadores	JOHANSON DIELECTRICS 501R15W103KV4E MLCC, 0805, X7R, 500V, 10 NF	Farnell	1886013	0,0994 €	50	4,97 €
	MULTICOMP MC0805F104Z160CT MLCC, 0805, Y5V, 16V, 100 NF	Farnell	1759144	0,0119 €	100	1,19 €
	PANASONIC EEEHA0J220R CONDENSADOR, ELECTROLÍTICO, 6,3 V, 22 UF	Farnell	1973274	0,302 €	1	0,30 €
	MULTICOMP MC0805F105Z250CT CAP, CERÁMICA, 1 UF, 25V, Y5V, 0805	Farnell	1759429	0,02 €	100	2,04 €
Inductores	TDK MIMZ2012R301A FERRITE BEAD, 0.15OHM, 600MA, 0805	Farnell	1669724	0,094 €	5	0,47 €
<b>TOTAL (Sin IVA)</b>						<b>79,36 €</b>
<b>TOTAL (Con 21% de IVA)</b>						<b>96,02 €</b>

Tabla 6.1: Costes de los componentes electrónicos

<b>COSTES ADICIONALES</b>	
<b>CONCEPTO</b>	<b>PRECIO</b>
Fabricación y envío de la PCB	27,15 €
Equipos del laboratorio y mesa de montaje	50,00 €
<b>TOTAL</b>	<b>77,15 €</b>

Tabla 6.2: Costes de fabricación y montaje

## 6.2. Costes personales

Tras tratar los gastos materiales, es el momento ahora de analizar los gastos personales, i.e. el esfuerzo y horas de dedicación por parte del ingeniero para lograr desarrollar el producto final.

Antes de enumerar las actividades desarrolladas y el cómputo total de horas, es preciso aclarar cuánto vale el trabajo del ingeniero. Para ello, y considerando que el trabajo de desarrollo ha sido elaborado por un ingeniero técnico (titulado universitario), los convenios actuales vigentes junto con las retenciones fiscales pertinentes, fijan el coste total del trabajador por parte de la empresa que se muestra en la tabla 6.3.

<b>COSTES INGENIERO</b>	
<b>CONCEPTO</b>	<b>PRECIO</b>
Salario bruto anual (convenio 2013)	23.500 €
Retenciones (Seguridad Social, Desempleo, etc.)	29,90 %
Precio/hora (año laboral de 1.770 horas)	17,25 €
<b>TOTAL</b>	<b>30.526,5 €</b>

Tabla 6.3: Costes de los ingenieros

En base a lo recogido en la tabla anterior, podemos ahora analizar el parte de trabajo desarrollado por el ingeniero, así como el coste asociado a dicho trabajo. Este aspecto se recoge en la tabla 6.4.

<b>REGISTRO DE DEDICACIÓN</b>		
<b>TAREA</b>	<b>HORAS EMPLEADAS</b>	<b>COSTE ASOCIADO</b>
<b><u>Arranque</u></b>		
Especificación y captura de requisitos	1	17,25 €
<b><u>Aprendizaje y estudio</u></b>		
Microcontrolador LPC11E13	15	258,75 €
Módulo transceptor de radio ANAREN	12	207,00 €
IDE wxDevC++ y librería wxWidgets	8	138,00 €
<b><u>Diseño Hardware</u></b>		
Elección y compra de componentes	4	69,00 €
Creación de componentes y sus huellas	5	86,25 €
Captura esquemática (posicionamiento y rutado)	4	69,00 €
Análisis de consumo y generación de B.O.M	3	51,75 €
Layout PCB (posicionamiento y rutado)	10	172,50 €
Planos de masa, serigrafía y generación ficheros Gerber	3	51,75 €
Documentación y control de cambios	3	51,75 €
<b><u>Fabricación</u></b>		
Artesanía (montaje de componentes)	15	258,75 €
Verificación del <i>hardware</i> y resolución de problemas	7	120,75 €
Documentación y control de cambios	2	34,50 €
<b><u>Diseño Software</u></b>		
Programación y desarrollo del <i>firmware</i>	70	1.207,50 €
Verificación del <i>firmware</i> y resolución de problemas	10	172,50 €
Elaboración de aplicación para PC	35	603,75 €
Verificación del <i>software</i> y resolución de problemas	10	172,50 €
Documentación y control de cambios	4	69,00 €

<b>Cierre</b>		
Cumplimiento de las especificaciones y requisitos	2	34,50 €
Análisis de prestaciones	3	51,75 €
<b>Documentación</b>		
Redacción de la memoria	75	1.293,75 €
Elaboración de presentación	10	172,50 €
<b>TOTAL</b>	<b>311</b>	<b>5.364,75 €</b>

Tabla 6.4: Parte de trabajo realizado y coste asociado

### 6.3. Coste total

Una vez hecho el desglose total de los gastos incurridos en el proyecto realizado, es el momento de ponerlos todos en conjunto para comprobar cuál ha sido el coste total del proyecto. Esto se puede observar en la tabla 6.5.

<b>COSTE TOTAL</b>	
<b>CONCEPTO</b>	<b>PRECIO</b>
Costes componentes electrónicos	96,02 €
Costes adicionales	77,15 €
Costes dedicación ingeniero	5.364,75 €
<b>TOTAL</b>	<b>5.537,92 €</b>

Tabla 6.5: Costes totales del proyecto realizado



# Capítulo 7

## Conclusiones y líneas futuras

*Perfection is attained not when there is nothing more to add,  
but when there is nothing more to obtain.  
~ Antoine de Saint Exupéry ~*

Para finalizar este Trabajo Fin de Grado me gustaría comentar brevemente cuáles han sido las conclusiones obtenidas así como las posibles líneas futuras de desarrollo del mismo.

El proyecto que ha sido realizado ha resultado ser una experiencia muy positiva y enriquecedora. Abarcar el diseño de un producto desde el comienzo, con la captura de requisitos y especificaciones, hasta las pruebas finales de cumplimiento de los mismos supone un reto al que todo ingeniero debe enfrentarse tarde o temprano en el mundo laboral. Por este motivo considero que el trabajo realizado ha resultado idóneo en este sentido, y me ha ayudado a adquirir una visión global de todo lo que supone el desarrollo de un producto desde la idea inicial hasta la venta del mismo. Asimismo, tener experiencia en cada una de las fases de desarrollo de un producto siempre es una experiencia muy positiva, pues uno al final nunca sabe si algún día tendrá que trabajar en alguna de estas áreas.

De este modo, a continuación paso a enumerar algunos de los principales aspectos que considero que este proyecto me ha permitido conocer:

- Conocimiento de todas y cada una de las fases de desarrollo de un producto.
- Manejo de herramientas CAD de captura esquemática y *layout* de sistemas electrónicos.
- Diseño, montaje y verificación de una placa de circuito impreso.

- Conocimiento de la arquitectura de los procesadores *Cortex-M0* y microcontroladores que la implementan.
- Conocimiento de la arquitectura y manera de trabajar de un *chip* de radio.
- Comprensión de la problemática que supone la comunicación vía radio y la necesidad de mecanismos de detección y corrección de errores.
- Formalizar un protocolo de comunicaciones capaz de comunicar de manera fiable varios módulos de radio entre sí.
- Diseño y elaboración de una interfaz gráfica de usuario que permita la comunicación entre el PC y los módulos de radio.

Si bien es cierto que los requisitos recogidos en la primera fase del proyecto se han conseguido cumplir, nunca se puede decir que no haya algún aspecto modificable o mejorable. Por este motivo, y para concluir el trabajo deojo a continuación algunas breves ideas acerca de los cambios y mejoras que futuras versiones del prototipo diseñado pudieran implementar. Éstas son principalmente:

- Si la máxima potencia de transmisión que se puede obtener, que es de +1dBm, se nos queda pequeña para el alcance a ciertas distancias, se puede optar por rediseñar el *layout* de la PCB e incorporar un amplificador de potencia entre el integrado CC2500 y la antena. Del mismo modo, se podría intentar sustituir la antena en forma de diapasón actual por una que fuese del tipo dipolo de media longitud de onda que presenta un patrón de radiación más eficiente.
- Actualmente la banda de transmisión de la señal está fijada a una concreta dentro de los 2.4-2.4835 Ghz. Sin embargo, en caso de una interferencia con algún dispositivo cercano más potente en la misma banda de frecuencias puede interrumpir las comunicaciones de manera permanente. De este modo, se podría dotar a los módulos de la capacidad de cambiar sus bandas de emisión en caso de que detecten un número de paquetes erróneos elevados y poder así continuar funcionando. Para ello habría que asegurarse que todos los módulos han cambiado de banda, o de lo contrario podríamos dejar incomunicado a alguno de ellos.

# Apéndice A

## Conjunto de instrucciones del procesador Cortex-M0

A continuación se muestra una tabla con un resumen del conjunto reducido de instrucciones que emplea el procesador Cortex-M0.

Operation	Description	Assembler	Cycles
Move	8-bit immediate	MOVS Rd, #<imm>	1
	Lo to Lo	MOVS Rd, Rm	1
	Any to Any	MOV Rd, Rm	1
	Any to PC	MOV PC, Rm	3
Add	3-bit immediate	ADDS Rd, Rn, #<imm>	1
	All registers Lo	ADDS Rd, Rn, Rm	1
	Any to Any	ADD Rd, Rd, Rm	1
	Any to PC	ADD PC, PC, Rm	3
Add	8-bit immediate	ADDS Rd, Rd, #<imm>	1
	With carry	ADCS Rd, Rd, Rm	1
	Immediate to SP	ADD SP, SP, #<imm>	1
	From address from SP	ADD Rd, SP, #<imm>	1
	From address from PC	ADR Rd, <label>	1
Subtract	Lo and Lo	SUBS Rd, Rn, Rm	1
	3-bit immediate	SUBS Rd, Rn, #<imm>	1
	8-bit immediate	SUBS Rd, Rd, #<imm>	1
	With carry	SBCS Rd, Rd, Rm	1
	Immediate from SP	SUB SP, SP, #<imm>	1
	Negate	RSBS Rd, Rn, #0	1
Multiply	Multiply	MULS Rd, Rm, Rd	1

Operation	Description	Assembler	Cycles
Compare	Compare	CMP Rn, Rm	1
	Negative	CMN Rn, Rm	1
	Immediate	CMP Rn, #<imm>	1
Logical	AND	ANDS Rd, Rd, Rm	1
	Exclusive OR	EORS Rd, Rd, Rm	1
	OR	ORRS Rd, Rd, Rm	1
	Bit clear	BICS Rd, Rd, Rm	1
	Move NOT	MVNS Rd, Rm	1
	AND test	TST Rn, Rm	1
Shift	Logical shift left by immediate	LSLs Rd, Rm, #<shift>	1
	Logical shift left by register	LSLs Rd, Rd, Rs	1
	Logical shift right by immediate	LSRS Rd, Rm, #<shift>	1
	Logical shift right by register	LSRS Rd, Rd, Rs	1
	Arithmetic shift right	ASRS Rd, Rm, #<shift>	1
	Arithmetic shift right by regist	ASRS Rd, Rd, Rs	1
Rotate	Rotate right by register	RORS Rd, Rd, Rs	1
Load	Word, immediate offset	LDR Rd, [Rn, #<imm>]	2
	Halfword, immediate offset	LDRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	LDRB Rd, [Rn, #<imm>]	2
	Word, register offset	LDR Rd, [Rn, Rm]	2
	Halfword, register offset	LDRH Rd, [Rn, Rm]	2
	Signed halfword, register offset	LDRSH Rd, [Rn, Rm]	2
	Byte, register offset	LDRB Rd, [Rn, Rm]	2
	Signed byte, register offset	LDRSB Rd, [Rn, Rm]	2
	PC-relative	LDR Rd, <label>	2
	SP-relative	LDR Rd, [SP, #<imm>]	2
	Multiple, excluding base	LDM Rn!, {<loreglist>}	1 + N <sup>[1]</sup>
	Multiple, including base	LDM Rn, {<loreglist>}	1 + N <sup>[1]</sup>
Store	Word, immediate offset	STR Rd, [Rn, #<imm>]	2
Store	Halfword, immediate offset	STRH Rd, [Rn, #<imm>]	2
	Byte, immediate offset	STRB Rd, [Rn, #<imm>]	2
	Word, register offset	STR Rd, [Rn, Rm]	2
	Halfword, register offset	STRH Rd, [Rn, Rm]	2
	Byte, register offset	STRB Rd, [Rn, Rm]	2
	SP-relative	STR Rd, [SP, #<imm>]	2
	Multiple	STM Rn!, {<loreglist>}	1 + N <sup>[1]</sup>
Push	Push	PUSH {<loreglist>}	1 + N <sup>[1]</sup>
	Push with link register	PUSH {<loreglist>, LR}	1 + N <sup>[1]</sup>
Pop	Pop	POP {<loreglist>}	1 + N <sup>[1]</sup>
	Pop and return	POP {<loreglist>, PC}	4 + N <sup>[2]</sup>

Operation	Description	Assembler	Cycles
Branch	Conditional	B<cc> <label>	1 or 3 <sup>[3]</sup>
	Unconditional	B <label>	3
	With link	BL <label>	4
	With exchange	BX Rm	3
	With link and exchange	BLX Rm	3
Extend	Signed halfword to word	SXTH Rd, Rm	1
	Signed byte to word	SXTB Rd, Rm	1
	Unsigned halfword	UXTH Rd, Rm	1
	Unsigned byte	UXTB Rd, Rm	1
Reserve	Bytes in word	REV Rd, Rm	1
	Bytes in both halfwords	REV16 Rd, Rm	1
	Signed bottom half word	REVSH Rd, Rm	1
State change	Supervisor Call	SVC <imm>	- <sup>[4]</sup>
	Disable interrupts	CPSID i	1
	Enable interrupts	CPSIE i	1
	Read special register	MRS Rd, <specreg>	4
	Write special register	MSR <specreg>, Rn	4
Hint	Send event	SEV	1
	Wait for interrupt	WFI	2 <sup>[5]</sup>
	Yield	YIELD <sup>[6]</sup>	1
	No operation	NOP	1
Barriers	Instruction synchronization	ISB	4
	Data memory	DMB	4
	Data synchronization	DSB	4

[1] N is the number of elements.

[2] N is the number of elements in the stack-pop list including PC and assumes load or store does not generate a HardFault exception.

[3] 3 if taken, 1 if not taken.

[4] Cycle count depends on core and debug configuration.

[5] Excludes time spend waiting for an interrupt or event.

[6] Executes as NOP.

Tabla A.1: Conjunto de instrucciones del procesador Cortex-M0



# Apéndice B

## Esquemáticos del diseño realizado

A continuación se muestran los esquemáticos correspondientes al diseño realizado, así como los componentes que conforman cada uno de ellos:

- Esquemático del módulo de radio:
  - 1 Microcontrolador LPC11E13
  - 1 Módulo de radio ANAREN
  - 2 botones
  - 3 condensadores
  - 5 resistencias
  - 1 diodo de protección y 2 diodos LED
- Esquemático de la red de alimentación y conversión USB-UART
  - 1 conector USB mini B
  - 1 conversor FTDI USB a UART
  - 1 regulador de tensión 5V-3.3V
  - 9 condensadores
  - 6 resistencias
  - 1 bobina
  - 3 diodos LED
- Esquemático de consumo de corriente

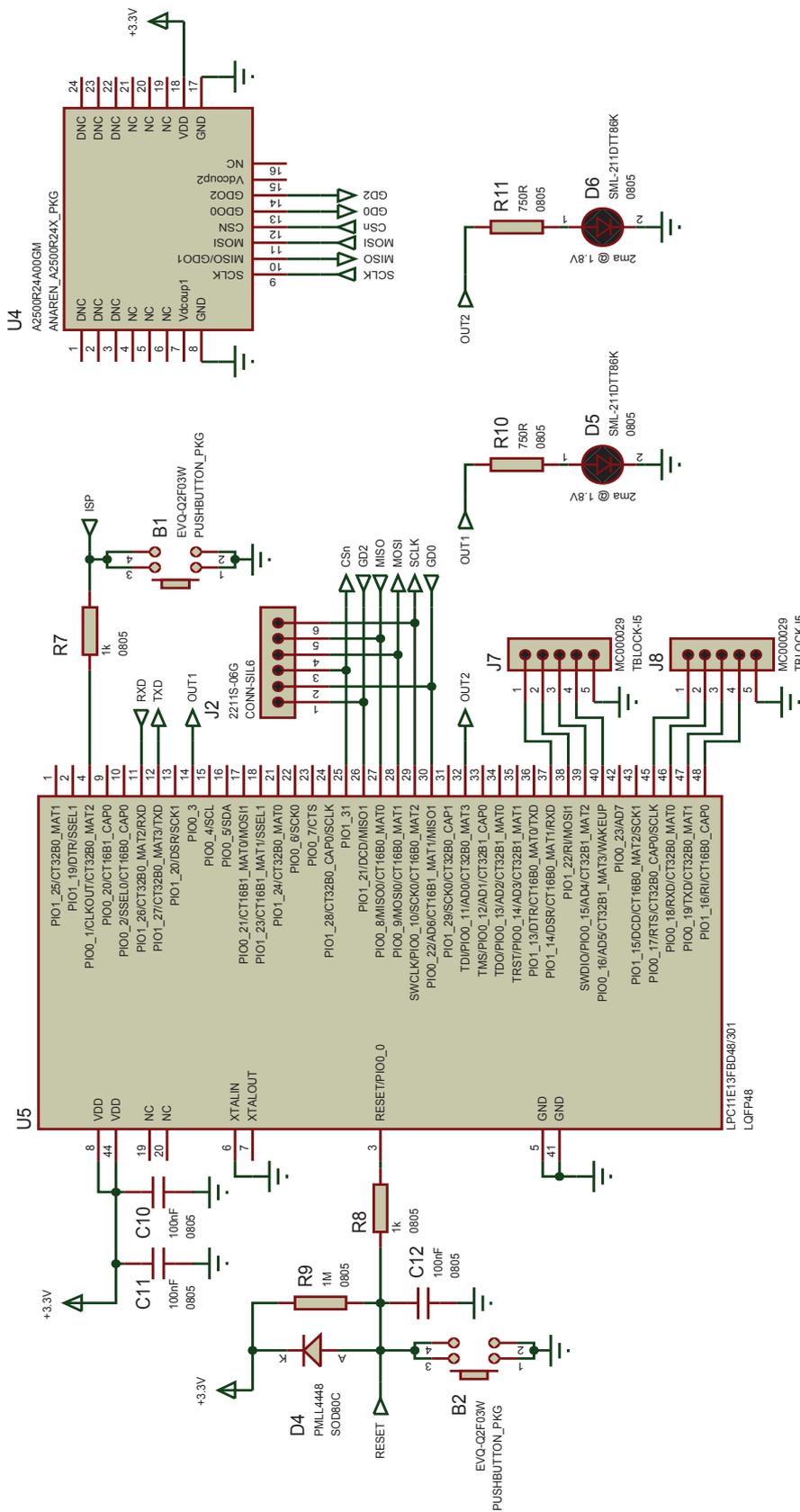
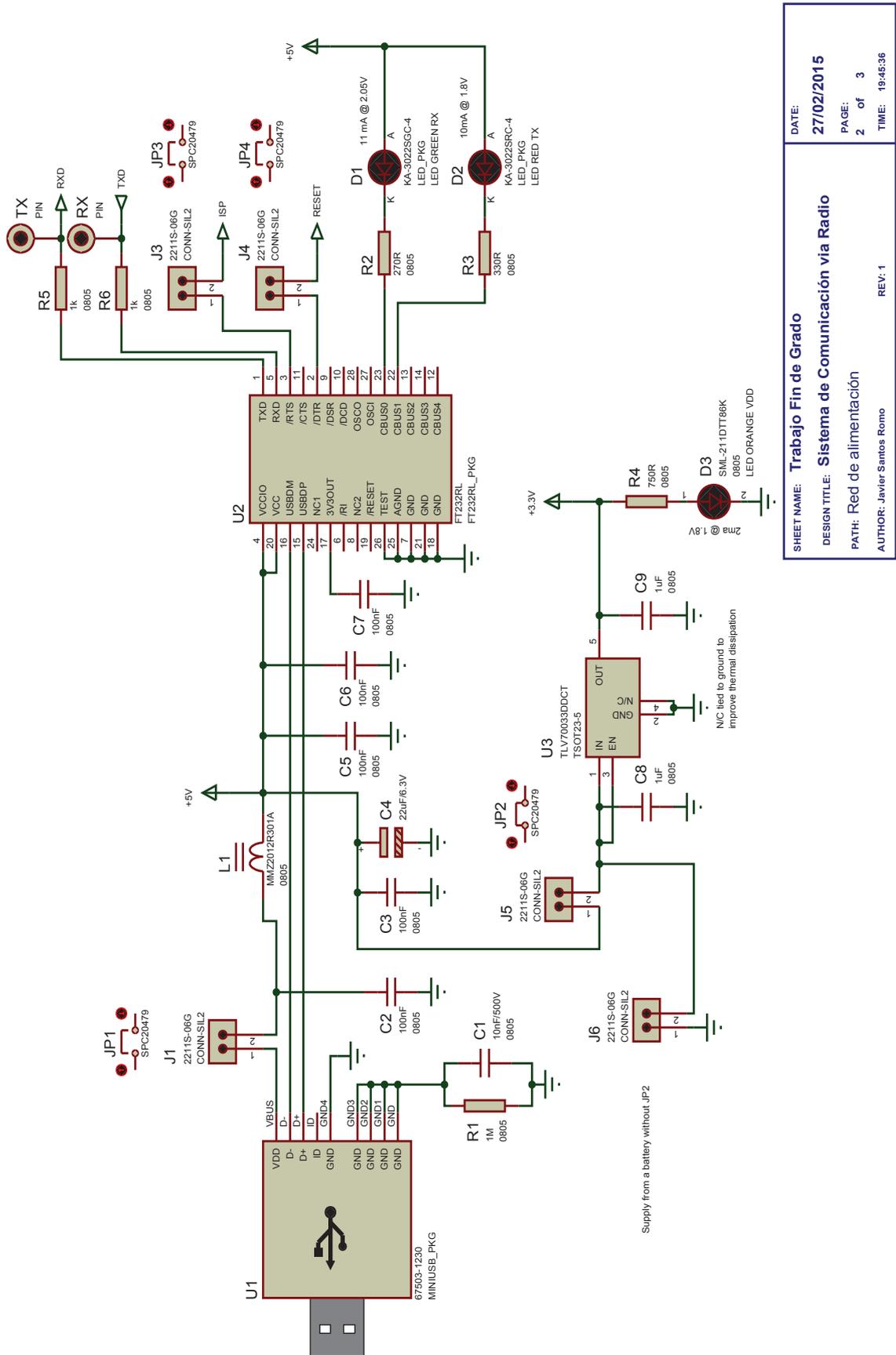


Figura B.1: Esquemático correspondiente al módulo de radio

SHEET NAME: <b>Trabajo Fin de Grado</b> DESIGN TITLE: <b>Sistema de Comunicación via Radio</b> PATH: <b>Módulo de Radio</b> AUTHOR: <b>Javier Santos Romo</b>		DATE: <b>27/02/2015</b> PAGE: <b>1 of 3</b> TIME: <b>19:45:36</b>
		REV: 1



SHEET NAME: Trabajo Fin de Grado	DATE: 27/02/2015
DESIGN TITLE: Sistema de Comunicación via Radio	PAGE: 2 of 3
PATH: Red de alimentación	TIME: 19:46:36
AUTHOR: Javier Santos Romo	REV: 1

Figura B.2: Esquemático correspondiente a la red de alimentación y conversión

### -> CURRENT CONSUMPTION:

LPC11E13FBD48/301  
 (- 7mA @ 3.3V a 50MHz CCLK) Normal Mode  
 (- 1mA @ 3.3V a 12MHz CCLK) Sleep Mode  
 (- 0.22mA @ 3.3V a 48MHz IRC)  
 (- 0.21mA @ 3.3V a 48MHz PLL)  
 (- 0.15mA @ 3.3V a 48MHz ROM)  
 (- 0.29mA @ 3.3V a 48MHz ADC)  
 (- 2x0.06mA @ 3.3V a 48MHz Timer)  
 (- 0.88mA @ 3.3V a 48MHz GPIO)  
 (- 0.10mA @ 3.3V a 48MHz IOCONFIG)  
 (- 0.45mA @ 3.3V a 48MHz SPI)  
 (- 0.82mA @ 3.3V a 48MHz UART)  
 - 33.80mW (10.24mA @ 3.3V)

FT232RL

- 75mW (15mA @ 5V Normal Operation)  
 KA-3022SRC-4 and KA-3022SGC-4  
 - (10mA + 11mA @ 5V)

SML-211DIT86K x3

- (2mA + 2mA + 2mA @ 3.3V)

### -> POWER DISSIPATION (REGULATOR) :

TLV70033DDCT:

-  $R_{th}(J-A) = 280^{\circ}C/W$   
 -  $I_o = 50mA \rightarrow V_{dropout} = 1.7V \rightarrow P_{dis} = 85mW$   
 -  $25^{\circ}C + 85mW \times 280^{\circ}C/W = 48.8^{\circ}C$

A2500R24A00GM

(- 17.25 mA in RX @ 250 kBaud)  
 (- 21.5 mA in TX @ 1dBm)  
 (- 17.675 mA RX/TX 90%)  
 - 58.33mW (17.675mA @ 3.3V)

3.3V		5V
-----		
10.24mA		15mA
+ 17.68mA		+ 10mA
+ 6mA		+ 11mA
-----		
33.9mA	->	36mA -> 69.9mA

### -> TOTAL:

< 500mA (Supplied by USB Standar)

SHEET NAME: <b>Trabajo Fin de Grado</b>	DATE: <b>16/05/2015</b>
DESIGN TITLE: <b>Sistema de Comunicación via Radio</b>	PAGE: <b>3 of 3</b>
PATH: <b>Consumo de corriente estimado</b>	TIME: <b>21:23:07</b>
AUTHOR: <b>Javier Santos Romo</b>	REV: <b>1</b>

Figura B.3: Esquemático correspondiente al consumo de corriente estimado

# Apéndice C

## Layouts del diseño realizado

A continuación se presentan las diferentes capas de *layouts* del diseño realizado. Así, se tienen las correspondientes a las capas de cobre de la cara superior e inferior respectivamente, la máscara de cobre de la capa superior que recoge los *pads* de los diferentes integrados, la capa de pasivado *resist* de ambas caras mostrada como el negativo de la misma, la capa de serigrafía de la cara superior, y finalmente la capa mecánica y los agujeros y vías realizados.

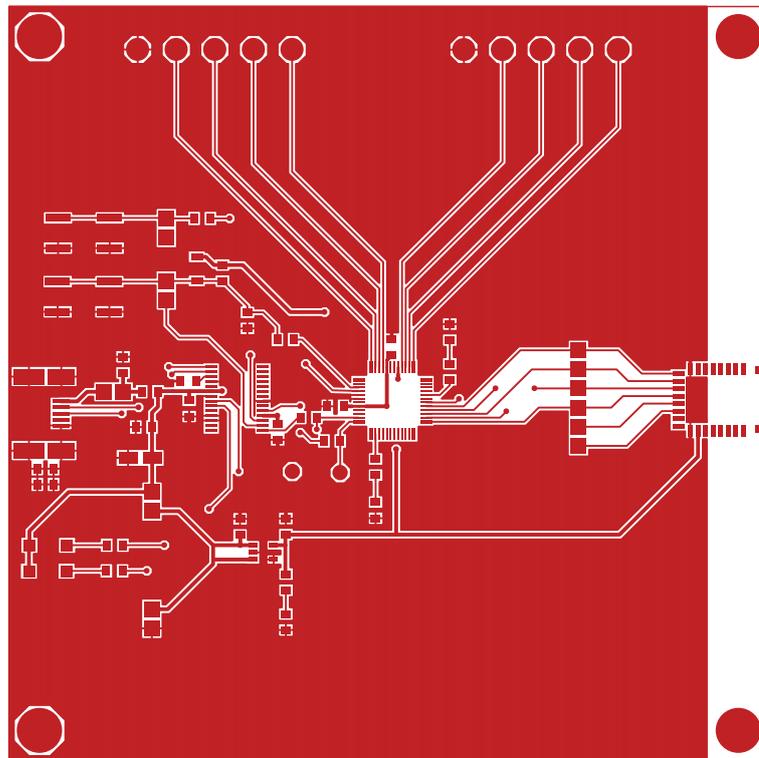


Figura C.1: Placa PCB realizada: *Top copper*

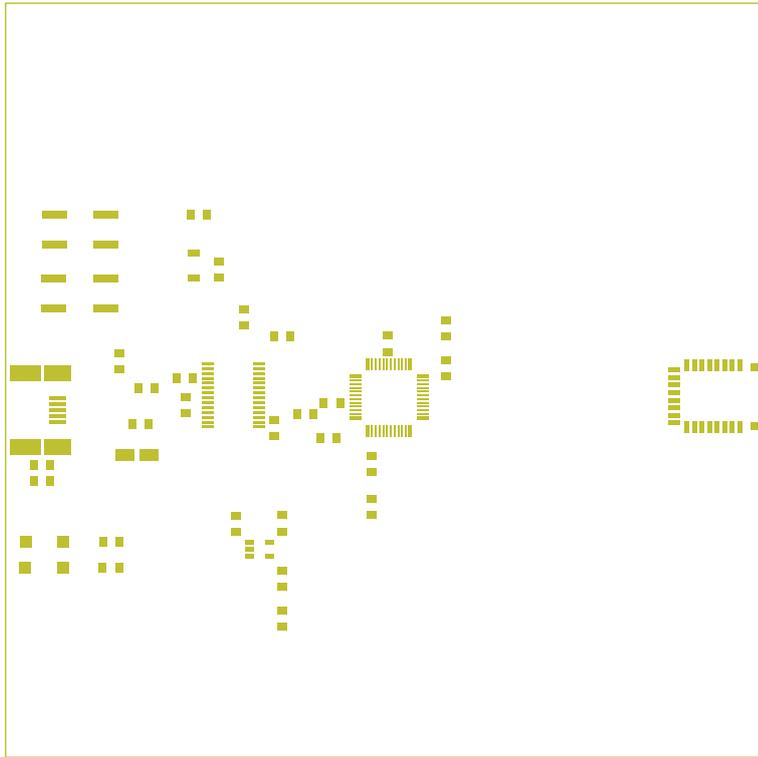


Figura C.2: Placa PCB realizada: *Top mask*

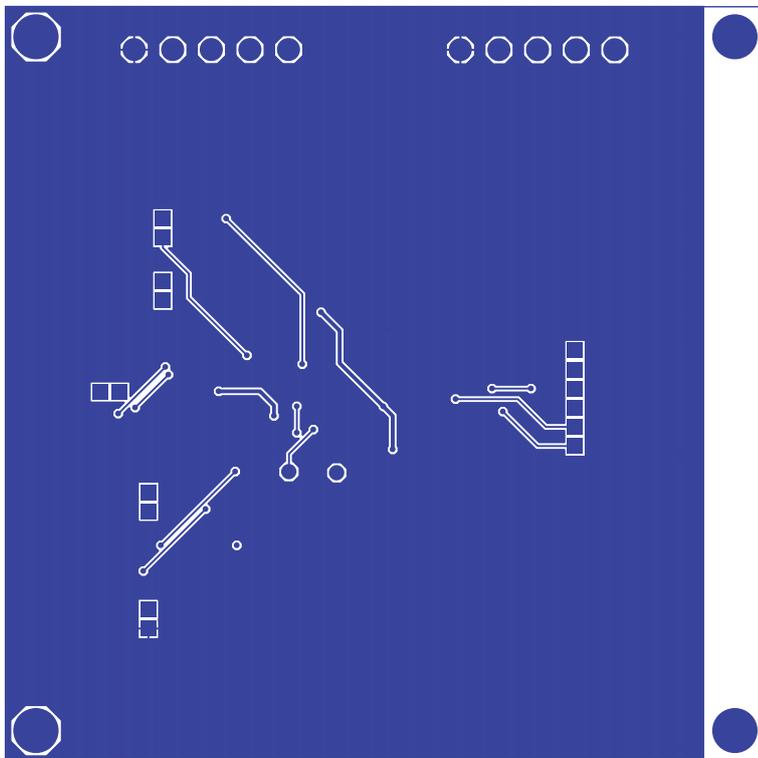


Figura C.3: Placa PCB realizada: *Bottom copper*

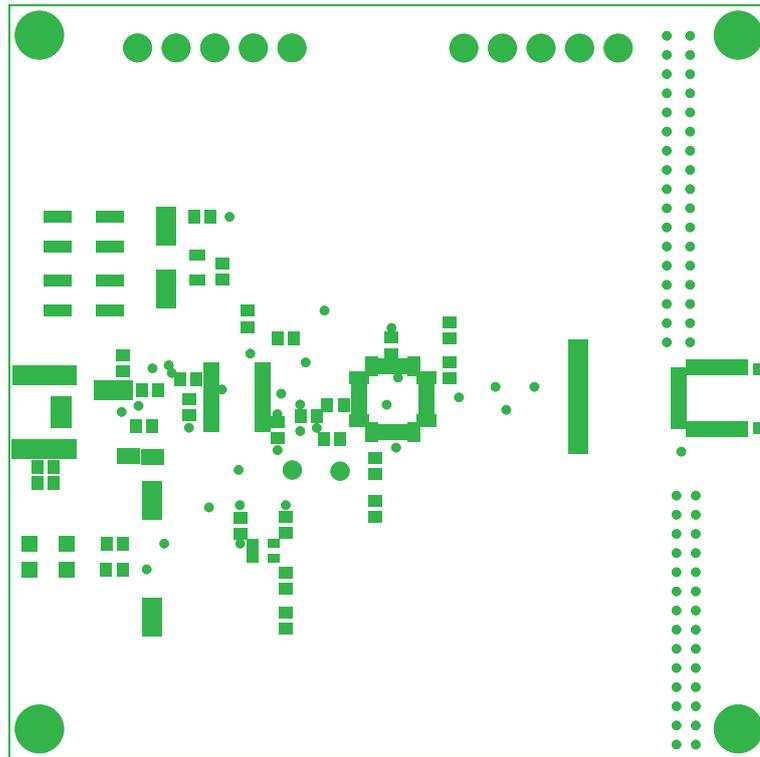


Figura C.4: Placa PCB realizada: *Top resist*

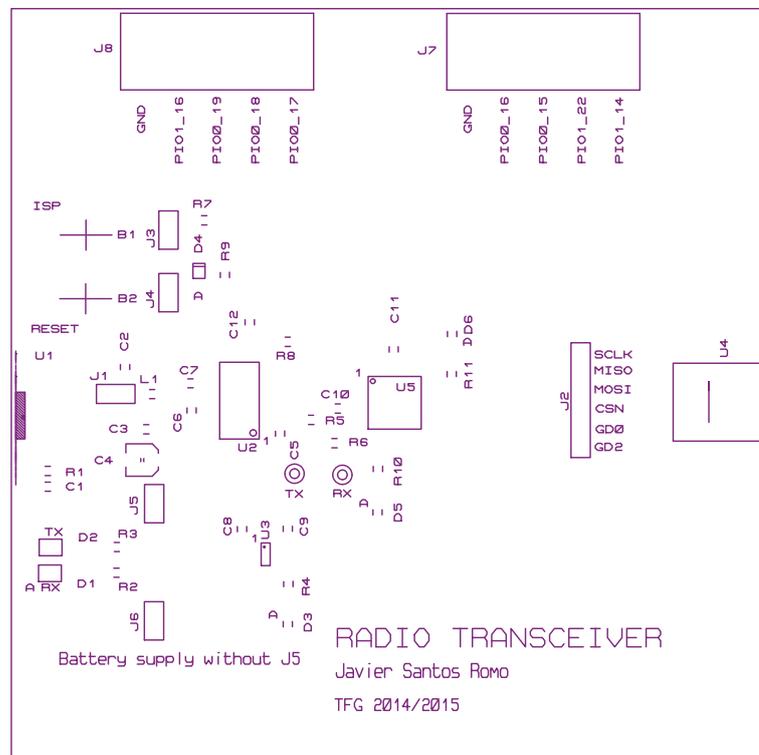


Figura C.5: Placa PCB realizada: *Top silk*

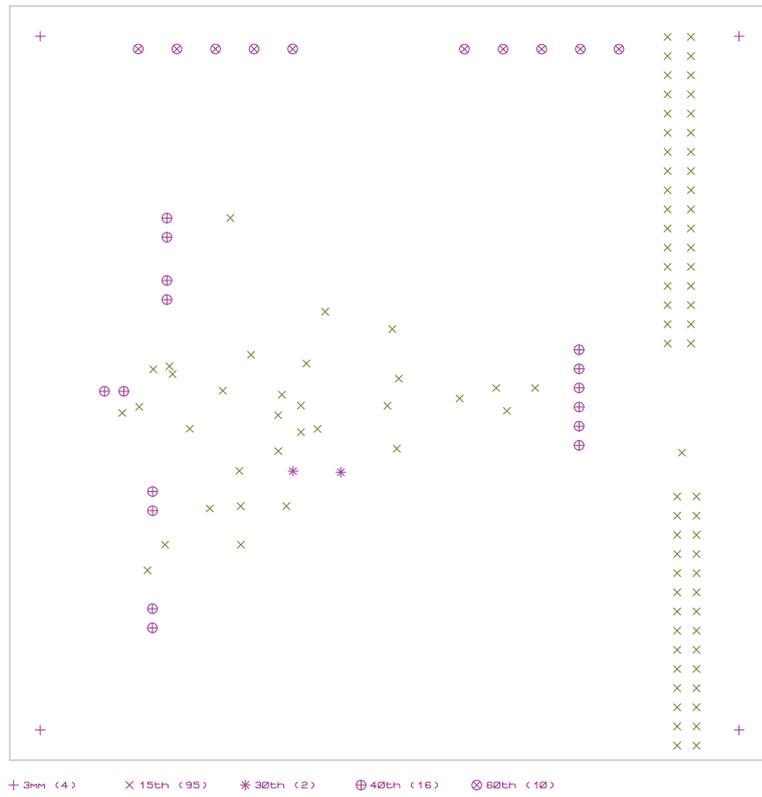


Figura C.6: Placa PCB realizada: *Mechanic and drill holes*

# Apéndice D

## Código del módulo maestro

A continuación se lista el código correspondiente al módulo de radio maestro.

```
1  /*****
2  * @file      init.c
3  * @brief     Interrupts Vector Table and Clock and PLL initialization
4  * @version   V2.00
5  * @date      17. June 2015
6  *****/
7
8  #include "lpc11E13.h"
9
10 void init(void);
11 //void NMI_interrupt(void);
12 //void Hard_Fault_interrupt(void);
13 //void SVCcall_interrupt(void);
14 //void PendSV_interrupt(void);
15 //void SysTic_interrupt(void);
16 void PIN_INT0_interrupt(void);
17 //void PIN_INT1_interrupt(void);
18 //void PIN_INT2_interrupt(void);
19 //void PIN_INT3_interrupt(void);
20 //void PIN_INT4_interrupt(void);
21 //void PIN_INT5_interrupt(void);
22 //void PIN_INT6_interrupt(void);
23 //void PIN_INT7_interrupt(void);
24 //void GINT0_interrupt(void);
25 //void GINT1_interrupt(void);
26 void Default_Handler(void);
27 //void SSP1_interrupt(void);
28 //void I2C_interrupt(void);
29 void CT16B0_interrupt(void);
30 //void CT16B1_interrupt(void);
31 //void CT32B0_interrupt(void);
32 //void CT32B1_interrupt(void);
33 //void SSP0_interrupt(void);
34 void USART_interrupt(void);
```

```
35 //void ADC_interrupt(void);
36 //void WWDT_interrupt(void);
37 //void BOD_interrupt(void);
38 //void FLASH_EEPROM_interrupt(void);
39 //void IOH_interrupt(void);
40
41 // The following are 'declared' in the linker script
42 extern unsigned char INIT_DATA_VALUES;
43 extern unsigned char INIT_DATA_START;
44 extern unsigned char INIT_DATA_END;
45 extern unsigned char BSS_START;
46 extern unsigned char BSS_END;
47
48 // The section "vectors" is placed at the beginning of flash by the linker
   script
49 const void * Vectors[] __attribute__((section(".vectors"))) =
50 {
51     (void *)0x10002000,    /* Top of stack */
52     init,                 /* Reset Handler */
53     Default_Handler,     /*NMI_interrupt,        /* NMI */
54     Default_Handler,     /*Hard_Fault_interrupt, /* Hard Fault */
55     0,                    /* Reserved */
56     0,                    /* Reserved */
57     0,                    /* Reserved */
58     0,                    /* Reserved */
59     0,                    /* Reserved */
60     0,                    /* Reserved */
61     0,                    /* Reserved */
62     Default_Handler,     /*SVCcall_interrupt,   /* SVC */
63     0,                    /* Reserved */
64     0,                    /* Reserved */
65     Default_Handler,     /*PendSV_interrupt,    /* PendSV */
66     Default_Handler,     /*SysTic_interrupt,    /* SysTick */
67
68 /* External interrupt handlers follow */
69
70     PIN_INT0_interrupt,   /* GPIO pin interrupt 0 (0) */
71     Default_Handler,     /*PIN_INT1_interrupt,   /* GPIO pin
       interrupt 1 (1) */
72     Default_Handler,     /*PIN_INT2_interrupt,   /* GPIO pin
       interrupt 2 (2) */
73     Default_Handler,     /*PIN_INT3_interrupt,   /* GPIO pin
       interrupt 3 (3) */
74     Default_Handler,     /*PIN_INT4_interrupt,   /* GPIO pin
       interrupt 4 (4) */
75     Default_Handler,     /*PIN_INT5_interrupt,   /* GPIO pin
       interrupt 5 (5) */
76     Default_Handler,     /*PIN_INT6_interrupt,   /* GPIO pin
       interrupt 6 (6) */
77     Default_Handler,     /*PIN_INT7_interrupt,   /* GPIO pin
       interrupt 7 (7) */
78     Default_Handler,     /*GINT0_interrupt,      /* GPIO GROUP0
       interrupt (8) */
```

```

79     Default_Handler,           //GINT1_interrupt,           /* GPIO GROUP0
      interrupt (9) */
80     Default_Handler,           /* RESERVED (10) */
81     Default_Handler,           /* RESERVED (11) */
82     Default_Handler,           /* RESERVED (12) */
83     Default_Handler,           /* RESERVED (13) */
84     Default_Handler,           //SSP1_interrupt,           /* SSP1 interrupt
      (14) */
85     Default_Handler,           //I2C_interrupt,           /* I2C interrupt
      (15) */
86     CT16B0_interrupt,          /* CT16B0 interrupt (16) */
87     Default_Handler,           //CT16B1_interrupt,        /* CT16B1 interrupt
      (17) */
88     Default_Handler,           //CT32B0_interrupt,        /* CT32B0 interrupt
      (18) */
89     Default_Handler,           //CT32B1_interrupt,        /* CT32B1 interrupt
      (19) */
90     Default_Handler,           //SSP0_interrupt,           /* SSP0 interrupt
      (20) */
91     USART_interrupt,           /* UART interrupt (21) */
92     Default_Handler,           /* RESERVED (22) */
93     Default_Handler,           /* RESERVED (23) */
94     Default_Handler,           //ADC_interrupt,           /* ADC interrupt
      (24) */
95     Default_Handler,           //WWDT_interrupt,          /* WDT interrupt
      (25) */
96     Default_Handler,           //BOD_interrupt,           /* BOD interrupt
      (26) */
97     Default_Handler,           //FLASH_EEPROM_interrupt, /* Flash/EEPROM
      interrupt (27) */
98     Default_Handler,           /* RESERVED (28) */
99     Default_Handler,           /* RESERVED (29) */
100    Default_Handler,           /* RESERVED (30) */
101    Default_Handler,           //IOH_interrupt            /* I/O Handler
      interrupt (31) */
102 };
103
104 extern inline clock_init()
105 {
106     // This function sets the main clock to the PLL
107     // The PLL input is the built in 12MHz RC oscillator
108     // This is multiplied up to 48MHz for the main clock (MSEL = 3, PSEL =
109     // 1)
110     SYSPLLCLKSEL = 0;           // select internal RC oscillator
111     SYSPLLCTRL = 3 | (1 << 5); // set divisors/multipliers
112     MAINCLKSEL = 3;           // Use PLL output as main clock
113     PDRUNCFG &= ~(1<<7);     // Power up the PLL. This MUST be done
      before update ~xxUEN registers
114     SYSPLLCLKUEN = 0;         // inform PLL of update
115     SYSPLLCLKUEN = 1;
116     MAINCLKUEN = 0;          // Inform core of clock update
117     MAINCLKUEN = 1;

```

```

118     while(!(SYSPLLSTAT & 1)); //Wait until PLL is locked
119 }
120
121 void init()
122 {
123     // do global/static data initialization
124     unsigned char *src;
125     unsigned char *dest;
126     unsigned len;
127     src= &INIT_DATA_VALUES;
128     dest= &INIT_DATA_START;
129     len= &INIT_DATA_END-&INIT_DATA_START;
130     while (len--)
131         *dest++ = *src++;
132
133     // zero out the uninitialized global/static variables
134     dest = &BSS_START;
135     len = &BSS_END - &BSS_START;
136     while (len--)
137         *dest++=0;
138
139     clock_init();
140     main();
141 }
142
143 void Default_Handler()
144 {
145     _printf("\nDefault Handler. -Halt-\n");
146     while(1);
147 }

```

Código D.1: Fichero de inicialización del módulo maestro

```

1 /*****
2 *                                     TRABAJO FIN DE GRADO                               *
3 *****/
4 *                                     *
5 *      Desarrollo de un Sistema de Comunicacion Digital via Radio                    *
6 *                                     para Entornos Industriales                       *
7 *                                     *
8 *****/
9 * @author:      Javier Santos Romo                                               *
10 *****/
11 * @brief:      Main program and subfunctions                                     *
12 *****/
13 * Revision history:      17/06/2015                                             *
14 *****/
15
16 /*****
17 *                                     Header files                               *
18 *****/

```

```

19
20 #include "lpc1114.h"
21 #include "cc2500.h"
22
23 /*****
24         Clocks and Interrupts
25 *****/
26
27 #define enable_interrupts() asm("cpsie i")
28 #define disable_interrupts() asm("cpsid i")
29
30 #define PCLK 48000000
31 #define _delay_us(n) delay_loop((n*(PCLK/1000)-6000)/4000)
32 #define _delay_ms(n) delay_loop((n*(PCLK/1000)-6)/4)
33
34 void delay_loop(unsigned int) __attribute__((naked));
35 void delay_loop(unsigned int d)
36 {
37     asm volatile ("Ldelay_loop: sub r0,#1");
38     asm volatile ("bne Ldelay_loop");
39     asm volatile ("bx lr");
40 }
41
42 //-----
43 // int U0putch(int data)
44 //
45 // DESCRIPTION:
46 //     This function is used to transmit a data through UART
47 //
48 // ARGUMENTS:
49 //     Data to transfer
50 //-----
51
52 int U0putch(int data)
53 {
54     while(!(ULSR & 0x00000040)); //Wait until TEMPTY is empty
55     UTHR = data; //Transmit data
56 }
57
58 //-----
59 // unsigned char U0getch()
60 //
61 // DESCRIPTION:
62 //     This function is used to get a data from UART
63 //
64 // ARGUMENTS:
65 //     None
66 //-----
67
68 unsigned char U0getch()
69 {
70     while(!(ULSR & 0x00000001)); //Wait until RBR is empty
71     return URBR; //Return data

```

```

72 }
73
74 #define putchar(c) U0putch(c)
75 #include "printf.c"
76
77 /*****
78         Definitions
79 *****/
80
81 #define LENGTHFRAME 16
82 #define COMMANDLENGTH 12
83 #define MAXITER 10
84
85 unsigned char buf_TX[64]={0};
86 unsigned char buf_RX[64]={0};
87
88 const unsigned char address_master[4] ="maes";
89
90 volatile unsigned char iter=0;           //Number of times a message has
    been retransmitted
91
92 volatile unsigned char received_packet = 0; //Indicate if one packet has
    been received
93
94 volatile unsigned char temp_expire = 0; //Indicate if temp has been
    overflowed
95
96 volatile unsigned char EOF = 'L';       //Indicate End Of Frame
97
98 volatile unsigned char message_OK = 2;  //Indicate if the message received
    is correct
99
    //0 -> Not OK  1 -> OK  2 ->
    Tristate
100
101 volatile unsigned char readcommand[COMMANDLENGTH]={0}; //Master reads
    the command from the computer
102
103 volatile unsigned char num_read_bytes = 0;
104
105 const unsigned char RFSettings[] ={
106     0x01, // IOCFG2           GDO2Output Pin Configuration
107     0x2E, // IOCFG1           GDO1Output Pin Configuration
108     0x3F, // IOCFG0           GDO0Output Pin Configuration
109     0x07, // FIFOTHR         RX FIFO and TX FIFO Thresholds
110     0xD3, // SYNC1          Sync Word, High Byte
111     0x91, // SYNC0          Sync Word, Low Byte
112     0x10, // PKTLEN         Packet Length (16 bytes)
113     0x0E, // PKTCTRL1       Packet Automation Control
114     0x44, // PKTCTRL0       Packet Automation Control
115     0x6D, // ADDR           Device Address
116     0x00, // CHANNR         Channel Number
117     0x0A, // FSCTRL1        Frequency Synthesizer Control
118     0x00, // FSCTRL0        Frequency Synthesizer Control

```

```

119         0x5D, // FREQ2           Frequency Control Word, High Byte
120         0x93, // FREQ1           Frequency Control Word, Middle Byte
121         0xB1, // FREQ0           Frequency Control Word, Low Byte
122         0x2D, // MDMCFG4           Modem Configuration
123         0x3B, // MDMCFG3           Modem Configuration
124         0x73, // MDMCFG2           Modem Configuration
125         0x22, // MDMCFG1           Modem Configuration
126         0xF8, // MDMCFG0           Modem Configuration
127         0x00, // DEVIATN           Modem Deviation Setting
128         0x07, // MCSM2             Main Radio Control State Machine
129             Configuration
130         0x0F, // MCSM1             Main Radio Control State Machine
131             Configuration
132         0x18, // MCSM0             Main Radio Control State Machine
133             Configuration
134         0x1D, // FOCCFG           Frequency Offset Compensation
135             Configuration
136         0x1C, // BSCFG           Bit Synchronization Configuration
137         0xC7, // AGCTRL2           AGC Control
138         0x00, // AGCTRL1           AGC Control
139         0xB0, // AGCTRL0           AGC Control
140         0x87, // WOREVT1           High Byte Event0 Timeout
141         0x6B, // WOREVT0           Low Byte Event0 Timeout
142         0xF8, // WORCTRL           Wake On Radio Control
143         0xB6, // FRENDD1           Front End RX Configuration
144         0x10, // FRENDD0           Front End TX configuration
145         0xEA, // FSCAL3           Frequency Synthesizer Calibration
146         0x0A, // FSCAL2           Frequency Synthesizer Calibration
147         0x00, // FSCAL1           Frequency Synthesizer Calibration
148         0x11, // FSCAL0           Frequency Synthesizer Calibration
149         0x41, // RCCTRL1           RC Oscillator Configuration
150         0x00, // RCCTRL0           RC Oscillator Configuration
151         0x59, // FSTEST           Frequency Synthesizer Calibration
152             Control
153         0x7F, // PTEST            Production Test
154         0x3F, // AGCTEST           AGC Test
155         0x88, // TEST2            Various Test Settings
156         0x31, // TEST1            Various Test Settings
157         0x0B, // TEST0            Various Test Settings
158     };
159
160     /*****
161     SPI Functions
162     *****/
163
164     //-----
165     // void SPIconfig()
166     //
167     // DESCRIPTION:
168     //     This function is used to configure the LPC11E13 SPI0
169     //
170     // ARGUMENTS:
171     //     None

```

```

167 //-----
168
169 void spiconfig()
170 {
171     /* SSP0 Configuration */
172     PRESETCTRL = 1;           //SSP0 reset de-asserted
173     SSP0CLKDIV = 1;         //SSP0 PCLOCK = 48 MHz
174
175     PIO0_22 = 0x00000000;    //GD0 is GPIO (Input by defect)
176     PIO1_21 = 0x00000000;    //GD2 is GPIO (Input by defect)
177
178     SSP0_CRO = 0x00000107;    //8 bit transfer, CPHA=0, CPOL=0, SPI
179     frame format and SCR = 1
180     SSP0_CPSR = 4;           //Clock Rate = 48 MHz / (4*(SCR+1)) -->
181     6 MHz (supported by CC2500)
182     SSP0_CRI = (1<<1);      //Mode Master and SSP0 Module enabled
183     PIO0_8 = 0x00000001;     //PIO0_8 is MISO0
184     PIO0_9 = 0x00000001;     //PIO0_9 is MOSI0
185     SWCLK_PIO0_10 = 0x00000002; //PIO0_10 is SCK0
186     PIO1_31 = 0x00000000;    //PIO1_31 is GPIO (CSn)
187     DIR1 |= (1<<31);        //PIO1_31 is output
188     SET1 = (1<<31);         //PIO1_31 is "1" (SS inactive)
189 }
190
191 //-----
192 // unsigned int spixfer(unsigned int d)
193 //
194 // DESCRIPTION:
195 //     This function is used to transfer a data to CC2500
196 //     (It's supposed that /SS is active before calling this function)
197 // ARGUMENTS:
198 //     Data to transfer
199 //-----
200 unsigned int spixfer(unsigned char data)
201 {
202     while(!(SSP0_SR&(0x00000002))); // Wait until TX FIFO is empty
203     SSP0_DR=data;                 // Begin the transfer
204     while(SSP0_SR&(0x00000010)); // Wait until finish the transfer (
205     BSY==0)
206     return SSP0_DR;               // Return the data received (TFE
207     ==0)
208 }
209
210 //-----
211 // unsigned char strobe(unsigned char addr)
212 //
213 // DESCRIPTION:
214 //     This function is used to transfer a strobe to CC2500
215 // ARGUMENTS:
216 //     Strobe
217 //-----

```

```

216 unsigned char strobe(unsigned char addr)
217 {
218     unsigned char st;
219     CLR1 = (1<<31);           //Put SS low
220     while(WO_8);             //MCU must wait until the SO pin from
        CC2500 goes low before
221                               //starting to transfer the strobe byte.
        This indicates that
222                               //the voltage regulator has stabilized and
        the crystal is running.
223     st=spixfer(addr);        //Transfer the strobe
224     SET1 = (1<<31);         //Release SS
225     return st;              //Return Status Byte
226 }
227
228 //-----
229 // void regwr(unsigned char addr, unsigned char data)
230 //
231 // DESCRIPTION:
232 //     This function is used to write a register of CC2500
233 // ARGUMENTS:
234 //     Register address and data
235 //-----
236
237 void regwr(unsigned char addr, unsigned char data)
238 {
239     CLR1 = (1<<31);           //Put SS low
240     while(WO_8);             //MCU must wait until the SO pin from
        CC2500 goes low before
241                               //starting to transfer the strobe byte.
        This indicates that
242                               //the voltage regulator has stabilized and
        the crystal is running.
243     spixfer(addr);           //Transfer the address
244     spixfer(data);           //Transfer the data
245     SET1 = (1<<31);         //Release SS
246 }
247
248 //-----
249 // unsigned char regrd(unsigned char addr)
250 //
251 // DESCRIPTION:
252 //     This function is used to read a register of CC2500
253 // ARGUMENTS:
254 //     Register address
255 //-----
256
257 unsigned char regrd(unsigned char addr)
258 {
259     unsigned char data;
260     CLR1 = (1<<31);           //Put SS low
261     while(WO_8);             //MCU must wait until the SO pin from
        CC2500 goes low before

```



```

305         spixfer(addr|READ_BURST);           //Transfer the first address
306         while(length-- *dst++=spixfer(0); //Transfer all the data since the
           first address
307         SET1 = (1<<31);                   //Release SS
308     }
309
310 //-----
311 // void lockpll()
312 //
313 // DESCRIPTION:
314 //     This function is used to lock the PLL
315 // ARGUMENTS:
316 //     None
317 //-----
318
319 void lockpll()
320 {
321     while(regrd(FSCAL1)==0x3F);           //PLL will be locked if FSCAL1!=0x3F
322 }
323
324 //-----
325 // void ccreset()
326 //
327 // DESCRIPTION:
328 //     This function is used to initiate CC2500 after a Power On Reset
329 // ARGUMENTS:
330 //     None
331 //-----
332
333 void ccreset()
334 {
335     CLR1 = (1<<31);                       //Put SS low
336     _delay_us(1);
337     SET1 = (1<<31);                       //Release SS
338     _delay_us(40);
339     CLR1 = (1<<31);                       //Put SS low
340     while(WO_8);
341     spixfer(SRES);
342     SET1 = (1<<31);                       //Release SS
343 }
344
345 //-----
346 // void makeframe(unsigned char *buff, unsigned char *command, unsigned
           char len)
347 //
348 // DESCRIPTION:
349 //     This function is used to copy a set of parameters from one address
           to another.
350 // ARGUMENTS:
351 //     Destiny address, Source address and number of parameters
352 //-----
353

```

```

354 void makeframe(unsigned char *buff, unsigned char *command, unsigned char
    len)
355 {
356     while(len--)
357         *buff++ = *command++;
358 }
359
360 //-----
361 // void answer_to_computer(unsigned char *buff)
362 //
363 // DESCRIPTION:
364 //     This function is used to transmit a set of parameters throw UART
365 // ARGUMENTS:
366 //     Source address
367 //-----
368
369 void answer_to_computer(unsigned char *buff)
370 {
371     while(*buff != 'L')           //End of Response
372         U0putch(*buff++);
373 }
374
375 //-----
376 // void initialize_vector(unsigned char *buff, unsigned char length)
377 //
378 // DESCRIPTION:
379 //     This function is used to initialize a vector
380 // ARGUMENTS:
381 //     Vector to initialize and its length
382 //-----
383
384 void initialize_vector(unsigned char *buff, unsigned char length)
385 {
386     while(length--)
387         *buff++ = 0;
388 }
389
390 /*****
391             INTERRUPT SECTION
392 *****/
393
394 void USART_interrupt(void)
395 {
396     if(ULSR & 1)                 //We have received a byte
397     {
398         //_printf("\nEn la rutina de USART\n");
399         readcommand[num_read_bytes++] = URBR; //readcommand save the
            bytes received (Clear flag)
400         if(num_read_bytes == COMMANDLENGTH) //If we reach the length of
            the frame
401         {
402             message_OK = (readcommand[num_read_bytes - 1] == EOF) ? 1 : 0;
403             num_read_bytes = 0;

```

```

404     }
405 }
406 }
407
408 void CT16B0_interrupt (void)
409 {
410     CT16B0_IR = (1<<1);           //Clear flag for match channel
411     1
412     //_printf("\nEn la rutina Timer\n");
413     temp_expire = 1;
414     iter++;
415 }
416
417 void PIN_INT0_interrupt (void)
418 {
419     //We'll enter here when it was reached the final of a packet (IOCFG2=0
420     x01) in RX FIFO
421     RISE = 1;                     //Clear flag of rising-edge in
422     PINTSELO
423
424     CT16B0_TCR = 0x00;           //Stop Timer
425     //_printf("CT16B0_TC=%d\n", CT16B0_TC);
426     CT16B0_TC = 0;              //Reset Timer Counter
427
428     //_printf("\n\nDentro de la rutina PINTSELO\n\n");
429
430     received_packet = 1;
431 }
432
433 /*****
434 MAIN SECTION
435 *****/
436
437 int main()
438 {
439     unsigned char i, num,lqi;
440     unsigned char finished_cycle = 1;    //Inform if a cycle transmit-
441     receive has completed
442
443     disable_interrupts();
444
445     SYSAHBCLKCTRL |= (1<<6) | (1<<7) | (1<<11) | (1<<12) | (1<<16) | (1<<19); //Turn
446     on clock for GPIO, CT16B0, SSP0, USART, IOCON and GPIOPINT
447     SYSAHBCLKCTRL &= ~(1<<5);           //Turn off clock for I2C
448
449     PIO0_18 = 0x00000000;           //UART_ISP cabled is GPIO now (Error in
450     the prototype)
451     PIO0_19 = 0x00000000;
452
453     /***** GD2 is a Pin Interrupt *****/
454     PIO1_21 = 0x00000000;           //PIO1_21 (GD2) is GPIO (Input by
455     defect)
456
457

```

```

450  PINTSEL0 = 45;           //PINTSEL0 is interrupt for PIO1_21
451
452  ISEL = 0;               //PINTSEL0 is edge sensitive
453  IENR = 1;               //PINTSEL0 is rising-edge sensitive
454      enabled
455  SIENR = 1;              //PINTSEL0 is updated like in IENR
456  RISE = 1;               //Clear flag of rising-edge in PINTSEL0
457
458  ISERO = 1;              //PINTSEL0 enabled in NVIC
459  IPRO |= 0x00000000;     //PINTSEL0 is highest priority
460
461  /***** USART Configuration *****/
462  PIO1_26 = 0x00000002;   //PIO1_26 is RXD
463  PIO1_27 = 0x00000002;   //PIO1_27 is TXD
464
465  UARTCLKDIV = 1;         //UART PCLOCK = 48 MHz
466
467  ULCR = 0x83;           // 10000011 Line Control Register
468      //      .. Character length 11 = 8
469      bits
470      //      . Stop bits          0 = 1 bit
471      //      . Parity enable       0 = No
472      parity
473      //      . Even parity         non sense
474      //      . Stick parity        non sense
475      //      . Break control       0 = No
476      break
477      //      . DLAB                1 =
478      Access DLM,DLL
479
480  /* Baud Rate = 48MHz/(16*17*(1+8/15)) = 115089.5 Baudios */
481  UDLL = 17;              //8 bits LSB for Baud Rate Generator
482  UDLM = 0;               //8 bits MSB for Baud Rate Generator
483  UFDR = 0x000000F8;     //DivAddVal = 8 and MulVal = 15
484
485  ULCR = 0x03;           // [7] DLAB = 0
486
487  UFCR = 0x07;           // 00000111 FIFO Control Register
488      //      . FIFO enable        1 = TX&RX
489      FIFO on
490      //      . RX FIFO Reset      1 = Reset
491      RX FIFO
492      //      . TX FIFO Reset      1 = Reset
493      TX FIFO
494      //      .. RX int.t trigger level 00 =
495      1 byte
496
497  UIER = 1;               //Interrupt on RX
498
499  ISERO = (1<<21);        //USART0 interrupt enabled in NVIC
500
501  /***** Timer Configurator *****/
502  CT16B0_PR = 999;       //Pre-scale 1000

```

```

494
495     CT16B0_TC = 0;                //Sure TC is 0
496
497     CT16B0_MR1 = 1440;           //Timer will count 30 ms
498
499     CT16B0_MCR = (1<<3);        //Interrupt will happen when TC reach
    MR1
500
501     ISERO = (1<<16);             //CT16B0 interrupt enabled in NVIC
502
503     ////////////////////////////////// Main program //////////////////////////////////
504
505     TDI_PIO0_11 = 0x00000001;
506     DIRO |= (1<<11);
507     WO_11 = 1;
508
509     _printf("Programa main\n");
510
511     spiconfig();
512
513     ccreset();                   //When Reset is completed CC2500 is in
    IDLE State
514
515     _printf("\nMARCSTATE= %02X\n", regrd(MARCSTATE));
516
517     burstwr(0x00, RFSettings, 47); //Register Configuration for CC2500
    from SmartRF Studio
518
519     regwr(PATABLE, 0xFF);        //Output Power +1 dBm @21.5 mA
520
521     _printf("\n\nConfiguracion completa\n\n");
522
523     strobe(SFRX);                //Flush the FIFO's
524     strobe(SFTX);
525
526     strobe(SFSTXON);             // Go to FSTXON State (To quick switch
    to TX)
527
528     lockpll();                   //We are sure PLL is locked
529
530     _printf("PARTNUM= %02X\n", regrd(PARTNUM));
531     _printf("VERSION= %02X\n", regrd(VERSION));
532     _printf("MARCSTATE= %02X\n", regrd(MARCSTATE));
533
534     _printf("PKTLEN= %02X\n", regrd(PKTLEN));
535     _printf("ADDR= %02X\n", regrd(ADDR));
536     _printf("MCSM1= %02X\n", regrd(MCSM1));
537     _printf("IOCFG0= %02X\n", regrd(IOCFG0));
538
539     //If an interrupt is not enabled, asserting its interrupt signal
    changes the interrupt state to
540     //pending, but the NVIC never activates the interrupt, regardless of
    its priority.

```

```

541  RISE = 1;                                     //Clear flag of
      rising-edge in PINTSEL0
542  ICPRO = 1;                                   //Clear pending
      interrupt in PINTSEL0
543
544  enable_interrupts();
545
546  while(1)
547  {
548      if(message_OK == 1)                       //If message received is correct
549      {
550          makeframe(buf_TX, readcommand, 4);
551          makeframe(&buf_TX[4], address_master, 4);           //Copy in &
              buf_TX[4] the address of the master
552          makeframe(&buf_TX[8], &readcommand[4], 1);         //Copy in &
              buf_TX[8] the type of command
553          makeframe(&buf_TX[9], &readcommand[5], 6);         //6 Bytes of
              data
554          makeframe(&buf_TX[15], &readcommand[11], 1);       //End of Frame
555          burstwr(TXFIFO, buf_TX, regrd(PKTLEN));
556
557          //_printf("\n\nLo que se va a enviar\n\n");
558          //for (i=0;i<16;i++) U0putch(buf_TX[i]);
559
560          message_OK = 2;                         //Tristate
561          initialize_vector(readcommand, COMMANDLENGTH);
562          finished_cycle = 0;
563
564          CT16B0_TCR = 0x01;                     //Timer begins to count
565          strobe(STX);                          //Switch to RX once the message is
              sent (MCSM1)
566      }
567      else if (message_OK == 0)
568      {
569          message_OK = 2;                         //Tristate
570          initialize_vector(readcommand, COMMANDLENGTH);
571          num_read_bytes = 0;
572          finished_cycle = 1;
573          _printf("Error en la trama a enviar");
574      }
575
576      if(temp_expire)
577      {
578          if(iter == MAXITER)                   //If we rise the maximum number of
              retransmissions
579          {
580              answer_to_computer((unsigned char*) "ErrorL");
581              CT16B0_TCR = 0x00;                 //Stop the Timer
582              CT16B0_TC = 0;                   //TC equals to 0
583              iter = 0;                         //Restart the number of
              retransmissions
584              finished_cycle = 1;
585          }

```

```

586     else
587     {
588         //_printf("\n\nTemporizador expirado\n\n");
589         burstwr(TXFIFO, buf_TX, regrd(PKTLEN)); //Transmit
           again the message
590         CT16B0_TC = 0;
591         strobe(STX); //Switch to RX once the message is
           sent (MCSM1)
592     }
593     temp_expire = 0;
594 }
595
596 if(received_packet)
597 {
598     num=regrd(RXBYTES)&0x7F; //Read NUM_RXBYTES
599
600     //_printf("BYTES EN RX FIFO=%d\n", num);
601
602     if(num>0) //Indicates CRC is OK and data can
           be read from RX FIFO
603     {
604         burstrd(RXFIFO, buf_RX, num);
605         iter = 0; //Restart the timer
606
607         //_printf("MENSAJE RECIBIDO\n");
608         //for (i=0;i<num;i++) U0putch(buf_RX[i]);
609
610         //_printf("\nMENSAJE PASADO A PC\n");
611
612         answer_to_computer(&buf_RX[4]); //Answer to PC
613
614         finished_cycle = 1;
615
616         i=(signed char)buf_RX[num-2]; //RSSI
617         //_printf("\nI=%x\n", i);
618         if(i>=128)
619             i=(i-256)/2-72;
620         else
621             i=i/2-72;
622
623         if(i&0x80)
624         {
625             i=-i;
626             _printf("\nRSSI = -%d dBm\n", i);
627         }
628         else
629             _printf("\nRSSI = +%d dBm\n", i);
630
631         _printf("LQI = %02X\n", buf_RX[num-1]&0x7F);
632     }
633     else
634     {

```

```
635     if(iter == MAXITER)           //If we rise the maximum number
        of retransmissions
636     {
637         answer_to_computer((unsigned char*)"ErrorL");
638         CT16B0_TCR = 0x00;       //Stop the Timer
639         CT16B0_TC = 0;          //TC equals to 0
640         iter = 0;               //Restart the number of
        retransmissions
641         finished_cycle = 1;
642     }
643     else
644     {
645         if(!finished_cycle)      //Only retransmit if a cycle
        has not finished
646         {
647             //Prevent of a glitch detected
        in PINTSEL0 (received packet)
        //_printf("\n\nCRC incorrecto, Retransmito de nuevo
        \n\n");
648             burstwr(TXFIFO, buf_TX, regd(PKTLLEN));
649
650             CT16B0_TCR = 0x01;    //Restart Timer
651             strobe(STX);          //Switch to RX once the message
        is sent (MCSM1)
652             iter++;
653         }
654     }
655     received_packet = 0;
656 }
657 }
658 }
659 }
```

Código D.2: Programa principal del módulo maestro

# Apéndice E

## Código del módulo esclavo

A continuación se lista el código correspondiente a los módulos de radio esclavos.

```
1  /*****
2  * @file      init.c
3  * @brief     Interrupts Vector Table and Clock and PLL initialization
4  * @version   V2.00
5  * @date      17. June 2015
6  *****/
7
8  #include "lpc11E13.h"
9
10 void init(void);
11 //void NMI_interrupt(void);
12 //void Hard_Fault_interrupt(void);
13 //void SVCcall_interrupt(void);
14 //void PendSV_interrupt(void);
15 //void SysTic_interrupt(void);
16 void PIN_INT0_interrupt(void);
17 //void PIN_INT1_interrupt(void);
18 //void PIN_INT2_interrupt(void);
19 //void PIN_INT3_interrupt(void);
20 //void PIN_INT4_interrupt(void);
21 //void PIN_INT5_interrupt(void);
22 //void PIN_INT6_interrupt(void);
23 //void PIN_INT7_interrupt(void);
24 //void GINT0_interrupt(void);
25 //void GINT1_interrupt(void);
26 void Default_Handler(void);
27 //void SSP1_interrupt(void);
28 //void I2C_interrupt(void);
29 void CT16B0_interrupt(void);
30 void CT16B1_interrupt(void);
31 //void CT32B0_interrupt(void);
32 //void CT32B1_interrupt(void);
33 //void SSP0_interrupt(void);
34 //void USART_interrupt(void);
```

```

35 //void ADC_interrupt(void);
36 //void WWDT_interrupt(void);
37 //void BOD_interrupt(void);
38 //void FLASH_EEPROM_interrupt(void);
39 //void IOH_interrupt(void);
40
41 // The following are 'declared' in the linker script
42 extern unsigned char INIT_DATA_VALUES;
43 extern unsigned char INIT_DATA_START;
44 extern unsigned char INIT_DATA_END;
45 extern unsigned char BSS_START;
46 extern unsigned char BSS_END;
47
48 // The section "vectors" is placed at the beginning of flash by the linker
  script
49 const void * Vectors[] __attribute__((section(".vectors"))) =
50 {
51     (void *)0x10002000,    /* Top of stack */
52     init,                 /* Reset Handler */
53     Default_Handler,     /*NMI_interrupt,        /* NMI */
54     Default_Handler,     /*Hard_Fault_interrupt, /* Hard Fault */
55     0,                    /* Reserved */
56     0,                    /* Reserved */
57     0,                    /* Reserved */
58     0,                    /* Reserved */
59     0,                    /* Reserved */
60     0,                    /* Reserved */
61     0,                    /* Reserved */
62     Default_Handler,     /*SVCcall_interrupt,    /* SVC */
63     0,                    /* Reserved */
64     0,                    /* Reserved */
65     Default_Handler,     /*PendSV_interrupt,     /* PendSV */
66     Default_Handler,     /*SysTic_interrupt,     /* SysTick */
67
68 /* External interrupt handlers follow */
69
70     PIN_INT0_interrupt,   /* GPIO pin interrupt 0 (0) */
71     Default_Handler,     /* GPIO pin interrupt 1 (1) */
72     Default_Handler,     /*PIN_INT2_interrupt,    /* GPIO pin
       interrupt 2 (2) */
73     Default_Handler,     /*PIN_INT3_interrupt,    /* GPIO pin
       interrupt 3 (3) */
74     Default_Handler,     /*PIN_INT4_interrupt,    /* GPIO pin
       interrupt 4 (4) */
75     Default_Handler,     /*PIN_INT5_interrupt,    /* GPIO pin
       interrupt 5 (5) */
76     Default_Handler,     /*PIN_INT6_interrupt,    /* GPIO pin
       interrupt 6 (6) */
77     Default_Handler,     /*PIN_INT7_interrupt,    /* GPIO pin
       interrupt 7 (7) */
78     Default_Handler,     /*GINT0_interrupt,       /* GPIO GROUP0
       interrupt (8) */
79     Default_Handler,     /*GINT1_interrupt,       /* GPIO GROUP0

```

```

    interrupt (9) */
80 Default_Handler,      /* RESERVED (10) */
81 Default_Handler,      /* RESERVED (11) */
82 Default_Handler,      /* RESERVED (12) */
83 Default_Handler,      /* RESERVED (13) */
84 Default_Handler,      //SSP1_interrupt,      /* SSP1 interrupt
    (14) */
85 Default_Handler,      //I2C_interrupt,      /* I2C interrupt
    (15) */
86 CT16B0_interrupt,     /* CT16B0 interrupt (16) */
87 CT16B1_interrupt,     /* CT16B1 interrupt (17) */
88 Default_Handler,      //CT32B0_interrupt,     /* CT32B0 interrupt
    (18) */
89 Default_Handler,      //CT32B1_interrupt,     /* CT32B1 interrupt
    (19) */
90 Default_Handler,      //SSP0_interrupt,      /* SSP0 interrupt
    (20) */
91 Default_Handler,      //USART_interrupt     /* UART interrupt
    (21) */
92 Default_Handler,      /* RESERVED (22) */
93 Default_Handler,      /* RESERVED (23) */
94 Default_Handler,      //ADC_interrupt,      /* ADC interrupt
    (24) */
95 Default_Handler,      //WWDT_interrupt,     /* WDT interrupt
    (25) */
96 Default_Handler,      //BOD_interrupt,      /* BOD interrupt
    (26) */
97 Default_Handler,      //FLASH_EEPROM_interrupt, /* Flash/EEPROM
    interrupt(27) */
98 Default_Handler,      /* RESERVED (28) */
99 Default_Handler,      /* RESERVED (29) */
100 Default_Handler,     /* RESERVED (30) */
101 Default_Handler,     //IOH_interrupt      /* I/O Handler
    interrupt (31) */
102 };
103
104 extern inline clock_init()
105 {
106     // This function sets the main clock to the PLL
107     // The PLL input is the built in 12MHz RC oscillator
108     // This is multiplied up to 48MHz for the main clock (MSEL = 3, PSEL =
109     // 1)
110     SYSPLLCLKSEL = 0;          // select internal RC oscillator
111     SYSPLLCTRL = 3 | (1 << 5); // set divisors/multipliers
112     MAINCLKSEL = 3;          // Use PLL output as main clock
113     PDRUNCFG &= ~(1<<7);    // Power up the PLL. This MUST be done
114     // before update ~xxUEN registers
115     PDRUNCFG &= ~(1<<4);    // Power up the ADC. This MUST be done
116     // before update ~xxUEN registers
117     SYSPLLCLKUEN = 0;        // inform PLL of update
118     SYSPLLCLKUEN = 1;
119     MAINCLKUEN = 0;         // Inform core of clock update

```

```

118     MAINCLKUEN = 1;
119     while (!(SYSPLLSTAT & 1));    //Wait until PLL is locked
120 }
121
122 void init()
123 {
124     // do global/static data initialization
125     unsigned char *src;
126     unsigned char *dest;
127     unsigned len;
128     src= &INIT_DATA_VALUES;
129     dest= &INIT_DATA_START;
130     len= &INIT_DATA_END-&INIT_DATA_START;
131     while (len--)
132         *dest++ = *src++;
133
134     // zero out the uninitialized global/static variables
135     dest = &BSS_START;
136     len = &BSS_END - &BSS_START;
137     while (len--)
138         *dest++=0;
139
140     clock_init();
141     main();
142 }
143
144 void Default_Handler()
145 {
146     _printf("\nDefault Handler. -Halt-\n");
147     while(1);
148 }

```

Código E.1: Fichero de inicialización de los módulos esclavos

```

1  /*****
2  *                               TRABAJO FIN DE GRADO                               *
3  *****/
4  *
5  *   Desarrollo de un Sistema de Comunicacion Digital via Radio   *
6  *                               para Entornos Industriales                               *
7  *
8  *****/
9  * @author:      Javier Santos Romo
10 *****/
11 * @brief:      Main program and subfunctions
12 *****/
13 * Revision history:      17/06/2015
14 *****/
15
16 /*****
17                               Header files

```

```

18  *****/
19
20  #include "lpc11E13.h"
21  #include "cc2500.h"
22
23  /******
24          Clocks and Interrupts
25  *****/
26
27  #define enable_interrupts() asm("cpsie i")
28  #define disable_interrupts() asm("cpsid i")
29  #define sleep_mode() asm("wfi")
30
31  #define PCLK 48000000
32  #define _delay_us(n) delay_loop((n*(PCLK/1000)-6000)/4000)
33  #define _delay_ms(n) delay_loop((n*(PCLK/1000)-6)/4)
34
35  void delay_loop(unsigned int) __attribute__((naked));
36  void delay_loop(unsigned int d)
37  {
38      asm volatile ("Ldelay_loop: sub r0,#1");
39      asm volatile ("bne Ldelay_loop");
40      asm volatile ("bx lr");
41  }
42
43  //-----
44  // int U0putch(int data)
45  //
46  // DESCRIPTION:
47  //     This function is used to transmit a data through UART
48  //
49  // ARGUMENTS:
50  //     Data to transfer
51  //-----
52
53  int U0putch(int data)
54  {
55      while(!(ULSR & 0x00000040)); //Wait until TEMPTY is empty
56      UTHR = data; //Transmit data
57  }
58
59  //-----
60  // unsigned char U0getch()
61  //
62  // DESCRIPTION:
63  //     This function is used to get a data from UART
64  //
65  // ARGUMENTS:
66  //     None
67  //-----
68
69  unsigned char U0getch()
70  {

```

```

71     while(!(ULSR & 0x00000001)); //Wait until RBR is empty
72     return URBR; //Return data
73 }
74
75 #define putchar(c) U0putch(c)
76 #include "printf.c"
77
78 /*****
79         Definitions
80 *****/
81
82 #define MESSAGELENGTH 16
83
84 unsigned char buf_TX[64]={0};
85 unsigned char buf_RX[64]={0};
86
87 const unsigned char address_slave[4] __attribute__((section(".vectors"))) =
88     "SgCr";
89
90 volatile unsigned char chip_select = 0; //Chip is selected if user
91     has pushed PIO0_1 low
92
93 volatile unsigned char wake_up = 0; //Inform when radio is woke
94     up
95
96 typedef enum Commands {READPINS = '1', BROADCAST, SETPIN, RESETPIN,
97     TEMPERATURE_SENSOR, RECOVERCONFIG} Commands;
98
99 Commands commands;
100
101 const unsigned char RFSettings[] __attribute__((section(".vectors"))) ={
102     0x01, // IOCFG2 GDO2Output Pin Configuration
103     0x2E, // IOCFG1 GDO1Output Pin Configuration
104     0x80, // IOCFG0 GDO0Output Pin Configuration
105     0x07, // FIFOTHR RX FIFO and TX FIFO Thresholds
106     0xD3, // SYNC1 Sync Word, High Byte
107     0x91, // SYNC0 Sync Word, Low Byte
108     0x10, // PKTLEN Packet Length (16 bits)
109     0x0E, // PKTCTRL1 Packet Automation Control
110     0x44, // PKTCTRL0 Packet Automation Control
111     0x53, // ADDR Device Address
112     0x00, // CHANNR Channel Number
113     0x0A, // FSCTRL1 Frequency Synthesizer Control
114     0x00, // FSCTRL0 Frequency Synthesizer Control
115     0x5D, // FREQ2 Frequency Control Word, High Byte
116     0x93, // FREQ1 Frequency Control Word, Middle Byte
117     0xB1, // FREQ0 Frequency Control Word, Low Byte
118     0x2D, // MDMCFG4 Modem Configuration
119     0x3B, // MDMCFG3 Modem Configuration
120     0x73, // MDMCFG2 Modem Configuration
121     0x22, // MDMCFG1 Modem Configuration
122     0xF8, // MDMCFG0 Modem Configuration
123     0x00, // DEVIATN Modem Deviation Setting

```

```

120     0x07, // MCSM2           Main Radio Control State Machine
        Configuration
121     0x0F, // MCSM1           Main Radio Control State Machine
        Configuration
122     0x18, // MCSM0           Main Radio Control State Machine
        Configuration
123     0x1D, // FOCCFG         Frequency Offset Compensation
        Configuration
124     0x1C, // BSCFG           Bit Synchronization Configuration
125     0xC7, // AGCCTRL2       AGC Control
126     0x00, // AGCCTRL1       AGC Control
127     0xB0, // AGCCTRL0       AGC Control
128     0x87, // WOREVT1        High Byte Event0 Timeout
129     0x6B, // WOREVT0        Low Byte Event0 Timeout
130     0xF8, // WORCTRL         Wake On Radio Control
131     0xB6, // FREND1         Front End RX Configuration
132     0x10, // FREND0         Front End TX configuration
133     0xEA, // FSCAL3         Frequency Synthesizer Calibration
134     0x0A, // FSCAL2         Frequency Synthesizer Calibration
135     0x00, // FSCAL1         Frequency Synthesizer Calibration
136     0x11, // FSCAL0         Frequency Synthesizer Calibration
137     0x41, // RCCTRL1        RC Oscillator Configuration
138     0x00, // RCCTRL0        RC Oscillator Configuration
139     0x59, // FSTEST         Frequency Synthesizer Calibration
        Control
140     0x7F, // PTEST           Production Test
141     0x3F, // AGCTEST         AGC Test
142     0x88, // TEST2           Various Test Settings
143     0x31, // TEST1           Various Test Settings
144     0x0B, // TEST0           Various Test Settings
145                                     };
146
147  /*****
148                                     SPI Functions
149  *****/
150
151  //-----
152  // void SPIconfig()
153  //
154  // DESCRIPTION:
155  //     This function is used to configure the LPC11E13 SPI0
156  //
157  // ARGUMENTS:
158  //     None
159  //-----
160
161  void spiconfig()
162  {
163      /* SSP0 Configuration */
164      PRESETCTRL = 1;           //SSP0 reset de-asserted
165      SSPOCLKDIV = 1;          //SSP0 PCLOCK = 48 MHz
166
167      PIO0_22 = 0x00000000;    //GD0 is GPIO (Input by defect)

```

```

168     PIO1_21 = 0x00000000;           //GD2 is GPIO (Input by defect)
169
170     SSP0_CRO = 0x00000107;         //8 bit transfer, CPHA=0, CPOL=0, SPI
        frame format and SCR = 1
171     SSP0_CPSR = 4;                 //Clock Rate = 48 MHz / (4*(SCR+1)) -->
        6 MHz (supported by CC2500)
172     SSP0_CR1 = (1<<1);            //Mode Master and SSP0 Module enabled
173     PIO0_8 = 0x00000001;          //PIO0_8 is MIS00
174     PIO0_9 = 0x00000001;          //PIO0_9 is MOSI0
175     SWCLK_PIO0_10 = 0x00000002;    //PIO0_10 is SCK0
176     PIO1_31 = 0x00000000;         //PIO1_31 is GPIO (CSn)
177     DIR1 |= (1<<31);              //PIO1_31 is output
178     SET1 = (1<<31);               //PIO1_31 is "1" (SS inactive)
179 }
180
181 //-----
182 // unsigned int spixfer(unsigned int d)
183 //
184 // DESCRIPTION:
185 //     This function is used to transfer a data to CC2500
186 //     (It's supposed that /SS is active before calling this function)
187 // ARGUMENTS:
188 //     Data to transfer
189 //-----
190
191 unsigned int spixfer(unsigned char data)
192 {
193     while(!(SSP0_SR&(0x00000002))); // Wait until TX FIFO is empty
194     SSP0_DR=data;                  // Begin the transfer
195     while(SSP0_SR&(0x00000010)); // Wait until finish the transfer (
        BSY==0)
196     return SSP0_DR;              // Return the data received (TFE
        ==0)
197 }
198
199 //-----
200 // unsigned char strobe(unsigned char addr)
201 //
202 // DESCRIPTION:
203 //     This function is used to transfer a strobe to CC2500
204 // ARGUMENTS:
205 //     Strobe
206 //-----
207
208 unsigned char strobe(unsigned char addr)
209 {
210     unsigned char st;
211     CLR1 = (1<<31);               //Put SS low
212     while(WO_8);                  //MCU must wait until the SO pin from CC2500
        goes low before
213                                     //starting to transfer the strobe byte. This
        indicates that

```

```

214                                     //the voltage regulator has stabilized and the
                                     crystal is running.
215     st=spixfer(addr);                 //Transfer the strobe
216     SET1 = (1<<31);                 //Release SS
217     return st;                       //Return Status Byte
218 }
219
220 //-----
221 // void regwr(unsigned char addr, unsigned char data)
222 //
223 // DESCRIPTION:
224 //     This function is used to write a register of CC2500
225 // ARGUMENTS:
226 //     Register address and data
227 //-----
228
229 void regwr(unsigned char addr, unsigned char data)
230 {
231     CLR1 = (1<<31);                 //Put SS low
232     while(WO_8);                   //MCU must wait until the SO pin from CC2500 goes
                                     low before
233                                     //starting to transfer the strobe byte. This
                                     indicates that
234                                     //the voltage regulator has stabilized and the
                                     crystal is running.
235     spixfer(addr);                 //Transfer the address
236     spixfer(data);                 //Transfer the data
237     SET1 = (1<<31);                 //Release SS
238 }
239
240 //-----
241 // unsigned char regrd(unsigned char addr)
242 //
243 // DESCRIPTION:
244 //     This function is used to read a register of CC2500
245 // ARGUMENTS:
246 //     Register address
247 //-----
248
249 unsigned char regrd(unsigned char addr)
250 {
251     unsigned char data;
252     CLR1 = (1<<31);                 //Put SS low
253     while(WO_8);                   //MCU must wait until the SO pin from
                                     CC2500 goes low before
254                                     //starting to transfer the strobe byte.
                                     This indicates that
255                                     //the voltage regulator has stabilized
                                     and the crystal is running.
256     spixfer(addr|READ_SINGLE);     //Transfer the address
257     data=spixfer(0);               //CC2500 transfer the data
258     SET1 = (1<<31);                 //Release SS
259     return data;                   //Return Register

```

```
260 }
261
262 //-----
263 // void burstwr(unsigned char addr,unsigned char *src,unsigned char length
264 // )
265 // DESCRIPTION:
266 // This function is used to write a set of parameters in burst mode
267 // ARGUMENTS:
268 // Start address, Set of data and number of data
269 //-----
270
271 void burstwr(unsigned char addr,unsigned char *src,unsigned char length)
272 {
273     CLR1 = (1<<31); //Put SS low
274     while(WO_8); //MCU must wait until the SO pin
275     //starting to transfer the strobe
276     //byte. This indicates that
277     //the voltage regulator has
278     //stabilized and the crystal is
279     //running.
280     spixfer(addr|WRITE_BURST); //Transfer the first address
281     while(length--) spixfer(*src++); //Transfer all the data since the
282     //first address
283     SET1 = (1<<31); //Release SS
284 }
285
286 //-----
287 // void burstrd(unsigned char addr,unsigned char *dst,unsigned char length
288 // )
289 // DESCRIPTION:
290 // This function is used to read a set of parameters in burst mode
291 // ARGUMENTS:
292 // Start address, Set of address and number of data
293 //-----
294
295 void burstrd(unsigned char addr,unsigned char *dst,unsigned char length)
296 {
297     CLR1 = (1<<31); //Put SS low
298     while(WO_8); //MCU must wait until the SO pin
299     //starting to transfer the strobe
300     //byte. This indicates that
301     //the voltage regulator has
302     //stabilized and the crystal is
303     //running.
304     spixfer(addr|READ_BURST); //Transfer the first address
305     while(length--) *dst++=spixfer(0); //Transfer all the data since the
306     //first address
307     SET1 = (1<<31); //Release SS
308 }
```

```

301
302 //-----
303 // void lockpll()
304 //
305 // DESCRIPTION:
306 //     This function is used to lock the PLL
307 // ARGUMENTS:
308 //     None
309 //-----
310
311 void lockpll()
312 {
313     while(regrd(FSCAL1)==0x3F);    //PLL will be locked if FSCAL1!=0x3F
314 }
315
316 //-----
317 // void ccreset()
318 //
319 // DESCRIPTION:
320 //     This function is used to iniciate CC2500 after a Power On Reset
321 // ARGUMENTS:
322 //     None
323 //-----
324
325 void ccreset()
326 {
327     CLR1 = (1<<31);    //Put SS low
328     _delay_us(1);
329     SET1 = (1<<31);    //Release SS
330     _delay_us(40);
331     CLR1 = (1<<31);    //Put SS low
332     while(WO_8);
333     spixfer(SRES);
334     SET1 = (1<<31);    //Release SS
335 }
336
337 //-----
338 // void makeframe(unsigned char *buff, unsigned char *command, unsigned
339 // char len)
340 //
341 // DESCRIPTION:
342 //     This function is used to copy a set of parameters from one address
343 //     to another.
344 // ARGUMENTS:
345 //     Destiny address, Source address and number of parameters
346 //-----
347
348 void makeframe(unsigned char *buff, unsigned char *command, unsigned char
349 len)
350 {
351     while(len--)
352         *buff++ = *command++;
353 }

```

```
351
352 //-----
353 // void initialize_vector(unsigned char *buff, unsigned char length)
354 //
355 // DESCRIPTION:
356 //     This function is used to initialize a vector
357 // ARGUMENTS:
358 //     Vector to initialize and its length
359 //-----
360
361 void initialize_vector(unsigned char *buff, unsigned char length)
362 {
363     while(length--)
364         *buff++ = 0;
365 }
366
367 //-----
368 // unsigned char check_address(unsigned char *slave, unsigned char *
369 // received, unsigned char length)
370 //
371 // DESCRIPTION:
372 //     This function is used to check if an received address is correct
373 // ARGUMENTS:
374 //     Address of slave, address received and length of the address
375 //-----
376
377 unsigned char check_address(unsigned char *slave, unsigned char *received,
378 unsigned char length)
379 {
380     unsigned char length2 = length, OK_1 = 1, OK_2 = 1;
381     unsigned char *received2 = received;
382
383     while(length--      //While loop for checking hardcoded address
384     {
385         if(*slave++ != *received++)    //If address isn't equal
386         {
387             OK_1 = 0;
388             break;
389         }
390     }
391
392     while(length2--      //While loop for checking broadcast address
393     {
394         if(*received2++)    //If address isn't equal
395         {
396             OK_2 = 0;
397             break;
398         }
399     }
400
401     if(OK_1 || OK_2)
402         return 1;    //If address is correct returns 1
403     else
```

```

402         return 0;
403     }
404
405     //-----
406     // unsigned int read_sensor()
407     //
408     // DESCRIPTION:
409     //     This function is used to read the measure of the analog temperature
410     //     sensor
411     // ARGUMENTS:
412     //     None
413     //-----
414     unsigned int read_sensor()
415     {
416         unsigned int sensor_value;
417         ADCCR |= (1<<24);           //Start the conversion
418         while (!(ADCDR6 & (1<<31))); //When bit 31 (DONE) is set, conversion
419             sensor_value = ((ADCDR6 >> 6) & 0x000003FF); //Store the value captured
420             by the sensor (10 bit resolution)
421             ADCCR &= ~(1<<24);           //Stop the conversion
422
423             // V_sensor = 2.43mV/°C * Temperature_sensor + 750mV
424             // V_sensor = 3.3V / 1024 * sensor_value
425             return sensor_value;
426     }
427     /*****
428             INTERRUPT SECTION
429     *****/
430
431     void PIN_INT0_interrupt (void)
432     {
433         //We'll enter here when PIO0_1 is pushed LOW
434         FALL = 1;           //PINTSEL0 falling-edge is cleared
435
436         _printf("\n\nDentro de la rutina PINTSEL\n\n");
437
438         chip_select = 1;
439         CT16B0_TCR = 0x01; //Timer begins to count
440     }
441
442     void CT16B0_interrupt (void)
443     {
444         CT16B0_IR = (1<<1); //Clear flag for match channel 1
445         CT16B0_TCR = 0x00; //Stop Timer
446         CT16B0_TC = 0; //Reset Timer
447
448         _printf("\n\nEn la rutina Timer\n\n");
449
450         chip_select = 0;
451     }

```

```

452
453 void CT16B1_interrupt(void)
454 {
455     CT16B1_IR = (1<<1);           //Clear flag for match channel 1
456     CT16B1_TCR = 0x00;
457     CT16B1_TC = 0;               //Reset Timer
458
459     if(wake_up)                  //If radio is woke up
460     {
461         CT16B1_MR1 = 1440;       //300ms asleep
462         strobe(SIDLE);           //Radio goes sleep from IDLE
463         strobe(SFRX);           //Flush the FIFO's
464         strobe(SFTX);
465         strobe(SPWD);           //Radio goes sleep
466     }
467     else
468     {
469         CT16B1_MR1 = 480;        //100ms woke up
470         strobe(SRX);            //Radio wakes up and goes RX mode
471     }
472     wake_up = !wake_up;         //Invert the status
473
474     CT16B1_TCR = 0x01;         //Timer begins
475 }
476
477 /*****
478     MAIN SECTION
479 *****/
480
481 int main (void)
482 {
483     unsigned char i,num,pin;
484     unsigned char port = 0x00;   //8 bits for the pins
485     unsigned char* ptr_port = &port;
486     unsigned char temperature[3]={0};
487
488     //address_slave[1] = 0x00;
489
490     disable_interrups();
491
492     SYSAHBCLKCTRL |= (1<<6) | (1<<7) | (1<<8) | (1<<11) | (1<<12) | (1<<13) | (1<<16)
493         | (1<<19); //Turn on clock for GPIO, CT16B0, CT16B1, SSP0, USART,
494         ADC, IOCON and GPIOINT
495     SYSAHBCLKCTRL &= ~(1<<5);    //Turn off clock for I2C
496
497     PIO0_18 = 0x00000000;        //UART_ISP cabled is GPIO now (Error in
498         the prototype)
499     PIO0_19 = 0x00000000;
500
501     /***** PIO0_1 is a Pin Interrupt *****/
502     //PIO0_1 (ISP) is GPIO (Input by defect with pull-up enabled)
503
504     PINTSELO = 1;                //PINTSELO is interrupt for PIO0_1

```

```

502
503 ISEL = 0;           //PINTSELO is edge sensitive
504 IENF = 1;         //PINTSELO is falling-edge sensitive enabled
505 SIENF = 1;        //PINTSELO is updated like in IENF
506 FALL = 1;         //PINTSELO falling-edge is cleared
507
508 ISERO = 1;        //PINTSELO enabled in NVIC
509 //IPRO |= 0x00000000; //PINTSELO is highest priority
510
511 /***** USART Configuration *****/
512 PIO1_26 = 0x00000002; //PIO1_26 is RXD
513 PIO1_27 = 0x00000002; //PIO1_27 is TXD
514
515 UARTCLKDIV = 1;    //UART PCLOCK = 48 MHz
516
517 ULCR = 0x83;      // 10000011 Line Control Register
518 //      .. Character length 11 = 8 bits
519 //      . Stop bits          0 = 1 bit
520 //      . Parity enable      0 = No parity
521 //      . Even parity        non sense
522 //      . Stick parity       non sense
523 //      . Break control      0 = No break
524 //      . DLAB               1 = Access DLM,
525 //                          DLL
526
527 /* Baud Rate = 48MHz/(16*17*(1+8/15)) = 115089.5 Baudios */
528 UDLL = 17;        //8 bits LSB for Baud Rate Generator
529 UDLM = 0;         //8 bits MSB for Baud Rate Generator
530 UFDR = 0x000000F8; //DivAddVal = 8 and MulVal = 15
531
532 ULCR = 0x03;      // [7] DLAB = 0
533
534 UFCR = 0x07;      // 00000111 FIFO Control Register
535 //      . FIFO enable       1 = TX&RX FIFO on
536 //      . RX FIFO Reset     1 = Reset RX FIFO
537 //      . TX FIFO Reset     1 = Reset TX FIFO
538 //      .. RX int.t trigger level 00 = 1 byte
539
540 /***** GPIO Inputs and Outputs Configurator
541 *****/
542 PIO1_14 = 0x00000000; //PIO1_14 is GPIO
543 PIO1_22 = 0x00000000; //PIO1_22 is GPIO
544 SWDIO_PIO0_15 = 0x000000081; //PIO0_15 is GPIO and Digital Mode
545 PIO0_16 = 0x000000080; //PIO0_16 is GPIO and Digital Mode
546 DIRO |= (1<<15) | (1<<16); //Pins as outputs
547 DIR1 |= (1<<14) | (1<<22);
548
549 PIO0_17 = 0x00000000; //PIO0_17 is GPIO
550 PIO0_18 = 0x00000000; //PIO0_18 is GPIO
551 PIO0_19 = 0x00000000; //PIO0_19 is GPIO
552 PIO1_16 = 0x00000000; //PIO1_16 is GPIO
553 DIRO &~ (1<<17); //Pins as inputs
554 DIRO &~ (1<<18);

```

```

553 DIR0 &=~ (1<<19);
554 DIR1 &=~ (1<<16);
555
556 DIR0|= (1<<3); //salida PIO0_3 solamente para test
557
558 /***** Timer 0 Configurator *****/
559 CT16B0_PR = 14999; //Pre-escape 15000
560
561 CT16B0_TC = 0; //Sure TC is 0
562
563 CT16B0_MR1 = 48000; //Timer will count 15 s
564
565 CT16B0_MCR = (1<<3); //Interrupt will happen when TC reach MR1
566
567 ISER0 = (1<<16); //CT16B0 interrupt enabled in NVIC
568
569 /***** Timer 1 Configurator *****/
570 CT16B1_PR = 9999; //Pre-escape 10000
571
572 CT16B1_TC = 0; //Sure TC is 0
573
574 CT16B1_MR1 = 1440; //Timer will count 300ms
575
576 CT16B1_MCR = (1<<3); //Interrupt will happen when TC reach MR1
577
578 ISER0 = (1<<17); //CT16B0 interrupt enabled in NVIC
579
580 /***** ADC Configurator *****/
581 PIO0_22 = 0x00000001; //GD0 is input of analog temperature sensor (
    AD6)
582 ADCCR = 0x00000B40; //ADC->4MHz, BURST->0, START->0, AD6
583
584 //////////////// Main program //////////////////////
585
586 TDI_PIO0_11 = 0x00000001;
587 DIR0 |= (1<<11);
588 W0_11 = 1;
589
590 _printf("\n\nPrograma main\n");
591
592 spiconfig();
593
594 ccreset(); //When Reset is completed CC2500 is in IDLE State
595
596 _printf("\nMARCSTATE=%02X\n", regrd(MARCSTATE));
597
598 burstwr(0x00, RFSettings, 47); //Register Configuration for CC2500
    from SmartRF Studio
599
600 regwr(PATABLE, 0xFF); //Output Power +1 dBm @21.5 mA
601
602 _printf("\n\nConfiguracion completa\n\n");
603

```

```

604     strobe(SFRX);           //Flush the FIFO's
605     strobe(SFTX);
606
607     strobe(SPWD);           //Radio goes sleep mode
608
609     //If an interrupt is not enabled, asserting its interrupt signal
        changes the interrupt state to
610     //pending, but the NVIC never activates the interrupt, regardless of
        its priority.
611     FALL = 1;               //PINTSELO falling-edge is cleared
612     ICPRO = 1;              //Clear pending interrupt in PINTSELO
613     RISE |= (1<<1);         //PINTSEL1 rising-edge is cleared
614     ICPRO = (1<<1);         //Clear pending interrupt in PINTSEL1
615
616     enable_interrupts();
617
618     /***** Sleep Mode Configurator *****/
619     PCON = 0x00000000;      //Sleep mode indicated to enter
620     SCR &=~ (1<<2);         //SLEEPDEEP bit introduced
621
622     CT16B1_TCR = 0x01;      //Timer begins
623     sleep_mode();           //Sleep mode entered of MCU
624
625     while(1)
626     {
627         while(wake_up)
628         {
629             if(W1_21) //If received packet
630             {
631                 disable_interrupts();
632
633                 _printf("\n\nMARCSTATE=%02X\n", regrd(MARCSTATE));
634
635                 num=regrd(RXBYTES) & 0x7F; //Read NUM_RXBYTES
636
637                 _printf("BYTES EN RX FIFO=%d\n", num);
638
639                 if(num>0) //Indicate that CRC was OK and data can be read
                    from RX FIFO
640                 {
641                     burstrd(RXFIFO, buf_RX, num);
642
643                     if(check_address(address_slave, buf_RX, 4)) //If address
                        of slave or broadcast address received
644                     {
645                         _printf("MENSAJE RECIBIDO\n");
646                         for (i=0; i<num; i++) U0putch(buf_RX[i]);
647
648                         commands = buf_RX[8];
649
650                         _printf("\nMENSAJE ENVIADO\n");
651
652                         switch((char) commands) //Type of command

```

```

653     {
654         case READPINS:          // I/O Command
655             port = (B1_16 & 0x01); //Bit 0 is PIO1_16
656             port|= ((B0_19 & 0x01)<<1); //Bit 1 is
657                 PIO0_19
658             port|= ((B0_18 & 0x01)<<2); //Bit 2 is
659                 PIO0_18
660             port|= ((B0_17 & 0x01)<<3); //Bit 3 is
661                 PIO0_17
662             port|= ((B0_16 & 0x01)<<4); //Bit 4 is
663                 PIO0_16
664             port|= ((B0_15 & 0x01)<<5); //Bit 5 is
665                 PIO0_15
666             port|= ((B1_22 & 0x01)<<6); //Bit 6 is
667                 PIO1_22
668             port|= ((B1_14 & 0x01)<<7); //Bit 7 is
669                 PIO1_14
670
671             makeframe(buf_TX, &buf_RX[4], 4); //Copy in &
672                 buf_TX[0] the address of the master
673             makeframe(&buf_TX[4], address_slave, 4); //
674                 Copy in &buf_TX[4] the address of the
675                 slave
676             makeframe(&buf_TX[8], &buf_RX[8], 1); //Copy
677                 in &buf_TX[8] the type of command
678             makeframe(&buf_TX[14], ptr_port, 1); //Copy
679                 in &buf_TX[14] port
680             makeframe(&buf_TX[15], &buf_RX[15], 1); //End
681                 of Frame
682             burstwr(TXFIFO, buf_TX, regrd(PKTLLEN));
683
684             strobe(STX); //Switch to RX once the
685                 message is sent (MCSM1)
686             for (i=0; i<num; i++) U0putch(buf_TX[i]);
687             break;
688
689         case BROADCAST: // Address Command
690             if(chip_select) //Only if PIO0_1 was pushed
691                 LOW the Slave answers
692             {
693                 makeframe(buf_TX, &buf_RX[4], 4); //Copy
694                     in &buf_TX[0] the address of the
695                     master
696                 makeframe(&buf_TX[4], address_slave, 4);
697                     //Copy in &buf_TX[4] the address of
698                     the slave
699                 makeframe(&buf_TX[8], &buf_RX[8], 1); //
700                     Copy in &buf_TX[8] the type of
701                     command
702                 makeframe(&buf_TX[9], address_slave, 4);
703                     //Copy in &buf_TX[9] the address of
704                     the slave

```

```

682         makeframe(&buf_TX[15], &buf_RX[15], 1); //
           End of Frame
683         burstwr(TXFIFO, buf_TX, regdr(PKTLEN));
684
685         strobe(STX); //Switch to RX once the
           message is sent (MCSM1)
686         for (i=0; i<num; i++) U0putch(buf_TX[i]);
687     }
688     break;
689
690     case SETPIN: // Set Pin
691         pin = (buf_RX[13]-48)*10 + (buf_RX[14]-48);
           //Pin number in decimal
692
693         if(pin<=23) //If we are in PORT0
694         {
695             SET0 = (1<<pin); //Pin pin is set
696         }
697         else //If we are in PORT1
698         {
699             pin = pin-24; //Rest PORT0
700             SET1 = (1<<pin); //Pin pin is set
701         }
702
703         makeframe(buf_TX, &buf_RX[4], 4); //Copy in &
           buf_TX[0] the address of the master
704         makeframe(&buf_TX[4], address_slave, 4); //
           Copy in &buf_TX[4] the address of the
           slave
705         makeframe(&buf_TX[8], &buf_RX[8], 1); //Copy
           in &buf_TX[8] the type of command
706         makeframe(&buf_TX[9], (unsigned char*) "
           ACKACK", 6); //Copy in &buf_TX[9] the Set
           Pin ACK
707         makeframe(&buf_TX[15], &buf_RX[15], 1); //End
           of Frame
708         burstwr(TXFIFO, buf_TX, regdr(PKTLEN));
709
710         strobe(STX); //Switch to RX once the
           message is sent (MCSM1)
711         for (i=0; i<num; i++) U0putch(buf_TX[i]);
712     break;
713
714     case RESETPIN: //Reset Pin
715         pin = (buf_RX[13]-48)*10 + (buf_RX[14]-48);
           //Pin number in decimal
716
717         if(pin<=23) //If we are in PORT0
718         {
719             CLRO = (1<<pin); //Pin pin is reset
720         }
721         else //If we are in PORT1
722         {

```

```

723         pin = pin-24; //Rest PORT0
724         CLR1 = (1<<pin); //Pin pin is reset
725     }
726
727     makeframe(buf_TX, &buf_RX[4], 4); //Copy in &
728     buf_TX[0] the address of the master
729     makeframe(&buf_TX[4], address_slave, 4); //
730     Copy in &buf_TX[4] the address of the
731     slave
732     makeframe(&buf_TX[8], &buf_RX[8], 1); //Copy
733     in &buf_TX[8] the type of command
734     makeframe(&buf_TX[9], (unsigned char*) "
735     ACKACK", 6); //Copy in &buf_TX[9] the Set
736     Pin ACK
737     makeframe(&buf_TX[15], &buf_RX[15], 1); //End
738     of Frame
739     burstwr(TXFIFO, buf_TX, regd(PKTLEN));
740
741     strobe(STX); //Switch to IDLE once the
742     message is sent (MCSM1)
743     for (i=0; i<num; i++) U0putch(buf_TX[i]);
744     break;
745
746     case TEMPERATURE_SENSOR: // Analog Sensor
747     Temperature
748     _sprintf(temperature, "%03X", read_sensor());
749
750     makeframe(buf_TX, &buf_RX[4], 4); //Copy in &
751     buf_TX[0] the address of the master
752     makeframe(&buf_TX[4], address_slave, 4); //
753     Copy in &buf_TX[4] the address of the
754     slave
755     makeframe(&buf_TX[8], &buf_RX[8], 1); //Copy
756     in &buf_TX[8] the type of command
757     makeframe(&buf_TX[12], temperature, 3); //
758     Value from sensor
759     makeframe(&buf_TX[15], &buf_RX[15], 1); //End
760     of Frame
761     burstwr(TXFIFO, buf_TX, regd(PKTLEN));
762
763     strobe(STX); //Switch to RX once the
764     message is sent (MCSM1)
765     for (i=0; i<num; i++) U0putch(buf_TX[i]);
766     break;
767
768     case RECOVERCONFIG: // Recover Config Command
769     B0_16 = ((buf_RX[14] & 0x10) >> 4); //PIO0_16
770     B0_15 = ((buf_RX[14] & 0x20) >> 5); //PIO0_15
771     B1_22 = ((buf_RX[14] & 0x40) >> 6); //PIO1_22
772     B1_14 = ((buf_RX[14] & 0x80) >> 7); //PIO1_14
773
774     makeframe(buf_TX, &buf_RX[4], 4); //Copy in &
775     buf_TX[0] the address of the master

```

```
759         makeframe(&buf_TX[4], address_slave, 4); //
           Copy in &buf_TX[4] the address of the
           slave
760         makeframe(&buf_TX[8], &buf_RX[8], 1); //Copy
           in &buf_TX[8] the type of command
761         makeframe(&buf_TX[9], (unsigned char*) "
           ACKACK", 6); //Copy in &buf_TX[9] the
           recover config ACK
762         makeframe(&buf_TX[15], &buf_RX[15], 1); //End
           of Frame
763         burstwr(TXFIFO, buf_TX, regrd(PKTLEN));
764
765         strobe(STX); //Switch to RX once the
           message is sent (MCSM1)
766         for (i=0; i<num; i++) U0putch(buf_TX[i]);
767         break;
768     }
769     _printf("\nMARCSTATE=%02X\n", regrd(MARCSTATE));
770     initialize_vector(buf_TX, MESSAGELENGTH);
771 }
772 }
773     enable_interrupts();
774 }
775 }
776     sleep_mode(); //We can back to sleep state of MCU
777 }
778 }
```

Código E.2: Programa principal de los módulos esclavos



# Apéndice F

## Código de la aplicación para PC

A continuación se lista el código correspondiente a la aplicación para PC desarrollada con *wxDev-C++*.

```
1  ///-----  
2  ///  
3  /// @file      RadioTransceiverDlg.h  
4  /// @author    user  
5  /// Created:   08/02/2015 20:13:44  
6  /// @section   DESCRIPTION  
7  ///           RadioTransceiverDlg class declaration  
8  ///  
9  ///-----  
10  
11 #ifndef __RADIOTRANSCEIVERDLG_H__  
12 #define __RADIOTRANSCEIVERDLG_H__  
13  
14 #ifdef __BORLANDC__  
15     #pragma hdrstop  
16 #endif  
17  
18 #ifndef WX_PRECOMP  
19     #include <wx/wx.h>  
20     #include <wx/dialog.h>  
21 #else  
22     #include <wx/wxprec.h>  
23 #endif  
24  
25 ///////////////////////////////////////////////////COM.H////////////////////////////////////  
26  
27 #include <sys/time.h>  
28 #include <sys/types.h>  
29 #include <unistd.h>  
30 #include <string.h>  
31  
32 #include <fcntl.h>  
33 #include <stdio.h>
```

```
34
35 #include "windows.h"
36
37 #define SIZE_BUFF 200
38
39 #define true 1
40 #define false 0
41
42 #define PCFRAME 12
43 #define ADDRESS_LENGTH 4
44
45 //////////////////////////////////////////////////COM.H////////////////////////////////////
46
47 //////////////////////////////////////////////////SLAVE.H////////////////////////////////////
48
49 #define MAX_SLAVE 1024
50
51 //////////////////////////////////////////////////SLAVE.H////////////////////////////////////
52
53 //////////////////////////////////////////////////SLAVE LIST.H////////////////////////////////////
54
55 #include <errno.h>
56 #include <sys/stat.h>
57
58 //////////////////////////////////////////////////SLAVE LIST.H////////////////////////////////////
59
60 //Do not add custom headers between
61 //Header Include Start and Header Include End.
62 //wxDev-C++ designer will remove them. Add custom headers after the block.
63 ////Header Include Start
64 #include <wx/textctrl.h>
65 #include <wx/button.h>
66 #include <wx/richtext/richtextctrl.h>
67 #include <wx/stattext.h>
68 #include <wx/combobox.h>
69 ////Header Include End
70
71 ////Dialog Style Start
72 #undef RadioTransceiverDlg_STYLE
73 #define RadioTransceiverDlg_STYLE wxCAPTION | wxSYSTEM_MENU |
74     wxDIALOG_NO_PARENT | wxMINIMIZE_BOX | wxCLOSE_BOX
75 ////Dialog Style End
76
77 class RadioTransceiverDlg : public wxDialog
78 {
79     private:
80         DECLARE_EVENT_TABLE();
81
82         char path_com[1024];
83
84         char button_bbdd;
85
86         //////////////////////////////////////////////////COM variables////////////////////////////////////
```

```

86
87     HANDLE idComDev;
88     DCB dcb;
89     char pc_frame[PCFRAME];
90     char EOFr;          //Caracter ASCII de Nueva Línea (End Of Frame)
91     char buf_RX[SIZE_BUFF];
92
93     ///////////////////////////////////////////////////COM variables////////////////////////////////////
94
95     ///////////////////////////////////////////////////SLAVE variables////////////////////////////////////
96     typedef enum commands{READPINS=0,BROADCAST,SETPIN,
97                          RESETPIN,TEMPERATURE,RECOVERCONFIG} Orders;
98     Orders Commands;
99
100    typedef struct slave
101    {
102        char name[30];
103        char address[5];
104        char pin_state[9];
105    }Type_slave;
106
107    char pin_name[9][12];
108
109    char pin_number[9][12];
110
111    char command_name[255][12];
112
113    char command_number[255][2];
114
115    ///////////////////////////////////////////////////SLAVE variables////////////////////////////////////
116
117    ///////////////////////////////////////////////////SLAVE LIST variables////////////////////////////////////
118
119    typedef struct slave_list
120    {
121        int number;
122        Type_slave list[MAX_SLAVE];
123    }Type_slave_list;
124
125    Type_slave_list list_slave;
126
127    ///////////////////////////////////////////////////SLAVE LIST variables////////////////////////////////////
128
129    public:
130        RadioTransceiverDlg(wxWindow *parent, wxWindowID id = 1, const
            wxString &title = wxT("RadioTransceiver"), const wxPoint& pos =
            wxDefaultPosition, const wxSize& size = wxDefaultSize, long
            style = RadioTransceiverDlg_STYLE);
131        virtual ~RadioTransceiverDlg();
132        void cmb_slave_addressSelected(wxCommandEvent& event );
133        void cmb_dataSelected(wxCommandEvent& event );
134        void cmb_commandSelected(wxCommandEvent& event );
135        void cmb_slave_addressUpdated(wxCommandEvent& event );

```

```
136 void cmb_slave_addressUpdateUI (wxUpdateUIEvent& event);
137 void btn_load_bbddClick (wxCommandEvent& event);
138 void btn_connect_comClick (wxCommandEvent& event);
139 void btn_send_frameClick (wxCommandEvent& event);
140 void btn_search_slaveClick (wxCommandEvent& event);
141 void btn_recover_configClick (wxCommandEvent& event);
142 void btn_close_comClick (wxCommandEvent& event);
143 void btn_save_bbddClick (wxCommandEvent& event);
144 void cmb_slave_viewSelected (wxCommandEvent& event );
145 void btn_temperatureClick (wxCommandEvent& event);
146 void btn_delete_slaveClick (wxCommandEvent& event);
147 void Inicialization();
148
149 //////////////////////////////////////////////////COM metodos////////////////////////////////////
150
151 int AbrirCOM(const char* COM_PORT);
152 void CerrarCOM();
153 void EscribirCOM(const char* buffer) ;
154 void LeerCOM(char* buffer, int size);
155 int IniciarCOM(char* path);
156 void strcpy_sure(char *buff, char *command, char len);
157 void initialize_vector(char *buff, int length);
158 int tx_rx_frame(char *address_slave, char *command, char *data);
159
160 //////////////////////////////////////////////////COM metodos////////////////////////////////////
161
162 //////////////////////////////////////////////////SLAVE metodos////////////////////////////////////
163
164 void show_slave(const Type_slave *slave);
165 int detect_slave(Type_slave *slave, const char *string);
166 int temperature_slave(char *slave_address);
167 char translate_name_number(const char *string, char *matrix);
168 void restablish_config(Type_slave *slave);
169 int slave_pin_view(Type_slave *slave);
170
171 //////////////////////////////////////////////////SLAVE metodos////////////////////////////////////
172
173 //////////////////////////////////////////////////SLAVE LIST metodos////////////////////////////////////
174
175 void slave_list(const Type_slave_list *list);
176 int duplicate_slave_name(const Type_slave_list *list, const char*
    name);
177 void new_slave(Type_slave_list *list);
178 void delete_slave(Type_slave_list *list, const char* name);
179 Type_slave* const search_slave(Type_slave_list *list, const char*
    name);
180 void load_data_slaves(Type_slave_list *list);
181 void save_data_slaves(const Type_slave_list *list);
182 int num_reg_slaves(const char *bbdd,int size_reg);
183
184 //////////////////////////////////////////////////SLAVE LIST metodos////////////////////////////////////
185
186 private:
```

```
187 //Do not add custom control declarations between
188 //GUI Control Declaration Start and GUI Control Declaration End.
189 //wxDev-C++ will remove them. Add custom code after the block.
190 ///GUI Control Declaration Start
191 wxButton *btn_delete_slave;
192 wxTextCtrl *txt_temperature;
193 wxStaticText *WxStaticText16;
194 wxButton *btn_temperature;
195 wxButton *btn_save_bbdd;
196 wxButton *btn_close_com;
197 wxStaticText *WxStaticText15;
198 wxComboBox *cmb_slave_view;
199 wxTextCtrl *txt_pin_8;
200 wxTextCtrl *txt_pin_7;
201 wxTextCtrl *txt_pin_6;
202 wxTextCtrl *txt_pin_5;
203 wxTextCtrl *txt_pin_4;
204 wxTextCtrl *txt_pin_3;
205 wxTextCtrl *txt_pin_2;
206 wxStaticText *WxStaticText14;
207 wxStaticText *WxStaticText13;
208 wxStaticText *WxStaticText12;
209 wxStaticText *WxStaticText11;
210 wxStaticText *WxStaticText10;
211 wxStaticText *WxStaticText9;
212 wxStaticText *WxStaticText8;
213 wxStaticText *WxStaticText7;
214 wxTextCtrl *txt_pin_1;
215 wxStaticText *WxStaticText6;
216 wxComboBox *cmb_recover_config;
217 wxButton *btn_recover_config;
218 wxStaticText *WxStaticText5;
219 wxTextCtrl *txt_introduce_slave;
220 wxButton *btn_search_slave;
221 wxButton *btn_send_frame;
222 wxButton *btn_load_bbdd;
223 wxButton *btn_connect_com;
224 wxStaticText *WxStaticText4;
225 wxRichTextCtrl *hyperterminal;
226 wxStaticText *WxStaticText3;
227 wxStaticText *WxStaticText2;
228 wxStaticText *WxStaticText1;
229 wxComboBox *cmb_data;
230 wxComboBox *cmb_command;
231 wxComboBox *cmb_slave_address;
232 ///GUI Control Declaration End
233
234 private:
235 //Note: if you receive any error with these enum IDs, then you need
236 //to
237 //change your old form code that are based on the #define control
238 //IDs.
239 //#defines may replace a numeric value for the enum names.
```

```
238 //Try copy and pasting the below block in your old form header
      files.
239 enum
240 {
241     ///GUI Enum Control ID Start
242     ID_BTN_DELETE_SLAVE = 1047,
243     ID_TXT_TEMPERATURE = 1046,
244     ID_WXSTATICTEXT16 = 1044,
245     ID_BTN_TEMPERATURE = 1043,
246     ID_BTN_SAVE_BBDD = 1042,
247     ID_BTN_CLOSE_COM = 1041,
248     ID_WXSTATICTEXT15 = 1040,
249     ID_CMB_SLAVE_VIEW = 1039,
250     ID_TXT_PIN_8 = 1038,
251     ID_TXT_PIN_7 = 1037,
252     ID_TXT_PIN_6 = 1036,
253     ID_TXT_PIN_5 = 1035,
254     ID_TXT_PIN_4 = 1034,
255     ID_TXT_PIN_3 = 1033,
256     ID_TXT_PIN_2 = 1032,
257     ID_WXSTATICTEXT14 = 1031,
258     ID_WXSTATICTEXT13 = 1030,
259     ID_WXSTATICTEXT12 = 1029,
260     ID_WXSTATICTEXT11 = 1028,
261     ID_WXSTATICTEXT10 = 1027,
262     ID_WXSTATICTEXT9 = 1026,
263     ID_WXSTATICTEXT8 = 1025,
264     ID_WXSTATICTEXT7 = 1024,
265     ID_TXT_PIN_1 = 1023,
266     ID_WXSTATICTEXT6 = 1021,
267     ID_CMB_RECOVER_CONFIG = 1020,
268     ID_BTN_RECOVER_CONFIG = 1019,
269     ID_WXSTATICTEXT5 = 1018,
270     ID_TXT_INTRODUCE_SLAVE = 1017,
271     ID_BTN_SEARCH_SLAVE = 1014,
272     ID_BTN_SEND_FRAME = 1013,
273     ID_BTN_LOAD_BBDD = 1012,
274     ID_BTN_CONNECT_COM = 1011,
275     ID_WXSTATICTEXT4 = 1010,
276     ID_HYPERTERMINAL = 1009,
277     ID_WXSTATICTEXT3 = 1007,
278     ID_WXSTATICTEXT2 = 1006,
279     ID_WXSTATICTEXT1 = 1005,
280     ID_CMB_DATA = 1004,
281     ID_CMB_COMMAND = 1003,
282     ID_CMB_SLAVE_ADDRESS = 1002,
283     ///GUI Enum Control ID End
284     ID_DUMMY_VALUE_ //don't remove this value unless you have other
        enum values
285 };
286
287 private:
288     void OnClose(wxCloseEvent& event);
```

```

289         void CreateGUIControls();
290     };
291
292 #endif

```

Código F.1: Fichero de cabecera de la aplicación para PC

```

1  ///-----
2  ///
3  /// @file      RadioTransceiverDlg.cpp
4  /// @author    user
5  /// Created:   08/02/2015 20:13:45
6  /// @section   DESCRIPTION
7  ///           RadioTransceiverDlg class implementation
8  ///
9  ///-----
10
11 #include "RadioTransceiverDlg.h"
12
13 //Do not add custom headers
14 //wxDev-C++ designer will remove them
15 ////Header Include Start
16 ////Header Include End
17
18 //-----
19 // RadioTransceiverDlg
20 //-----
21 //Add Custom Events only in the appropriate block.
22 //Code added in other places will be removed by wxDev-C++
23 ////Event Table Start
24 BEGIN_EVENT_TABLE(RadioTransceiverDlg, wxDialog)
25     ///Manual Code Start
26     ///Manual Code End
27
28     EVT_CLOSE(RadioTransceiverDlg::OnClose)
29     EVT_BUTTON(ID_BTN_DELETE_SLAVE, RadioTransceiverDlg::
30         btn_delete_slaveClick)
31     EVT_BUTTON(ID_BTN_TEMPERATURE, RadioTransceiverDlg::btn_temperatureClick
32         )
33     EVT_BUTTON(ID_BTN_SAVE_BBDD, RadioTransceiverDlg::btn_save_bbddClick)
34     EVT_BUTTON(ID_BTN_CLOSE_COM, RadioTransceiverDlg::btn_close_comClick)
35     EVT_COMBOBOX(ID_CMB_SLAVE_VIEW, RadioTransceiverDlg::
36         cmb_slave_viewSelected)
37     EVT_BUTTON(ID_BTN_RECOVER_CONFIG, RadioTransceiverDlg::
38         btn_recover_configClick)
39     EVT_BUTTON(ID_BTN_SEARCH_SLAVE, RadioTransceiverDlg::
40         btn_search_slaveClick)
41     EVT_BUTTON(ID_BTN_SEND_FRAME, RadioTransceiverDlg::btn_send_frameClick)
42     EVT_BUTTON(ID_BTN_LOAD_BBDD, RadioTransceiverDlg::btn_load_bbddClick)
43     EVT_BUTTON(ID_BTN_CONNECT_COM, RadioTransceiverDlg::btn_connect_comClick
44         )

```

```

39     EVT_COMBOBOX (ID_CMB_DATA, RadioTransceiverDlg::cmb_dataSelected)
40     EVT_COMBOBOX (ID_CMB_COMMAND, RadioTransceiverDlg::cmb_commandSelected)
41     EVT_COMBOBOX (ID_CMB_SLAVE_ADDRESS, RadioTransceiverDlg::
        cmb_slave_addressSelected)
42 END_EVENT_TABLE ()
43 ////Event Table End
44
45 RadioTransceiverDlg::RadioTransceiverDlg (wxWindow *parent, wxWindowID id,
        const wxString &title, const wxPoint &position, const wxSize& size, long
        style)
46 : wxDialog (parent, id, title, position, size, style)
47 {
48     button_bbdd = 0;
49     ////////////////////COM inicializacion////////////////////
50
51     dcb = {0};
52     pc_frame [PCFRAME] = {0};           //Vector que almacenará el mensaje a enviar
53     EOFr = 'L';                         //Caracter ASCII L (End Of Frame)
54     buf_RX [SIZE_BUFF] = {0};          //Buffer de recepción
55
56     ////////////////////COM inicializacion////////////////////
57
58     ////////////////////SLAVE inicializacion////////////////////
59
60     strcpy (&pin_name [0] [0], "ORANGE");           //PIO1_14
61     strcpy (&pin_name [1] [0], "PIO1_22");
62     strcpy (&pin_name [2] [0], "PIO0_15");
63     strcpy (&pin_name [3] [0], "PIO0_16");
64     strcpy (&pin_name [4] [0], "PIO0_17");
65     strcpy (&pin_name [5] [0], "PIO0_18");
66     strcpy (&pin_name [6] [0], "PIO0_19");
67     strcpy (&pin_name [7] [0], "PIO1_16");
68
69     strcpy (&pin_number [0] [0], "000003");
70     strcpy (&pin_number [1] [0], "000046");           //22 + 24 (PORT0)
71     strcpy (&pin_number [2] [0], "000015");
72     strcpy (&pin_number [3] [0], "000016");
73     strcpy (&pin_number [4] [0], "000017");
74     strcpy (&pin_number [5] [0], "000018");
75     strcpy (&pin_number [6] [0], "000019");
76     strcpy (&pin_number [7] [0], "000040");           //16 + 24 (PORT0)
77
78     strcpy (&command_name [0] [0], "READ PINS");     //Commandos
79     strcpy (&command_name [1] [0], "BROADCAST");
80     strcpy (&command_name [2] [0], "SET PIN");
81     strcpy (&command_name [3] [0], "RESET PIN");
82
83     strcpy (&command_number [0] [0], "1");
84     strcpy (&command_number [1] [0], "2");
85     strcpy (&command_number [2] [0], "3");
86     strcpy (&command_number [3] [0], "4");
87     strcpy (&command_number [4] [0], "5");
88     strcpy (&command_number [5] [0], "6");

```

```

89
90 //////////////////////////////////////////////////SLAVE inicializacion////////////////////////////////////
91
92 //////////////////////////////////////////////////SLAVE LIST inicializacion////////////////////////////////////
93
94 list_slave = {0};
95
96 //////////////////////////////////////////////////SLAVE LIST inicializacion////////////////////////////////////
97
98 CreateGUIControls();
99
100 Inicialization();
101 }
102
103 RadioTransceiverDlg::~RadioTransceiverDlg()
104 {
105 }
106
107 void RadioTransceiverDlg::Inicialization()
108 {
109     cmb_slave_address->Disable(); //Desactivamos todos los botones
110     cmb_command->Disable();
111     cmb_data->Disable();
112     cmb_slave_view->Disable();
113     cmb_recover_config->Disable();
114
115     btn_close_com->Disable();
116     btn_recover_config->Disable();
117     btn_search_slave->Disable();
118     btn_delete_slave->Disable();
119     btn_load_bbdd->Disable();
120     btn_send_frame->Disable();
121     btn_save_bbdd->Disable();
122     btn_temperature->Disable();
123 }
124
125 void RadioTransceiverDlg::CreateGUIControls()
126 {
127     //Do not add custom code between
128     //GUI Items Creation Start and GUI Items Creation End.
129     //wxDev-C++ designer will remove them.
130     //Add the custom code before or after the blocks
131     ////GUI Items Creation Start
132
133     btn_delete_slave = new wxButton(this, ID_BTN_DELETE_SLAVE, _("Delete
134     Slave"), wxPoint(328, 335), wxSize(75, 25), 0, wxDefaultValidator, _
135     ("btn_delete_slave"));
136
137     txt_temperature = new wxTextCtrl(this, ID_TXT_TEMPERATURE, _(""),
138     wxPoint(153, 377), wxSize(60, 19), wxTE_READONLY, wxDefaultValidator
139     , _("txt_temperature"));

```

```
137 WxStaticText16 = new wxStaticText(this, ID_WXSTATICTEXT16, _("°C"),
138     wxPoint(216, 377), wxDefaultSize, 0, _("WxStaticText16"));
139 btn_temperature = new wxButton(this, ID_BTN_TEMPERATURE, _("Temperature
140     "), wxPoint(30, 375), wxSize(75, 25), 0, wxDefaultValidator, _("
141     btn_temperature"));
142 btn_save_bbdd = new wxButton(this, ID_BTN_SAVE_BBDD, _("Save BBDD"),
143     wxPoint(490, 70), wxSize(75, 25), 0, wxDefaultValidator, _("
144     btn_save_bbdd"));
145 btn_close_com = new wxButton(this, ID_BTN_CLOSE_COM, _("Close COM"),
146     wxPoint(490, 31), wxSize(75, 25), 0, wxDefaultValidator, _("
147     btn_close_com"));
148 WxStaticText15 = new wxStaticText(this, ID_WXSTATICTEXT15, _("Slave
149     Pins View"), wxPoint(559, 100), wxDefaultSize, 0, _("WxStaticText15
150     "));
151 wxArrayString arrayStringFor_cmb_slave_view;
152 cmb_slave_view = new wxComboBox(this, ID_CMB_SLAVE_VIEW, _(""), wxPoint
153     (530, 122), wxSize(145, 23), arrayStringFor_cmb_slave_view,
154     wxCB_DROPDOWN | wxCB_READONLY | wxCB_SORT, wxDefaultValidator, _("
155     cmb_slave_view"));
156 txt_pin_8 = new wxTextCtrl(this, ID_TXT_PIN_8, _(""), wxPoint(600, 370)
157     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_8"))
158     ;
159 txt_pin_7 = new wxTextCtrl(this, ID_TXT_PIN_7, _(""), wxPoint(600, 340)
160     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_7"))
161     ;
162 txt_pin_6 = new wxTextCtrl(this, ID_TXT_PIN_6, _(""), wxPoint(600, 310)
163     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_6"))
164     ;
165 txt_pin_5 = new wxTextCtrl(this, ID_TXT_PIN_5, _(""), wxPoint(600, 280)
166     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_5"))
167     ;
168 txt_pin_4 = new wxTextCtrl(this, ID_TXT_PIN_4, _(""), wxPoint(600, 250)
169     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_4"))
170     ;
171 txt_pin_3 = new wxTextCtrl(this, ID_TXT_PIN_3, _(""), wxPoint(600, 220)
172     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_3"))
173     ;
174 txt_pin_2 = new wxTextCtrl(this, ID_TXT_PIN_2, _(""), wxPoint(600, 190)
175     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_2"))
176     ;
```

```
164 WxStaticText14 = new wxStaticText(this, ID_WXSTATICTEXT14, _("PIO1_16")
165     , wxPoint(540, 371), wxDefaultSize, 0, _("WxStaticText14"));
166 WxStaticText13 = new wxStaticText(this, ID_WXSTATICTEXT13, _("PIO0_19")
167     , wxPoint(540, 341), wxDefaultSize, 0, _("WxStaticText13"));
168 WxStaticText12 = new wxStaticText(this, ID_WXSTATICTEXT12, _("PIO0_18")
169     , wxPoint(540, 311), wxDefaultSize, 0, _("WxStaticText12"));
170 WxStaticText11 = new wxStaticText(this, ID_WXSTATICTEXT11, _("PIO0_17")
171     , wxPoint(540, 281), wxDefaultSize, 0, _("WxStaticText11"));
172 WxStaticText10 = new wxStaticText(this, ID_WXSTATICTEXT10, _("PIO0_16")
173     , wxPoint(540, 251), wxDefaultSize, 0, _("WxStaticText10"));
174 WxStaticText9 = new wxStaticText(this, ID_WXSTATICTEXT9, _("PIO0_15"),
175     wxPoint(540, 221), wxDefaultSize, 0, _("WxStaticText9"));
176 WxStaticText8 = new wxStaticText(this, ID_WXSTATICTEXT8, _("PIO1_22"),
177     wxPoint(540, 191), wxDefaultSize, 0, _("WxStaticText8"));
178 WxStaticText7 = new wxStaticText(this, ID_WXSTATICTEXT7, _("PIO1_14"),
179     wxPoint(540, 161), wxDefaultSize, 0, _("WxStaticText7"));
180 txt_pin_1 = new wxTextCtrl(this, ID_TXT_PIN_1, _(""), wxPoint(600, 160)
181     , wxSize(45, 19), wxTE_READONLY, wxDefaultValidator, _("txt_pin_1"))
182     ;
183 WxStaticText6 = new wxStaticText(this, ID_WXSTATICTEXT6, _("Select
184     Slave"), wxPoint(189, 311), wxDefaultSize, 0, _("WxStaticText6"));
185 wxArrayString arrayStringFor_cmb_recover_config;
186 cmb_recover_config = new wxComboBox(this, ID_CMB_RECOVER_CONFIG, _(""),
187     wxPoint(153, 336), wxSize(145, 23),
188     arrayStringFor_cmb_recover_config, wxCB_DROPDOWN | wxCB_READONLY |
189     wxCB_SORT, wxDefaultValidator, _("cmb_recover_config"));
190 btn_recover_config = new wxButton(this, ID_BTN_RECOVER_CONFIG, _("
191     Recover Config."), wxPoint(30, 335), wxSize(91, 25), 0,
192     wxDefaultValidator, _("btn_recover_config"));
193 WxStaticText5 = new wxStaticText(this, ID_WXSTATICTEXT5, _("Introduce
194     Slave name"), wxPoint(155, 240), wxDefaultSize, 0, _("WxStaticText5"
195     ));
196 txt_introduce_slave = new wxTextCtrl(this, ID_TXT_INTRODUCE_SLAVE, _("")
197     , wxPoint(153, 266), wxSize(121, 19), wxTE_PROCESS_ENTER,
198     wxDefaultValidator, _("txt_introduce_slave"));
199 btn_search_slave = new wxButton(this, ID_BTN_SEARCH_SLAVE, _("Search
200     Slave"), wxPoint(30, 263), wxSize(75, 25), 0, wxDefaultValidator, _("
201     btn_search_slave"));
```

```
195 btn_send_frame = new wxButton(this, ID_BTN_SEND_FRAME, _("Send Frame"),
    wxPoint(386, 107), wxSize(75, 25), 0, wxDefaultValidator, _("
    btn_send_frame"));
196
197 btn_load_bbdd = new wxButton(this, ID_BTN_LOAD_BBDD, _("Load BBDD"),
    wxPoint(385, 69), wxSize(83, 25), 0, wxDefaultValidator, _("
    btn_load_bbdd"));
198
199 btn_connect_com = new wxButton(this, ID_BTN_CONNECT_COM, _("Connect COM
    "), wxPoint(385, 31), wxSize(82, 25), 0, wxDefaultValidator, _("
    btn_connect_com"));
200
201 WxStaticText4 = new wxStaticText(this, ID_WXSTATICTEXT4, _("
    Hyperterminal"), wxPoint(150, 8), wxDefaultSize, 0, _("WxStaticText4
    "));
202
203 hyperterminal = new wxRichTextCtrl(this, ID_HYPERTERMINAL, _(""),
    wxPoint(29, 32), wxSize(321, 130), wxRE_READONLY | wxRE_MULTILINE,
    wxDefaultValidator, _("hyperterminal"));
204 hyperterminal->SetMaxLength(0);
205 hyperterminal->SetFocus();
206 hyperterminal->SetInsertionPointEnd();
207
208 WxStaticText3 = new wxStaticText(this, ID_WXSTATICTEXT3, _("Data
    Selection"), wxPoint(380, 168), wxDefaultSize, 0, _("WxStaticText3"
    ));
209
210 WxStaticText2 = new wxStaticText(this, ID_WXSTATICTEXT2, _("Command
    Selection"), wxPoint(208, 168), wxDefaultSize, 0, _("WxStaticText2"
    ));
211
212 WxStaticText1 = new wxStaticText(this, ID_WXSTATICTEXT1, _("Slave
    Selection"), wxPoint(65, 168), wxDefaultSize, 0, _("WxStaticText1"
    ));
213
214 wxArrayString arrayStringFor_cmb_data;
215 cmb_data = new wxComboBox(this, ID_CMB_DATA, _(""), wxPoint(350, 195),
    wxSize(145, 23), arrayStringFor_cmb_data, wxCB_DROPDOWN |
    wxCB_READONLY | wxCB_SORT, wxDefaultValidator, _("cmb_data"));
216
217 wxArrayString arrayStringFor_cmb_command;
218 cmb_command = new wxComboBox(this, ID_CMB_COMMAND, _(""), wxPoint(190,
    195), wxSize(145, 23), arrayStringFor_cmb_command, wxCB_DROPDOWN |
    wxCB_READONLY | wxCB_SORT, wxDefaultValidator, _("cmb_command"));
219
220 wxArrayString arrayStringFor_cmb_slave_address;
221 cmb_slave_address = new wxComboBox(this, ID_CMB_SLAVE_ADDRESS, _(""),
    wxPoint(30, 195), wxSize(145, 23), arrayStringFor_cmb_slave_address,
    wxCB_DROPDOWN | wxCB_READONLY | wxCB_SORT, wxDefaultValidator, _("
    cmb_slave_address"));
222
223 SetTitle(_("RadioTransceiver"));
224 SetIcon(wxNullIcon);
```

```

225     SetSize(8,8,700,450);
226     Center();
227
228     ////GUI Items Creation End
229 }
230
231 void RadioTransceiverDlg::OnClose(wxCloseEvent& /*event*/)
232 {
233     Destroy();
234 }
235
236 //////////////////////////////////////////////////COM funciones//////////////////////////////////////
237
238 //-----
239 // int AbrirCOM(const char* COM_PORT)
240 //
241 // DESCRIPCION:
242 //     Esta función es empleada para abrir un puerto serie del PC
243 //
244 // ARGUMENTOS:
245 //     Puerto COM que se desea abrir
246 //-----
247
248 int RadioTransceiverDlg::AbrirCOM(const char* COM_PORT)
249 {
250     idComDev = CreateFileA(COM_PORT, GENERIC_READ | GENERIC_WRITE,
251         0, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
252
253     if(idComDev ==INVALID_HANDLE_VALUE)
254     {
255         //hyperterminal->AppendText(wxString::Format("Error 1, puerto serie
256             : %d\n", GetLastError()));
257         return false;
258     }
259
260     dcb.DCBlength=sizeof(dcb);
261
262     if(!GetCommState(idComDev, &dcb))
263     {
264         //hyperterminal->AppendText(wxString::Format("Error 2, puerto serie
265             \n"));
266         return false;
267     }
268
269     dcb.BaudRate = (DWORD) CBR_115200;
270     dcb.ByteSize = (BYTE) 8;
271     dcb.Parity = (BYTE) NOPARITY;
272     dcb.StopBits = (BYTE) ONESTOPBIT;
273
274     if(!SetCommState(idComDev, &dcb))
275     {
276         //hyperterminal->AppendText(wxString::Format("Error 3,
277             SetCommStatus\n"));

```

```
275     return false;
276 }
277
278 COMMTIMEOUTS timeouts={0};
279
280 //The maximum time allowed to elapse before the arrival of the next
281 //byte on
282 //the communications line, in milliseconds.If the interval between the
283 //arrival
284 //of any two bytes exceeds this amount, the ReadFile operation is
285 //completed and
286 //any buffered data is returned
287 timeouts.ReadIntervalTimeout = 50;
288
289 //A constant used to calculate the total time-out period for read
290 //operations,
291 //in milliseconds. For each read operation, this value is added to the
292 //product
293 //of the ReadTotalTimeoutMultiplier member and the requested number of
294 //bytes.
295 timeouts.ReadTotalTimeoutConstant = 50;
296
297 //The multiplier used to calculate the total time-out period for read
298 //operations,
299 //in milliseconds. For each read operation, this value is multiplied by
300 //the
301 //requested number of bytes to be read.
302 timeouts.ReadTotalTimeoutMultiplier = 10;
303
304 //Idem as ReadTotalTimeoutConstant
305 timeouts.WriteTotalTimeoutConstant = 50;
306
307 //Idem as ReadTotalTimeoutMultiplier
308 timeouts.WriteTotalTimeoutMultiplier = 10;
309
310 if(!SetCommTimeouts(idComDev, &timeouts))
311 {
312     //hyperterminal->AppendText(wxString::Format("Error 4, timeouts\n")
313     //);
314     return false;
315 }
316 return true;
317 }
318
319 //-----
320 // void CerrarCOM()
321 //
322 // DESCRIPCION:
323 //     Esta función es empleada para cerrar un puerto serie del PC
324 //
325 // ARGUMENTOS:
326 //     No tiene
327 //-----
```

```

319
320 void RadioTransceiverDlg::CerrarCOM()
321 {
322     CloseHandle(idComDev);
323 }
324
325 //-----
326 // void EscribirCOM(const char* buffer)
327 //
328 // DESCRIPCION:
329 //     Esta función es empleada para escribir en un puerto serie del PC
330 //
331 // ARGUMENTOS:
332 //     Buffer donde se encuentran los caracteres a escribir
333 //-----
334
335 void RadioTransceiverDlg::EscribirCOM(const char* buffer)
336 {
337     DWORD numbytes = 0;
338     if(!WriteFile(idComDev, buffer, PCFRAME, &numbytes, NULL))
339         hyperterminal->AppendText(wxString::Format("<-Error en la escritura
340             \n"));
341     //else
342     //hyperterminal->AppendText(wxString::Format("<-Escritura con exito
343         \n"));
344 }
345
346 //-----
347 // void LeerCOM(char* buffer, int size)
348 //
349 // DESCRIPCION:
350 //     Esta función es empleada para leer de un puerto serie del PC
351 //
352 // ARGUMENTOS:
353 //     Buffer donde se almacenarán los caracteres y tamaño de la lectura
354 //-----
355
356 void RadioTransceiverDlg::LeerCOM(char* buffer, int size)
357 {
358     DWORD numbytes = 0;
359     if(!ReadFile(idComDev, buffer, size, &numbytes, NULL))
360         hyperterminal->AppendText(wxString::Format("<-Error en la lectura\n
361             "));
362     //else
363     //hyperterminal->AppendText(wxString::Format("<-Lectura con exito\n
364         "));
365 }
366
367 //-----
368 // int IniciarCOM(char* path)
369 //
370 // DESCRIPCION:

```

```
367 //      Esta función es empleada para localizar un puerto serie del PC
368 //      habilitado
369 //      ARGUMENTOS:
370 //      Variable donde se almacenará el path del COM
371 //-----
372
373 int RadioTransceiverDlg::IniciarCOM(char* path)
374 {
375     int i;
376     for (i=20;i>1;i--)
377     {
378         sprintf(path, "\\\\.\\COM%d", i);
379         if(AbrirCOM(path))
380             goto L1;
381         CerrarCOM();
382     }
383     hyperterminal->AppendText(wxString::Format("->Puerto serie no
384         encontrado\n"));
385     return 0;
386 L1:
387     CerrarCOM();
388     hyperterminal->AppendText(wxString::Format("->Puerto serie: %d\
389         nVelocidad: 115200 baudios\n", i));
390     return 1;
391 }
392 //-----
393 // void strcpy_sure(char *buff, char *command, char len)
394 //
395 // DESCRIPCION:
396 //      Esta función es empleada para copiar datos entre dos bufferes
397 //
398 // ARGUMENTOS:
399 //      Buffer origen, buffer destino y número de caracteres a copiar
400 //-----
401 void RadioTransceiverDlg::strcpy_sure(char *buff, char *command, char len)
402 {
403     while(len--)
404         *buff++ = *command++;
405 }
406 //-----
407 // void initialize_vector(char *buff, int length)
408 //
409 // DESCRIPCION:
410 //      Esta función es empleada para inicializar el contenido de un vector
411 //
412 // ARGUMENTOS:
413 //      Buffer a inicializar y longitud del vector a inicializar
414 //-----
415
416
```

```

417 void RadioTransceiverDlg::initialize_vector(char *buff, int length)
418 {
419     while(length--)
420         *buff++ = 0;
421 }
422
423 //-----
424 // int tx_rx_frame(char *address_slave, char *command, char *data)
425 //
426 // DESCRIPCION:
427 //     Esta función es empleada para transmitir y recibir una orden
428 //
429 // ARGUMENTOS:
430 //     Módulo esclavo destino, orden a realizar y datos a enviar
431 //-----
432
433 int RadioTransceiverDlg::tx_rx_frame(char *address_slave, char *command,
434     char *data)
435 {
436     int j;
437
438     initialize_vector(buf_RX, SIZE_BUFF);
439     initialize_vector(pc_frame, PCFRAME);
440
441     strcpy_sure(pc_frame, address_slave, ADDRESS_LENGTH);
442     strcpy_sure(&pc_frame[4], command, 1);
443     strcpy_sure(&pc_frame[5], data, 6);
444     strcpy_sure(&pc_frame[11], &EOFr, 1);
445
446     //for(j=0; j<12; j++)
447     // hyperterminal->AppendText(wxString::Format("Caracter %c\n",
448     //     pc_frame[j]));
449
450     EscribirCOM(pc_frame);
451
452     LeerCOM(buf_RX, SIZE_BUFF);
453
454     if(!strcmp(buf_RX, "Error") || (buf_RX[4] != pc_frame[4]) || !strlen(
455         buf_RX))
456         return 1; //Si se leyó error o '0' o el comando solicitado y el
457         recibido son distintos
458     else
459     {
460         hyperterminal->AppendText(wxString::Format("\nMensaje recibido : ")
461             );
462         for(j=0; j<36; j++) //Limitamos el número de caracteres a mostrar
463             por el terminal
464         {
465             if (buf_RX[j] == 0x00) //En caso de que sea NULL representamos
466                 espacio
467                 hyperterminal->AppendText(wxString::Format(" "));
468             else

```

```

462         hyperterminal->AppendText (wxString::Format ("%c",buf_RX[j]))
463         ;
464     }
465     return 0;
466 }
467
468 ///////////////////////////////////////////////////COM funciones//////////////////////////////////////
469
470 ///////////////////////////////////////////////////SLAVE funciones//////////////////////////////////////
471
472 //-----
473 // char translate_name_number(const char* string, char* matrix)
474 //
475 // DESCRIPCION:
476 //     Esta función es empleada para traducir de un nombre a un número
477 //
478 // ARGUMENTOS:
479 //     Nombre a traducir y matriz que contiene los nombres
480 //-----
481
482 char RadioTransceiverDlg::translate_name_number(const char* string, char*
matrix)
483 {
484     char i=0;
485
486     while(*matrix)
487     {
488         if(!strcmp(matrix,string))
489             return i;
490         matrix+=12;    //Siguiete fila de la matriz
491         i++;
492     }
493 }
494
495 //-----
496 // void show_slave(const Type_slave *slave)
497 //
498 // DESCRIPCION:
499 //     Esta función es empleada para mostrar los esclavos en la interfaz
gráfica
500 //
501 // ARGUMENTOS:
502 //     Esclavo cuyo nombre se desea mostrar
503 //-----
504
505 void RadioTransceiverDlg::show_slave(const Type_slave *slave)
506 {
507     cmb_slave_address->Append(slave->name);
508     cmb_recover_config->Append(slave->name);
509     cmb_slave_view->Append(slave->name);
510 }
511

```

```

512 //-----
513 // int detect_slave(Type_slave *slave, const char *name)
514 //
515 // DESCRIPCION:
516 //     Esta función es empleada para detectar la presencia de un módulo
517 //
518 // ARGUMENTOS:
519 //     Nombre del esclavo a localizar y estructura para almacenarlo
520 //-----
521
522 int RadioTransceiverDlg::detect_slave(Type_slave *slave, const char *name)
523 {
524     char broadcast_address[4]={0};
525     char contador;
526
527     for(contador = 0; contador <=10;contador++)
528     {
529         if(!tx_rx_frame(broadcast_address,command_number[BROADCAST],(char*)
530             "no_use"))
531             goto L1;
532     }
533     hyperterminal->AppendText(wxString::Format("\n->Error, the operation
534         couldn't be set\n"));
535     return 1; //Hubo un error
536 L1:
537     strcpy(slave->name, name); //Mostramos el nuevo esclavo en la interfaz
538         gráfica
539     strcpy_sure(slave->address, &buf_RX[5],4);
540     cmb_slave_address->Append(slave->name);
541     cmb_slave_view->Append(slave->name);
542     cmb_recover_config->Append(slave->name);
543     return 0;
544 }
545
546 //-----
547 // int temperature_slave(char *slave_address)
548 //
549 // DESCRIPCION:
550 //     Esta función es empleada para leer la temperatura de un módulo
551 //
552 // ARGUMENTOS:
553 //     Dirección del esclavo cuya temperatura se desea medir
554 //-----
555
556 int RadioTransceiverDlg::temperature_slave(char *slave_address)
557 {
558     char temperature[3]={0}; //3 digitos hexadecimales
559     float value;
560     char j;
561     char contador;
562
563     for(contador = 0;contador <=10; contador++)
564     {

```

```

562         if(!tx_rx_frame(slave_address,command_number[TEMPERATURE],(char*)"
563             no_use"))
564             goto L1;
565     }
566     hyperterminal->AppendText(wxString::Format("\n->Error, the operation
567         couldn't be set\n"));
568     return 1;
569 L1:
570     strcpy_sure(temperature,&buf_RX[8],3);
571
572     for(j=0;j<3;j++)    //Convertimos los dígitos de ASCII a binario
573     {
574         if((temperature[j]>=48) && (temperature[j]<=57))
575             temperature[j] = temperature[j]-48;
576         if((temperature[j]>=65) && (temperature[j]<=70))
577             temperature[j] = temperature[j]-55;
578     }
579
580     value = temperature[0]*16*16 + temperature[1]*16 + temperature[2];
581     value = (value*3.3)/1024;    //ADC del micro es de 10bits de
582         resolución
583     value = (value - 0.75)/0.00243; //Curva de transferencia del sensor
584
585     txt_temperature->Clear();
586     txt_temperature->AppendText(wxString::Format("%.2f",value));
587     return 0;
588 }
589
590 //-----
591 // void restablish_config(Type_slave *slave)
592 //
593 // DESCRIPCION:
594 //     Esta función es empleada para reestablecer la configuración de un
595 //     módulo
596 //
597 // ARGUMENTOS:
598 //     Estructura del esclavo cuya configuración se desea recuperar
599 //-----
600
601 void RadioTransceiverDlg::restablish_config(Type_slave *slave)
602 {
603     char i;
604     char state[6] = {0};
605     char contador;
606
607     for(i=0;i<4;i++)    //Los 4 pines que son salida
608     {
609         if(slave->pin_state[i] == '1')
610             state[5] |= (1<<(7-i));
611     }
612     for(contador = 0;contador <=10; contador++)
613     {

```

```

610         if(!tx_rx_frame(slave->address,command_number[RECOVERCONFIG],state)
611             )
612             goto L1;
613     }
614     hyperterminal->AppendText(wxString::Format("\n->Error, the operation
615         couldn't be set\n"));
616     hyperterminal->AppendText(wxString::Format("\n->Recover configuration
617         couldn't be set\n"));
618     return;
619 L1:
620     hyperterminal->AppendText(wxString::Format("\n->Recover configuration
621         completed\n"));
622     slave_pin_view(slave);
623 }
624 //-----
625 // int slave_pin_view(Type_slave *slave)
626 //
627 // DESCRIPCION:
628 //     Esta función es empleada para leer el estado de los pines de un
629 //     módulo
630 //
631 // ARGUMENTOS:
632 //     Estructura del esclavo cuyos pines se desean leer
633 //-----
634 int RadioTransceiverDlg::slave_pin_view(Type_slave *slave)
635 {
636     char port;
637     char* ptr_port = &port;
638     char contador;
639
640     txt_pin_1->Clear();
641     txt_pin_2->Clear();
642     txt_pin_3->Clear();
643     txt_pin_4->Clear();
644     txt_pin_5->Clear();
645     txt_pin_6->Clear();
646     txt_pin_7->Clear();
647     txt_pin_8->Clear();
648
649     for(contador = 0;contador <=10;contador++)
650     {
651         if(!tx_rx_frame(slave->address,command_number[READPINS],(char*)"
652             no_use"))
653             goto L1;
654     }
655     hyperterminal->AppendText(wxString::Format("\n->Error, the operation
656         couldn't be set\n"));
657     return 1;
658 L1:
659     hyperterminal->AppendText(wxString::Format("\n->Read Pins measured
660         successfully\n"));

```

```

655 strcpy_sure(ptr_port, &buf_RX[10], 1);
656
657 //En base a lo recibido actuamos
658
659 (port&0x01)?txt_pin_8->AppendText("SET"):txt_pin_8->AppendText("RESET")
660 ;
661 (port&0x01)?slave->pin_state[7] = '1':slave->pin_state[7] = '0';
662
663 (port&0x02)?txt_pin_7->AppendText("SET"):txt_pin_7->AppendText("RESET")
664 ;
665 (port&0x02)?slave->pin_state[6] = '1':slave->pin_state[6] = '0';
666
667 (port&0x04)?txt_pin_6->AppendText("SET"):txt_pin_6->AppendText("RESET")
668 ;
669 (port&0x04)?slave->pin_state[5] = '1':slave->pin_state[5] = '0';
670
671 (port&0x08)?txt_pin_5->AppendText("SET"):txt_pin_5->AppendText("RESET")
672 ;
673 (port&0x08)?slave->pin_state[4] = '1':slave->pin_state[4] = '0';
674
675 (port&0x10)?txt_pin_4->AppendText("SET"):txt_pin_4->AppendText("RESET")
676 ;
677 (port&0x10)?slave->pin_state[3] = '1':slave->pin_state[3] = '0';
678
679 (port&0x20)?txt_pin_3->AppendText("SET"):txt_pin_3->AppendText("RESET")
680 ;
681 (port&0x20)?slave->pin_state[2] = '1':slave->pin_state[2] = '0';
682
683 (port&0x40)?txt_pin_2->AppendText("SET"):txt_pin_2->AppendText("RESET")
684 ;
685 (port&0x40)?slave->pin_state[1] = '1':slave->pin_state[1] = '0';
686
687 (port&0x80)?txt_pin_1->AppendText("SET"):txt_pin_1->AppendText("RESET")
688 ;
689 (port&0x80)?slave->pin_state[0] = '1':slave->pin_state[0] = '0';
690
691 return 0;
692 }
693
694 ////////////////////////////////////////////////////////////////////SLAVE funciones//////////////////////////////////////////////////////////////////
695
696 ////////////////////////////////////////////////////////////////////SLAVE LIST funciones//////////////////////////////////////////////////////////////////
697
698 //-----
699 // void slave_list(const Type_slave_list *list_slave)
700 //
701 // DESCRIPCION:
702 // Esta función es empleada para actualizar la lista de todos los
703 // esclavos
704 //
705 // ARGUMENTOS:
706 // Estructura de la lista de esclavos

```

```

699 //-----
700
701 void RadioTransceiverDlg::slave_list(const Type_slave_list *list_slave)
702 {
703     int i;
704
705     //hyperterminal->AppendText(wxString::Format("Number of slaves: %d\n\n",
706     //list_slave->number));
707
708     if(list_slave->number == 0)
709         hyperterminal->AppendText(wxString::Format("Slave list is empty.\n\n"));
710
711     cmb_slave_address->Clear();
712     cmb_recover_config->Clear();
713     cmb_slave_view->Clear();
714
715     for(i=0;i<list_slave->number;i++)
716         show_slave(&list_slave->list[i]);
717 }
718 //-----
719 // int duplicate_slave_name(const Type_slave_list *list_slave, const char*
720 // slave_name)
721 //
722 // DESCRIPCION:
723 //     Esta función es empleada para comprobar la duplicidad de nombres
724 //
725 // ARGUMENTOS:
726 //     Estructura de la lista de esclavos y nombre del esclavo a comprobar
727 //-----
728 int RadioTransceiverDlg::duplicate_slave_name(const Type_slave_list *
729 list_slave, const char* slave_name)
730 {
731     int i;
732
733     for(i=0;i<list_slave->number;i++)
734     {
735         if(!strcmp(list_slave->list[i].name, slave_name))
736             return 1;
737     }
738     return 0;
739 }
740 //-----
741 // void new_slave(Type_slave_list *list_slave)
742 //
743 // DESCRIPCION:
744 //     Esta función es empleada para añadir un esclavo nuevo
745 //
746 // ARGUMENTOS:
747 //     Estructura de la lista de esclavos

```

```
748 //-----  
749  
750 void RadioTransceiverDlg::new_slave(Type_slave_list *list_slave)  
751 {  
752     if(duplicate_slave_name(list_slave, txt_introduce_slave->GetLineText(0)  
753         ))  
754     {  
755         hyperterminal->AppendText(wxString::Format("\nName of the slave  
756             exists on BBDD yet!!!\n"));  
757         return;  
758     }  
759     if(detect_slave(&list_slave->list[list_slave->number],  
760         txt_introduce_slave->GetLineText(0))  
761         return;  
762     else  
763     {  
764         list_slave->number++;  
765         hyperterminal->AppendText(wxString::Format("\nSlave added  
766             successfully\n"));  
767     }  
768 }  
769  
770 //-----  
771 // void delete_slave(Type_slave_list *list_slave, const char* slave_name)  
772 //  
773 // DESCRIPCION:  
774 //     Esta función es empleada para eliminar un esclavo de la BBDD  
775 //  
776 // ARGUMENTOS:  
777 //     Estructura de la lista de esclavos y nombre del esclavo a eliminar  
778 //-----  
779 void RadioTransceiverDlg::delete_slave(Type_slave_list *list_slave, const  
780     char* slave_name)  
781 {  
782     int i;  
783     for(i=0;i<list_slave->number;i++)  
784     {  
785         if(!strcmp(list_slave->list[i].name, slave_name))  
786             break;  
787     }  
788     list_slave->number--;  
789     for(i;i<list_slave->number;i++)  
790         list_slave->list[i] = list_slave->list[i+1];  
791     slave_list(list_slave);  
792 }  
793  
794  
795 //-----
```

```

796 // Type_slave* const search_slave(Type_slave_list *list_slave, const char*
      slave_name)
797 //
798 // DESCRIPCION:
799 //     Esta función es empleada para localizar la estructura de un esclavo
800 //
801 // ARGUMENTOS:
802 //     Estructura de la lista de esclavos y nombre del esclavo a buscar
803 //-----
804
805 RadioTransceiverDlg::Type_slave* const RadioTransceiverDlg::search_slave(
      Type_slave_list *list_slave, const char* slave_name)
806 {
807     int i;
808
809     for(i=0;i<list_slave->number;i++)
810     {
811         if(!strcmp(list_slave->list[i].name, slave_name))
812             return &list_slave->list[i];
813     }
814 }
815
816 //-----
817 // void load_data_slaves(Type_slave_list *list_slave)
818 //
819 // DESCRIPCION:
820 //     Esta función es empleada para cargar la BBDD de esclavos
821 //
822 // ARGUMENTOS:
823 //     Estructura de la lista de esclavos donde almacenar los mismos
824 //-----
825
826 void RadioTransceiverDlg::load_data_slaves(Type_slave_list *list_slave)
827 {
828     int fd,i;
829     // Compruebo existencia de archivo y de registros
830     list_slave->number = num_reg_slaves("slaves.dat",sizeof(Type_slave));
831
832     if(list_slave->number)
833     {
834         hyperterminal->AppendText(wxString::Format("\nNumber of slaves: %d\n
              \n",list_slave->number));
835
836         if(list_slave->number >= MAX_SLAVE)
837         {
838             hyperterminal->AppendText(wxString::Format("There is no enough
              registers for listing.\n"));
839             return;
840         }
841         fd = open("slaves.dat", O_RDONLY);
842
843         if(fd == -1)
844         {

```

```

845     hyperterminal->AppendText (wxString::Format ("Error opening
           slaves.dat\n"));
846     hyperterminal->AppendText (wxString::Format ("%s\n", strerror(
           errno)));
847     return;
848 }
849
850 for(i=0;i<list_slave->number;i++)
851     read(fd, &list_slave->list[i], sizeof(Type_slave));
852
853 close(fd);
854 }
855 else
856     hyperterminal->AppendText (wxString::Format ("There is any slave on
           BBDD\n"));
857 }
858
859 //-----
860 // void save_data_slaves(const Type_slave_list *list_slave)
861 //
862 // DESCRIPCION:
863 //     Esta función es empleada para guardar la BBDD de esclavos
864 //
865 // ARGUMENTOS:
866 //     Estructura de la lista de esclavos que se desea almacenar
867 //-----
868
869 void RadioTransceiverDlg::save_data_slaves(const Type_slave_list *
           list_slave)
870 {
871     int fd,i;
872                                     //////////access//////// //Permission///
873     fd = open("slaves.dat", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR
           );
874     //O_WRONLY -> Solamente se permite operaciones de escritura
875     //O_CREAT  -> Crea el archivo si no existía
876     //O_TRUNC  -> Si el fichero existe, borra todo su contenido
877
878     //Permisos requeridos si el fichero es creado, ignorado de otro modo
879     //S_IRUSR  -> Propietario puede leer
880     //S_IWUSR  -> Propietario puede escribir
881
882     if(fd == -1)
883     {
884         hyperterminal->AppendText (wxString::Format ("Error opening slaves.
           dat\n"));
885         hyperterminal->AppendText (wxString::Format ("%s\n", strerror(errno)
           ));
886         return;
887     }
888
889     for(i=0;i<list_slave->number;i++)
890         write(fd, &list_slave->list[i], sizeof(Type_slave));

```

```

891
892     close(fd);
893 }
894
895 //-----
896 // int num_reg_slaves(const char *bbdd,int size_reg)
897 //
898 // DESCRIPCION:
899 //     Esta función es empleada para calcular el número de esclavos en la
900 //     BBDD
901 //
902 // ARGUMENTOS:
903 //     Nombre del fichero que contiene los datos y tamaño de un esclavo
904 //-----
905
906 int RadioTransceiverDlg::num_reg_slaves(const char *bbdd,int size_reg)
907 {
908     int fd;
909     int nbytes,nreg;
910
911     fd = open(bbdd, O_RDONLY);
912
913     if(fd == -1)
914     {
915         hyperterminal->AppendText(wxString::Format("Error opening slaves.
916             dat\n"));
917         hyperterminal->AppendText(wxString::Format("%s\n", strerror(errno)
918             ));
919         return 0;
920     }
921
922     lseek(fd,0,SEEK_END); //The file offset shall be set to the size of
923         the file
924     nbytes = tell(fd);
925     //Reports the offset of the current byte relative to the beginning
926     //of the file associated with the file descriptor.(The value obtained
927     //by
928     //lseek function)
929     close(fd);
930     nreg=nbytes/size_reg;
931
932     return nreg;
933 }
934
935 //-----
936 //SLAVE LIST funciones//-----
937
938 //-----
939 //GRAPHIC INTERFACE funciones//-----
940
941 /*
942 * cmb_slave_addressSelected
943 */
944
945 void RadioTransceiverDlg::cmb_slave_addressSelected(wxCommandEvent& event )

```

```
940 {
941     if(strlen(cmb_command->GetStringSelection().mb_str()) &&
942         strlen(cmb_data->GetStringSelection().mb_str()) &&
943         strlen(cmb_slave_address->GetStringSelection().mb_str()))
944         btn_send_frame->Enable(true);
945         //hyperterminal->AppendText(wxString::Format("hay seleccion\n"));
946         ;
947     else
948         btn_send_frame->Disable();
949         //hyperterminal->AppendText(wxString::Format("No hay seleccion\n"));
950 }
951 /*
952  * cmb_dataSelected
953  */
954
955 void RadioTransceiverDlg::cmb_dataSelected(wxCommandEvent& event )
956 {
957     if(strlen(cmb_command->GetStringSelection().mb_str()) &&
958         strlen(cmb_data->GetStringSelection().mb_str()) &&
959         strlen(cmb_slave_address->GetStringSelection().mb_str()))
960         btn_send_frame->Enable(true);
961         //hyperterminal->AppendText(wxString::Format("hay seleccion\n"));
962         ;
963     else
964         btn_send_frame->Disable();
965         //hyperterminal->AppendText(wxString::Format("No hay seleccion\n"));
966 }
967 /*
968  * cmb_commandSelected
969  */
970
971 void RadioTransceiverDlg::cmb_commandSelected(wxCommandEvent& event )
972 {
973     if(strlen(cmb_command->GetStringSelection().mb_str()) &&
974         strlen(cmb_data->GetStringSelection().mb_str()) &&
975         strlen(cmb_slave_address->GetStringSelection().mb_str()))
976         btn_send_frame->Enable(true);
977         //hyperterminal->AppendText(wxString::Format("hay seleccion\n"));
978         ;
979     else
980         btn_send_frame->Disable();
981         //hyperterminal->AppendText(wxString::Format("No hay seleccion\n"));
982 }
983 /*
984  * load_bbddClick
985  */
986
987 void RadioTransceiverDlg::btn_load_bbddClick(wxCommandEvent& event)
988 {
989     button_bbdd = 1;
```

```

990     btn_load_bbdd->Disable();
991     btn_save_bbdd->Enable(true);
992
993     load_data_slaves(&list_slave);
994     slave_list(&list_slave);
995
996     //Añadimos los pines que son salida
997     cmb_data->Append(pin_name[0]);
998     cmb_data->Append(pin_name[1]);
999     cmb_data->Append(pin_name[2]);
1000    cmb_data->Append(pin_name[3]);
1001
1002    //Añadimos los comandos para los pines que son salida
1003    cmb_command->Append(command_name[SETPIN]);
1004    cmb_command->Append(command_name[RESETPIN]);
1005 }
1006
1007 /*
1008  * btn_save_bbddClick
1009  */
1010
1011 void RadioTransceiverDlg::btn_save_bbddClick(wxCommandEvent& event)
1012 {
1013     save_data_slaves(&list_slave);
1014     hyperterminal->AppendText(wxString::Format("\nBBDD saved successfully\n"
1015         "));
1016 }
1017 /*
1018  * connect_comClick
1019  */
1020
1021 void RadioTransceiverDlg::btn_connect_comClick(wxCommandEvent& event)
1022 {
1023     if(IniciarCOM(path_com))
1024     {
1025         AbrirCOM(path_com);
1026         btn_connect_com->Disable();
1027         btn_close_com->Enable(true);
1028
1029         if(!button_bbdd)
1030             btn_load_bbdd->Enable(true);
1031
1032         btn_recover_config->Enable(true);
1033         btn_search_slave->Enable(true);
1034         btn_delete_slave->Enable(true);
1035         btn_temperature->Enable(true);
1036         cmb_slave_address->Enable(true);
1037         cmb_command->Enable(true);
1038         cmb_data->Enable(true);
1039         cmb_slave_view->Enable(true);
1040         cmb_recover_config->Enable(true);
1041

```



```

1091 //hyperterminal->AppendText(wxString::Format("->Slave name %s\n",
1092 slave_struct->name));
1093 //hyperterminal->AppendText(wxString::Format("->Pin name %s\n",cmb_data
1094 ->GetStringSelection().mb_str()));
1095 //hyperterminal->AppendText(wxString::Format("->Command name %s\n",
1096 cmb_command->GetStringSelection().mb_str()));
1097
1098 for(contador = 0;contador <= 10; contador++)
1099 {
1100     if(!tx_rx_frame(slave_struct->address,command_number[number_command
1101 ], pin_number[number_pin])
1102         goto L1;
1103     }
1104     hyperterminal->AppendText(wxString::Format("\n->Error, the operation
1105     couldn't be set\n"));
1106     return;
1107 L1:
1108     if(!strcmp(command_number[number_command],command_number[SETPIN]))
1109     {
1110         slave_struct->pin_state[number_pin] = '1';
1111         hyperterminal->AppendText(wxString::Format("\n->Pin %s SET it
1112         successfully\n", &pin_name[number_pin][0]));
1113     }
1114     else if(!strcmp(command_number[number_command],command_number[RESETPIN
1115 ]))
1116     {
1117         slave_struct->pin_state[number_pin] = '0';
1118         hyperterminal->AppendText(wxString::Format("\n->Pin %s RESET it
1119         successfully\n", &pin_name[number_pin][0]));
1120     }
1121     slave_pin_view(slave_struct);
1122 }
1123
1124 /*
1125 * btn_search_slaveClick
1126 */
1127
1128 void RadioTransceiverDlg::btn_search_slaveClick(wxCommandEvent& event)
1129 {
1130     if(strlen(txt_introduce_slave->GetLineText(0)))
1131         new_slave(&list_slave);
1132     else
1133         hyperterminal->AppendText(wxString::Format("There is any name
1134         introduced\n"));
1135 }
1136
1137 /*
1138 * btn_delete_slaveClick
1139 */
1140
1141 void RadioTransceiverDlg::btn_delete_slaveClick(wxCommandEvent& event)
1142 {
1143     if(strlen(cmb_recover_config->GetStringSelection().mb_str()))

```

```
1135     {
1136         delete_slave(&list_slave, cmb_recover_config->GetStringSelection().
            mb_str());
1137         hyperterminal->AppendText(wxString::Format("Slave deleted
            successfully\n"));
1138     }
1139     else
1140         hyperterminal->AppendText(wxString::Format("There is any slave
            selected\n"));
1141 }
1142
1143 /*
1144  * btn_recover_configClick
1145  */
1146
1147 void RadioTransceiverDlg::btn_recover_configClick(wxCommandEvent& event)
1148 {
1149     if(strlen(cmb_recover_config->GetStringSelection().mb_str()))
1150         restablish_config(search_slave(&list_slave, cmb_recover_config->
            GetStringSelection().mb_str()));
1151     else
1152         hyperterminal->AppendText(wxString::Format("There is any slave
            selected\n"));
1153 }
1154
1155 /*
1156  * cmb_slave_viewSelected
1157  */
1158
1159 void RadioTransceiverDlg::cmb_slave_viewSelected(wxCommandEvent& event )
1160 {
1161     if(slave_pin_view(search_slave(&list_slave, cmb_slave_view->
            GetStringSelection().mb_str())))
1162         hyperterminal->AppendText(wxString::Format("Read Pins couldn't be
            measured\n"));
1163 }
1164
1165 /*
1166  * btn_temperatureClick
1167  */
1168
1169 void RadioTransceiverDlg::btn_temperatureClick(wxCommandEvent& event)
1170 {
1171     Type_slave* slave_struct;
1172
1173     if(strlen(cmb_recover_config->GetStringSelection().mb_str()))
1174     {
1175         slave_struct = search_slave(&list_slave, cmb_recover_config->
            GetStringSelection().mb_str());
1176         if(temperature_slave(slave_struct->address))
1177             hyperterminal->AppendText(wxString::Format("Temperature couldn'
            t be measured\n"));
1178     else
```

```
1179         hyperterminal->AppendText (wxString::Format ("Temperature
1180             measured successfully\n"));
1181     }
1182     else
1183         hyperterminal->AppendText (wxString::Format ("There is any slave
1184             selected\n"));
1185 }
1186
1187 //////////////////////////////////GRAPHIC INTERFACE funciones////////////////////////////////////
```

Código F.2: Programa principal de la aplicación para PC

# Índice terminológico

## A

ADC, 31, 54  
ANAREN, 86

## B

Batería, 49, 89, 93, 101, 111  
BBDD, 116  
Bluetooth, 47  
BOM, 85, 97  
Bootloader, 57, 62, 93, 112

## C

Cortex-M0, 53, 54, 65, 86, 132  
CRC, 30, 40, 42–44, 88, 124

## E

EEPROM, 57, 88

## F

FEC, 44  
FIFO, 31, 38, 78, 89  
FTDI, 87, 91, 104, 121

## G

GPIO, 58, 59, 64, 71, 72

## M

Makefile, 107, 113  
MinGW, 114  
MSK, 30, 45, 53

## N

NVIC, 54, 57, 64, 67

## P

PCB, 85–87, 92, 97, 119  
PLL, 58, 62  
PWM, 75, 76

## R

Reset, 34, 51, 56, 59, 64, 93  
ROM, 57, 62  
RS232, 88  
Rutado, 97, 98, 121

## S

SCB, 54, 62  
Sleep, 59, 66, 111  
SmartRF, 49  
SPI, 30, 36, 78, 95  
SRAM, 54, 57, 62, 66  
SSP, 54, 58, 64, 78  
Sync Word, 30, 40

## U

USART, 54, 58, 64, 81, 87  
USB, 88, 91, 104, 121

## W

WDT, 58, 62  
Whitening, 43  
Wi-Fi, 47  
wxWidgets, 114

# Bibliografía

- [1] Manual de Usuario. *CC2500 Low-Cost Low-Power 2.4 GHz RF Transceiver*.  
<http://www.tij.co.jp/jp/lit/ds/swrs040c/swrs040c.pdf>.
- [2] Texas Instruments. *CC2500 Errata Notes*.  
<http://www.ti.com/lit/er/swrz002e/swrz002e.pdf>.
- [3] José I. Alonso Montes y Pablo Almorox González y José A. Rodríguez Salazar. Wi-Fi: El diferente uso del espectro en EEUU y Europa. *Colegio Oficial de Ingenieros de Telecomunicación, COIT*, 149:59–63, Marzo 2005.
- [4] IEEE Computer Society. Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs). *Institute of Electrical and Electronics Engineers, IEEE*, pages 45–46, June 2005.
- [5] Texas Instruments. *SmartRF<sup>TM</sup> Studio 7 v1.16.1 (Rev. AA)*.  
<http://www.ti.com/lit/zip/swrc176>.
- [6] Texas Instruments. *SmartRF<sup>TM</sup> Studio 7 Overview*.  
<http://www.ti.com/lit/ug/swru195b/swru195b.pdf>.
- [7] Texas Instruments. *SmartRF<sup>TM</sup> Studio 7 Hands-on User Guide and Tutorial*.  
<http://www.ti.com/lit/ug/swru194b/swru194b.pdf>.
- [8] ARM. *ARM Cortex-M0 User Guide*.  
<http://www.ti.com/lit/er/swrz002e/swrz002e.pdf>.
- [9] Manual de Usuario. *UM10518 LPC11Exx User manual - NXP Semiconductors*.  
[http://www.nxp.com/documents/user\\_manual/UM10518.pdf](http://www.nxp.com/documents/user_manual/UM10518.pdf).
- [10] ARM. *ARM Cortex-M Series*.  
<http://www.arm.com/products/processors/cortex-m/index.php>.

- [11] Farnell. *Oficial Website*.  
<http://es.farnell.com/>.
- [12] FTDI Chip. *Oficial Website*.  
<http://www.ftdichip.com/>.
- [13] Labcenter Electronics. *Proteus Isis Schematic Capture*.  
[http://www.labcenter.com/products/pcb/schematic\\_intro.cfm](http://www.labcenter.com/products/pcb/schematic_intro.cfm).
- [14] Labcenter Electronics. *Ares PCB Layout Software*.  
[http://www.labcenter.com/products/pcb/pcb\\_intro.cfm](http://www.labcenter.com/products/pcb/pcb_intro.cfm).
- [15] Itead Studio. *Oficial Website*.  
<http://imall.iteadstudio.com/>.
- [16] SourceForge.net. *ISP Flasher*.  
<http://sourceforge.net/projects/lpc21isp/>.
- [17] wxDev C++ Developers. *Entorno Integrado de Desarrollo wxDev-C++*.  
<http://wxdsgn.sourceforge.net/>.
- [18] Embedded Artists. *LPC2103 Education Board*.  
<http://www.embeddedartists.com/node/55>.

