Capítulo 5

Aplicación Práctica. Transmisión de Voz en Banda ISM

En este capítulo se va a describir en profundidad la solución adoptada. Como se ha comentado, el proyecto se realiza en la placa de depuración del Kit de Desarrollo de Texas Instruments, SmartRF®04EB, ésta permite acceder y configurar los registros del CC1110 y del CC2510 a través del IAR Embedded Workbench. En el Capítulo 3 se hecho una descripción más profunda del aspecto Hardware de la misma y en el Capítulo 4 se detallan los conceptos para manejar el entorno Software.

Antes de realizar la primera toma de contacto con el Kit de Desarrollo, se elabora una hoja de ruta o pseudocódigo que es base para todas las modificaciones posteriores, y aunque se ha modificado varias veces, es punto de partida para todo el desarrollo que se ha seguido hasta la consecución del objetivo. El cual se ha ajustado sensiblemente con el resultado final. Esta hoja de ruta tiene marcados 7 hitos que se desglosan punto por punto. A continuación se profundiza en la solución que se ha adoptado para resolver cada hito. Es seguro que hay otras formas de alcanzar la solución final, ya que se proporciona bastante libertad al desarrollador, ofreciendo muchas posibles variables. Se ha justificado la elección en cualquier caso. Tras desglosar la

solución se describe la configuración del IAR Workbench para que el código se implemente correctamente, ya que existen múltiples opciones por defecto a la hora de crear un nuevo workspace que hay que modificar.

5.1. Configuración en Modo Maestro

En primer lugar se presentan los hitos a resolver para dar una visión global y posteriormente se describe uno por uno:

1. Configurar la entrada de audio como analógica.
2. Configurar la entrada de audio como entrada al ADC.
3. Activar y configurar el ADC para tener una conversión que permita una calidad en la conversación aceptable.
4. Almacenar la conversión en un canal DMA.
5. Enviar la información del canal DMA por el canal radio correspondiente.
6. Recibir el audio digital por el canal de radiofrecuencia y almacenarlo en un canal DMA.
7. Enviar la información de audio digital por la salida de audio.

A continuación se presenta la solución de cada uno de los puntos anteriores, para el modo Maestro, explicando los detalles más relevantes del código utilizado. Posteriormente se realizará el proceso similar con el modo Esclavo ya que presenta considerables variaciones con el primero.

1. Configurar la entrada de audio como analógica. El bit correspondiente del registro ADCCFG debe establecerse como 1. Se hace con una sóla línea de código. Se puede confirmar el valor en el menú I/O Register del editor.

2. Configurar la entrada de audio como entrada al ADC. Esto es inmediato DIRP0_0=0 configura como entrada el puerto 0_0, destinado al micrófono. Además de esto en la primera fase del programa principal se describe una función cuya única misión es configurar la funcionalidad de los puertos de entrada y salida del SoC. Además de la entrada del micrófono, en esta fase se configuran los LEDs, y la salida de

audio hacia los altavoces. Cabría la posibilidad de utilizar botones e incluso el joystick. En el paquete de programas HAL [1] se incluyen funciones que facilitan su uso. En este proyecto no se utilizan, ya que el interlocutor no tiene que pulsar ningún botón, sólo hablar por el micrófono para que la voz digitalizada se escuche por la salida de audio de la otra SmartRF04EB.

Como se muestra en la Tabla 5.1, además de a 8 KHz hay otras velocidades de muestreo posibles, dentro de los límites de los SoC CC1110 y CC2510, de la máxima velocidad de datos (500 kbps), longitud de paquete (255 bytes), y la conversión con tiempo de ADC. El tiempo de conversión ADC y, por extensión, la frecuencia de muestreo máxima es una función de la resolución seleccionada.

Resolución del ADC (bits)	**Tiempo de conversión (μs)**	**Máxima velocidad de muestreo (kHz)**
7	18,4	54,3
9	33,12	30,2
10	62,56	16,0
12	121,44	8,2

Tabla 5.4: Resolución ADC

Además, el CC2510 requiere 88,4μs de transición desde el estado Idle a los estados Receptor On (Rx On) o al estado Transmisor On (Tx On), y 721μs para calibrar el PLL.

3. Activar y configurar el ADC para tener una conversión que permita una calidad en la conversación aceptable. El audio se muestrea, acorde con la Recomendación G.711 de la ITU-T para frecuencias vocales a una tasa de 8KHz (125μs), usando el ADC del SoC con una resolución de 12 bits. Estos tres primeros puntos realmente se condensan mucho en un programa. "*tw_adc.c"*, compuesto por una única función que no tiene parámetros de entrada ni de salida, sólo ejecuta los comandos necesarios para configurar el funcionamiento del ADC. Además de esto hay que determinar los parámetros del DSM (Delta Sigma Modulator), de los cuales el más importante la tasa de muestreo. Se activará una interrupción del Timer 1 que copia los bits a la salida del ADC a los registros T1CC1H:T1CC1L. En la Figura 5.1 se pueden

obtener las configuraciones posibles de los registros del T1CC0 para obtener la tasa de muestreo deseada.

Sample Rate	T1CC0H	T1CC0L
8 kHz @ 24 MHz	0x0B	0xB7
8 kHz @ 26 MHz	0x0C	0xB1
16 kHz @ 24 MHz	0x05	0xDB
16 kHz @ 26 MHz	0x06	0x59
48 kHz @ 24 MHz	0x01	0xF3
48 kHz @ 26 MHz	0x02	0x1D
64 kHz @ 24 MHz	0x01	0x76
64 kHz @ 26 MHz	0x01	0x96

Figura 5.13: Posibles valores de los registros T1CC0

Los parámetros del DSM se sintetizan en una función dentro de un programa independiente, *"tw_dsm.c"*. Este será invocado desde el programa principal, al igual que el *"tw_adc.c"*, en la parte del mismo denominada Inicializaciones. Estas funciones están escritas por programadores de Texas Instruments y forman parte de paquetes de ayuda, lo cual simplifica la tarea del desarrollador, que debe configurar los parámetros acorde a sus necesidades.

Además de esto, en la primera fase del programa principal se activan las interrupciones, se programan los canales que se van a utilizar en FHSS. Lo cual es relativamente rápido ya que a partir del canal 0 se utilizarán 4 canales de 250KHz con una separación entre portadoras de 3,25MHz. En la actualidad los cuatro canales utilizados son 0, 13, 26 y 39. La frecuencia de base se establece en 2406 MHz para el CC2510 y en 433 MHz para el CC1110 y la separación entre canales de 250 KHz, por lo que los canales 0, 13, 26 y 39 corresponden a las frecuencias de 2406, 2409.250, 2412.500 y 2415.750 MHz respectivamente. Análogamente para el CC1110 las frecuencias serían 433.000 MHz, 436.250 MHz, 439.500 MHz y 442.750 MHz. La forma de cambiar de uno a otro será mediante una variable ActiveChIdx. Esta variable provoca que se vaya saltando desde el canal 0 hasta el 3 de forma rotativa. Tras activar un nuevo canal, hay que calibrarlo, esto se hace de forma trasparente al programador mediante la función SCAL() definida en la librería *"hal.h"*.

```
// FHSS
#define FH_ACTIVE_TABLE_LEN 4
#define CHANNEL_1   0   // 433.000 MHz
#define CHANNEL_2  13   // 433 + 13*.25 = 436.250 MHz
#define CHANNEL_3  26   // 433 + 26*.25 = 439.500 MHz
#define CHANNEL_4  39   // 433 + 39*.25 = 442.750 MHz
```

El programa no comienza hasta que no se activa la bandera "TramaAudioLista" por el controlador de audio, impulsado por una rutina de interrupción que se ejecuta una vez cada 125 μs.

Previamente se reserva memoria para dos buffer de entrada de audio (audioIn[i]) y dos de salida (audioOut[i]), podrían reservarse alguno más o incluso menos pero en caso de producirse algún error conviene al menos tener dos. A la hora de almacenar los datos se hará de forma alterna, es decir, una trama en cada buffer. Además de esto, en esta primera fase se inician las funciones del display LCD utilizando el programa *"lcd.c"*, las de la etapa radio mediante initRF(freq) definida en el programa *"tw_rf.c"* y se permiten todas las interrupciones, lo cual no ocurrirá siempre porque habrá funciones que deshabilitan las interrupciones, siendo independientes de que se produzcan en cualquier otro caso.

Como se ha descrito brevemente en la Introducción, se utilizan dos SoC configurados como Maestro y Esclavo respectivamente. Esto es debido a que uno de los dos, el Maestro envía una trama faro para que el Esclavo sea capaz de emparejarse. La sincronización es muy importante, ya que el esclavo debe conocer en todo momento en que canal está emitiendo el Maestro. El Esclavo conoce los canales FHSS porque se envían en cada paquete transmitido por el Maestro. En la Figura 5.2 se muestra un diagrama esquemático con los elementos utilizados hasta ahora y el proceso que siguen los datos de voz desde el micrófono.

Para cada modo de funcionamiento se implementa una estructura de paquetes distinta. Será necesario empaquetar los datos de audio a la salida del ADC en una estructura definida, además se envían unas cabeceras y unas colas que añade el

transmisor automáticamente con los indicadores LQI y RSSI, aunque estos últimos no se utilizan para nada concreto en este proyecto.

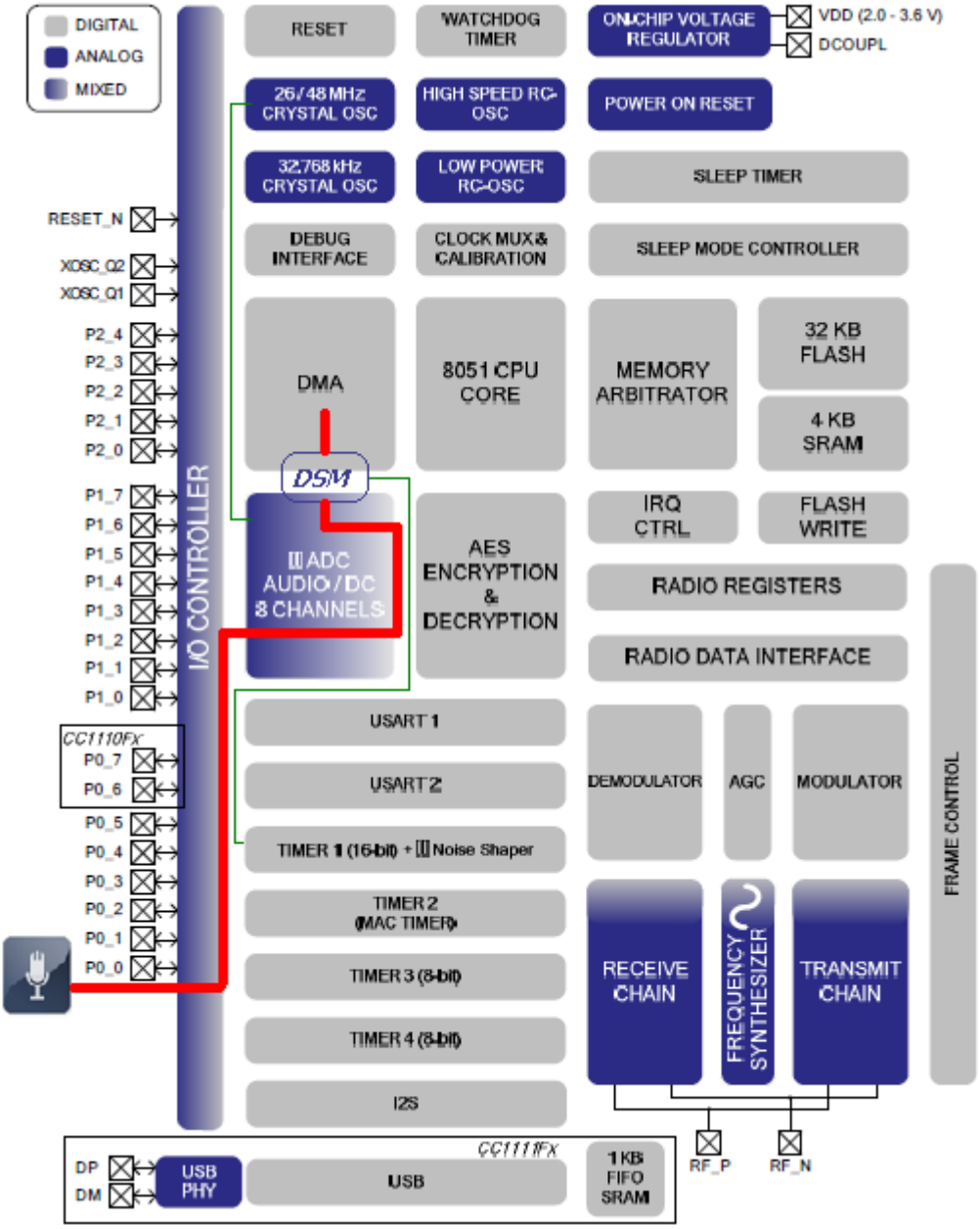

Figura 5.14: Ruta de los datos de audio desde el micro hasta el canal DMA

4. Almacenar la conversión en un canal DMA. La conversión se realiza en el ADC con una resolución de 12 bits y se muestrea en el DSM cada 125µs (8KHz). Por lo que a la salida del DSM se tienen los datos de audio digitalizados listos para enviar a la etapa de radio.

El procedimiento de empaquetado es sencillo, se definen cuatro estructuras, dos para el maestro y dos para el esclavo, transmisión y recepción respectivamente. Estos campos se irán completando a lo largo del bucle correspondiente a cada modo en el programa principal.

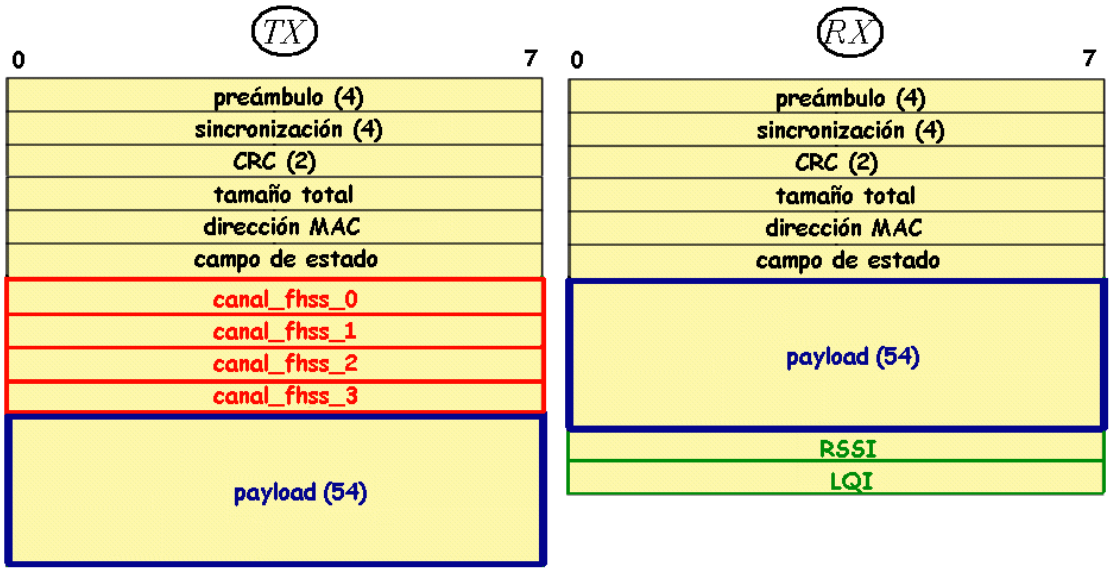

Figura 5.15: Estructura de los paquetes en modo Maestro

El paquete a transmitir por el maestro se rellena en orden, es decir, primero un byte para el tamaño total, un byte de dirección MAC, un byte con el campo de estado y por último se cargan en el payload las 54 muestras de audio procedentes del buffer audioIn que esté activo el 0 o el 1. Además de esto, la capa física PHY, añade en el transmisor 4 bytes de preámbulo, 4 bytes de sincronización y 2 bytes del CRC. Se espera que la calibración del canal activo este completada, ya sólo resta enviar los datos por la etapa de radiofrecuencia.

Los gastos generales de cabecera se pueden minimizar aumentando al máximo la longitud del paquete. Sin embargo, paquetes más largos tienen mayor probabilidad de ser dañados durante la transmisión que si el paquete fuese más corto. Además, la latencia de audio y el tiempo sin reproducir el audio en el receptor si un paquete se pierde aumentaría. Por lo que se ha buscado una solución de compromiso entre ambos.

5. Enviar la información del canal DMA por el canal radio correspondiente.

Se envía el paquete haciendo uso de la función rfSendPacket() definida en el programa *"tw_rf.c"*.

Cada 54 muestras (6,75ms) los datos se envían al buffer de transmisión mediante la función dmaToRadio(), que necesita como parámetros el tamaño total de datos a enviar y un puntero a los datos que se quieren mandar a la etapa RF, es decir un puntero a la estructura de datos de transmisión del Maestro.

6. Recibir el audio digital por el canal de radiofrecuencia y almacenarlo en un canal DMA. Tras enviar los datos, el Maestro permanece a la escucha para recibir los datos. Necesita un tiempo similar para recibir que el que empleó para enviar, 6,75ms. Esto se controla mediante un contador descendente (Timer2). El Maestro desconoce si el Esclavo ha alcanzado el estado emparejado y por tanto si le enviará un paquete de otras 54 muestras, por lo que permanece a la escucha.

Esta etapa se realiza mediante la función rfReceivePacket() que necesita un puntero a la estructura donde almacenar los datos recibidos, el tiempo que permanece a la escucha y el tamaño total de los datos a recibir. Además se modificará una variable de estado del paquete recibido (rxPacketStatus), la cual indica si el paquete se ha recibido correctamente o si ha habido algún error, de que tipo es. Esto en voz no tiene mucho sentido porque no hay programadas retransmisiones, pero conocer cual es el fallo puede ser útil en caso de enviar datos.

En el caso de que el Maestro no recibiese datos del esclavo desactiva el buffer de recepción y apaga el LED verde, esto indica que no se ha emparejado correctamente con el Esclavo porque hubo algún error. En caso de éxito, igualmente desactiva el buffer de recepción mientras copia los datos estructurándolos acorde con el paquete receptor que se muestra en la Figura 5.3. Por defecto en transmisión siempre se envían los bytes RSSI y LQI, por lo que se añade una cola al paquete en recepción para almacenar el contenido de estos campos.

En el Anexo I se muestra el pseudocódigo que sigue el SoC configurado como Maestro. Debido a la naturaleza del flujo de audio, el proceso descrito hasta ahora no se basa en ningún protocolo estándar, aunque se muestrea acorde a las recomendaciones de la ITU [21] y se hacen uso de las funciones definidas por Texas Instruments.[1]

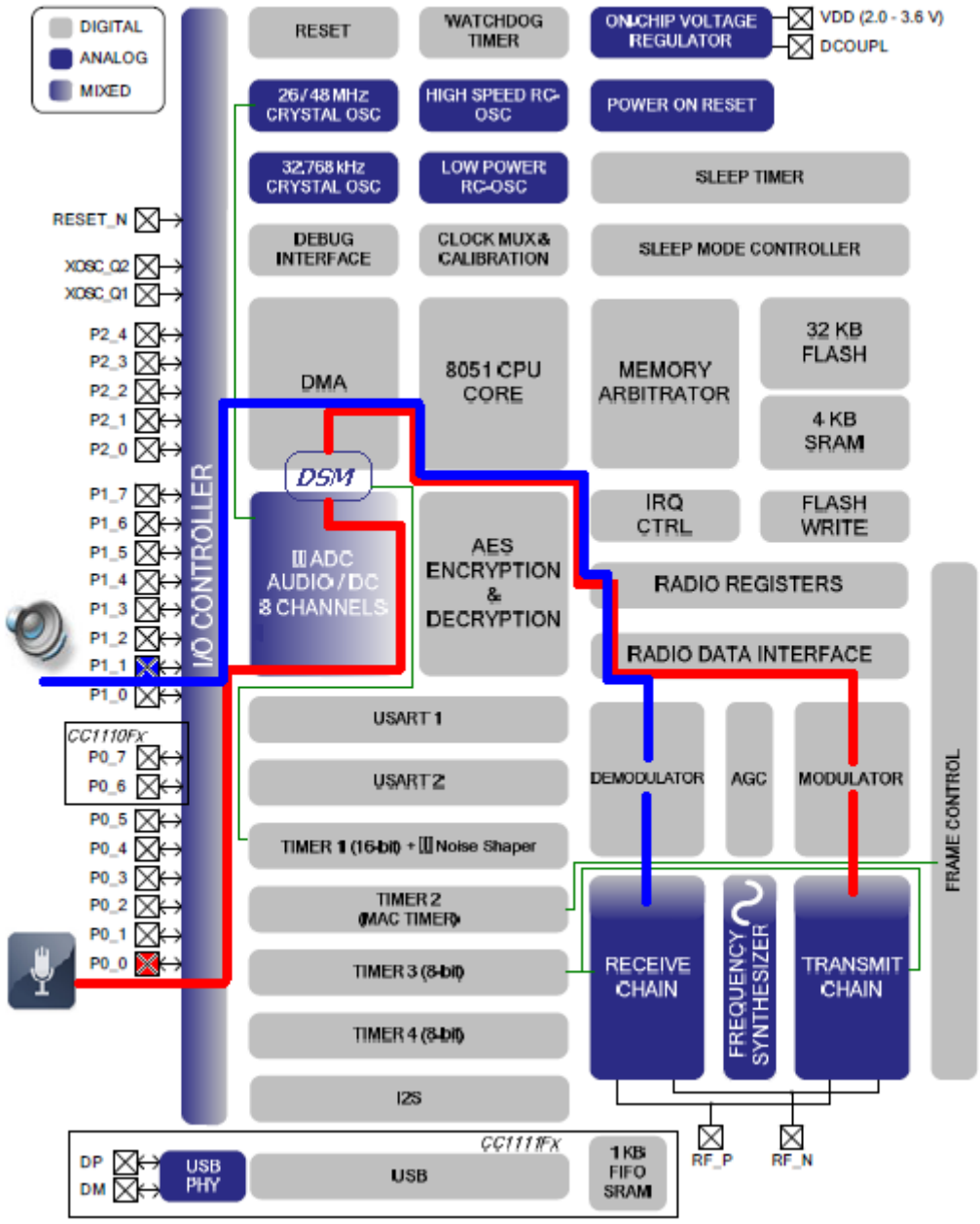

Figura 5.16: Esquemático de los datos de voz

7. Enviar la información de audio digital por la salida de audio. Este es el último paso. Sólo resta coger los datos del audio digital del campo payload del paquete recibido y proceder a almacenarlos en un buffer (audioOut). Mediante la función TIMER1_SET_PWM_PULSE_LENGTH() definida en la librería *"hal.h"* se saca por el puerto P1_1 la voz digitalizada.

Este es el proceso que se sigue en el Modo Maestro, varía en algunos aspectos con el que se lleva a cabo en el modo Esclavo que se describirá a continuación. El esquema final se puede observar en la Figura 5.4. De los múltiples módulos que posee el EM, se emplean 2 puertos, el ADC, los tres contadores principales, basados en el oscilador de 26MHz

5.2. Configuración en Modo Esclavo

A continuación se va a describir paso por paso el proceso que siguen los datos de audio en el SoC considerado configurado como Esclavo. Se destacarán las diferencias con el modo Maestro, ya que es más complejo.

1. Configurar la entrada de audio como analógica.
2. Configurar la entrada de audio como entrada al ADC.
3. Activar y configurar el ADC para tener una conversión que permita una calidad en la conversación aceptable.
4. Almacenar la conversión en un canal DMA.

El proceso de configurar el ADC, el DSM y el almacenamiento en los canales DMA son similares a los del Maestro, de hecho la fase de inicializaciones son comunes para ambos.

Este modo se basa en un Contador de Trama, T2. Se trata de un temporizador de cuenta atrás, que se inicializa a un valor de 229 tics, correspondiente a un período de 6,75 ms, tras recibir un paquete del Maestro. Al comienzo además se rellena el campo

5. Recibir el audio digital por el canal de radiofrecuencia y almacenarlo en un canal DMA. El Esclavo tiene dos estados de operación; “Escuchando al Maestro” y “Recibiendo Paquete”. Escuchando al Maestro, al iniciar o cuando pierde el sincronismo, esto se considera cuando se pierden más de 4 paquetes. La entrada o permanencia en este estado se controla mediante una variable booleana EsperandoBeaconFrame, puede tomar dos valores TRUE o FALSE. Por defecto, es decir al inicio se configura como FALSE y sólo se conmutará cuando se reciba

correctamente un paquete. Cuando se encuentra en este estado, ejecuta una función distinta que si estuviese sincronizado, ListenforMaster() definida en el programa *"tw_rf.c"*.El motivo por el que se utiliza está función se debe a que se diferencia de la recepción normal en que sale inmediatamente si un paquete se recibe correctamente antes de que se termine el tiempo de recepción. Permanece escuchando únicamente el canal 0. El tiempo destinado a recepción por la etapa de RF está controlado por el Timer 3 en la rutina de control de acceso al medio *"tw_mac.c"*. Como interfaz para el usuario se activa el LED naranja mientras permanezca en esta fase, adicionalmente se podría sacar un mensaje por pantalla.

En caso de que se reciba correctamente el paquete, sale de ListenforMaster() cambiando el valor de EsperandoBeaconFrame y pasa al estado Recibiendo Paquete. En esta fase se enciende el LED Verde y ejecuta la función rfReceivePacket() también está recogida en el programa *"tw_rf_cc1110.c"*, debido a que la configuración de los registros de RF son distintos es necesario modificarlos a la hora de inicializar la etapa de RF. Esta función se diferencia de la anterior en que ésta termina cuando acaba el contador de RX Timer 3, mientas que la anterior sale automáticamente si se recibe correctamente un paquete. Al igual que la anterior devuelve el estado del paquete recibido, TIMEOUT_ERROR si no encuentra la bandera de comienzo, PKT_ERROR si la longitud del paquete es distinta de la esperada, CRC_ERROR o PKT_OK si todo ha ido bien.

6. Enviar la información de audio digital por la salida de audio.

A continuación mirando esta variable de estado del paquete recibido se sabe si se ha producido algún error, lo cual implica que se apaga el LED Verde y se marca el buffer de recepción como no disponible y se aumenta el número de paquetes perdidos y se comprueba si se han superado el límite de 4 paquetes perdidos. En caso de que se supere, se resetea el contador de paquetes perdidos y vuelve al estado EsperandoBeaconFrame, entrando en modo re-sincronismo. Si no hubo error al recibir el paquete se resetea el número de paquetes perdidos y se enciende el LED Verde como indicador de que todo funciona correctamente. Destacar que este proceso ya es independiente de la etapa de RF y el SoC ya puede estar operando con el paquete que va a enviar al Maestro.

Hay varias maneras de actuar en caso de detectar que se ha perdido un paquete, la simple es silenciar el audio durante la duración del paquete perdido y la compleja podría ser, por ejemplo, el intercalado de datos a través de tres paquetes consecutivos. La idea detrás del enfoque de datos entrelazados es que si un paquete se pierde, los datos que faltan pueden ser re-creados a través de interpolación, basada en las muestras correctamente recibidas en los demás paquetes. En este proyecto se ha optado por un enfoque simple de silenciamiento, el cual se ha visto aceptable tras las pruebas experimentales. Si se pierde un paquete, el audio es simplemente silenciado por la duración del paquete. Teóricamente la interpolación de los datos que falten, usando cualquier otro algoritmo práctico conduce invariablemente a "clics" y "pops" audibles por el usuario. Otra ventaja de este enfoque simple es que la latencia, el tiempo de retardo desde que el audio se muestrea por el ADC y cuando la muestra es reproducida se reduce al mínimo. Para esta implementación, la latencia es de 13,5ms, un tiempo lo suficientemente corto que no es detectable por el oído humano.

Para comprobar que la palabra de sincronización, SYNC, ha sido enviada/recibida, se revisa el bit 3 ("SFD") del registro PKTSTATUS para ver si se ha activado ("1"). Este bit se activa cuando se encuentra la palabra SYNC y se desactiva ("0") tras recibir el paquete. Si el bit no está establecido, se supone que el paquete entrante se ha perdido y se apaga el receptor.

Por último antes de mandar los datos por la salida de audio P1_1, el SoC marca como no disponible el buffer audioOut(x) que esté activo, el cual vendrá indicado por activeOut, para que no pueda activarse la rutina que copia los datos de este buffer a la salida de PWM, TIMER1_SET_PWM_PULSE_LENGTH(). Copia los datos del campo payload que ha recibido del Maestro y lo pone disponible, por último conmutará activeOut.

El uso de la macro TIMER1_SET_PWM_PULSE_LENGTH(channel,value) , es simple, hay que pasarle como parámetros el canal y el valor que se quiere sacar por el puerto P1_1. El canal se configura como 1, valor por defecto. El valor es el tamaño del pulso como una palabra de 16 bits. Está definida en la librería *"hal.h"*. Está controlada por las interrupciones del Timer 1 (Contador para la Tasa de Muestreo). Es en esta etapa

final donde se implementa la realimentación para que el interlocutor se escuche a si mismo mezclando en el buffer de salida las muestras recibidas y las que salen del DSM.

Cuando termina el contador Timer 2, el contador de trama, pasa a transmitir datos al Maestro. Continúa rellenando los campos del paquete a enviar, campo de estado y el contenido de la estructura destinada a los paquetes copiando el buffer audioIn apuntado por activeIn. Previamente ha rellenado el tamaño y la dirección MAC

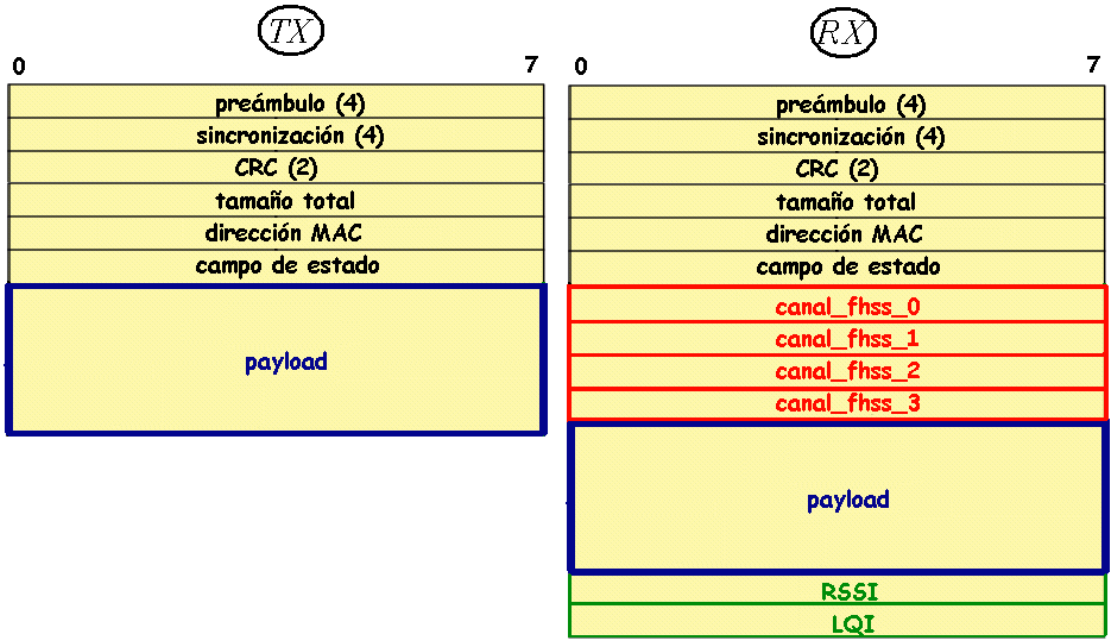

Figura 5.17: Estructura de los paquetes en Modo Esclavo

7. Enviar la información del canal DMA por el canal radio correspondiente.

Una vez lleno el paquete a transmitir lo envía a la etapa de RF mediante la función rfSendPacket(), para lo cual fuerza mediante SIDLE() el estado ocioso en recepción, esta función está contenida en la librería *"hal.h"*.

Tras enviar el paquete configura el siguiente canal FHSS mediante la función setChannel() contenida en el paquete de funciones de la etapa RF, y mediante el indicador de canal activo de FHSS. Este se va incrementando uno a uno hasta llegar al último que vuelve al primero.

Como se observa en la Figura 5.5 en la estructura para recepción se incluyen los campos que envía el Maestro en transmisión, los canales FHSS y los bytes RSSI y LQI.

Finalmente a continuación se resumen las características más importantes del funcionamiento los SoC:

- Muestras de audio por paquete: 54
- Frecuencia de muestreo del audio: 8 kHz
- Velocidad de datos de audio: 64 kbps en cada dirección (calidad de la voz)
- Velocidad de datos Radio: 250 kbps
- Longitud del paquete (excluyendo sincronización, CRC y preámbulo): 61 bytes
- Longitud total del paquete (incluyendo sincronización, CRC y preámbulo): 71 bytes
- Tiempo entre estado Idle a TX ON y de Idle a RX ON: 89 us
- Resolución: 12 bits
- Técnica de Espectro ensanchado: Salto de frecuencia (4 canales)
- Tasa salto: 148 saltos / seg
- Modulación: MSK
- Potencia de salida RF: 0 dBm (1 mW)
- Control Manual de volumen (altavoz): La tarjeta utiliza un Texas Instruments TPA122 Mono amplificador de energía. El volumen es controlado a través de un potenciómetro de giro

Para la consecución de estos eventos, se han ejecutado programas específicos, apoyados en conjuntos de funciones y librerías suministradas por el fabricante. Son de gran ayuda los manuales de usuario de las capas de abstracción HAL y CUL, propias de la pila de software TIMAC [1].

El siguiente paso es conectar los Módulos de Evaluación a la placa, en los conectores adecuados de la esquina superior derecha, a continuación se le acopla la antena. Lo último para tener listo el equipo es conectar mediante USB la placa SmartRF04EB a un PC con el que se pueda programar el SoC.

La herramienta Software IAR Embedded Workbench, recomendada por el fabricante, es la elegida para cargar en la memoria del SoC el programa que se ha desarrollado. Se describirá con mayor profundidad en el Capítulo 4 Entorno Software.

El código implementado consta de 9 programas y 6 librerías agrupados en un Workspace. El comportamiento de cada SoC se escribe en un programa principal denominado *"com_main_ccxx10.c"* donde xx se refiere a que son distintos para cada SoC y una librería principal *"voz.h"* para configurar el modo de funcionamiento comentando/descomentando una línea de código (#define MASTER). Es en esta librería donde se definen las estructuras, el comportamiento de los Temporizadores T1 (tasa de muestreo), T2 (contador de trama) y T3 (tiempo de espera en tx/rx en la etapa RF), los LEDs utilizados, los canales DMA, la tabla de canales FHSS y se hace referencia al resto de funciones prototipo de la capa de adaptación HAL: *"tw_dma.c", "tw_adc.c", "tw_rf.c," "tw_mac.c", "tw_interrupt.c", "tw_dsm.c", "lcd.c"*, las librerías *"hal.h", "hal2.h", "ioccxx10.h", "lcd.h", "RF04EB.h"* y las variables globales externas.

5.3. Parámetros de configuración del Workspace

En primer lugar es recomendable compilar cada programa individualmente para depurar posibles errores, tras esto mediante la opción Rebuild All del Workspace se puede compilar en conjunto y crear el ejecutable que se almacenará en la carpeta Output con la extensión .hex, además de los subprogramas con la extensión .r51

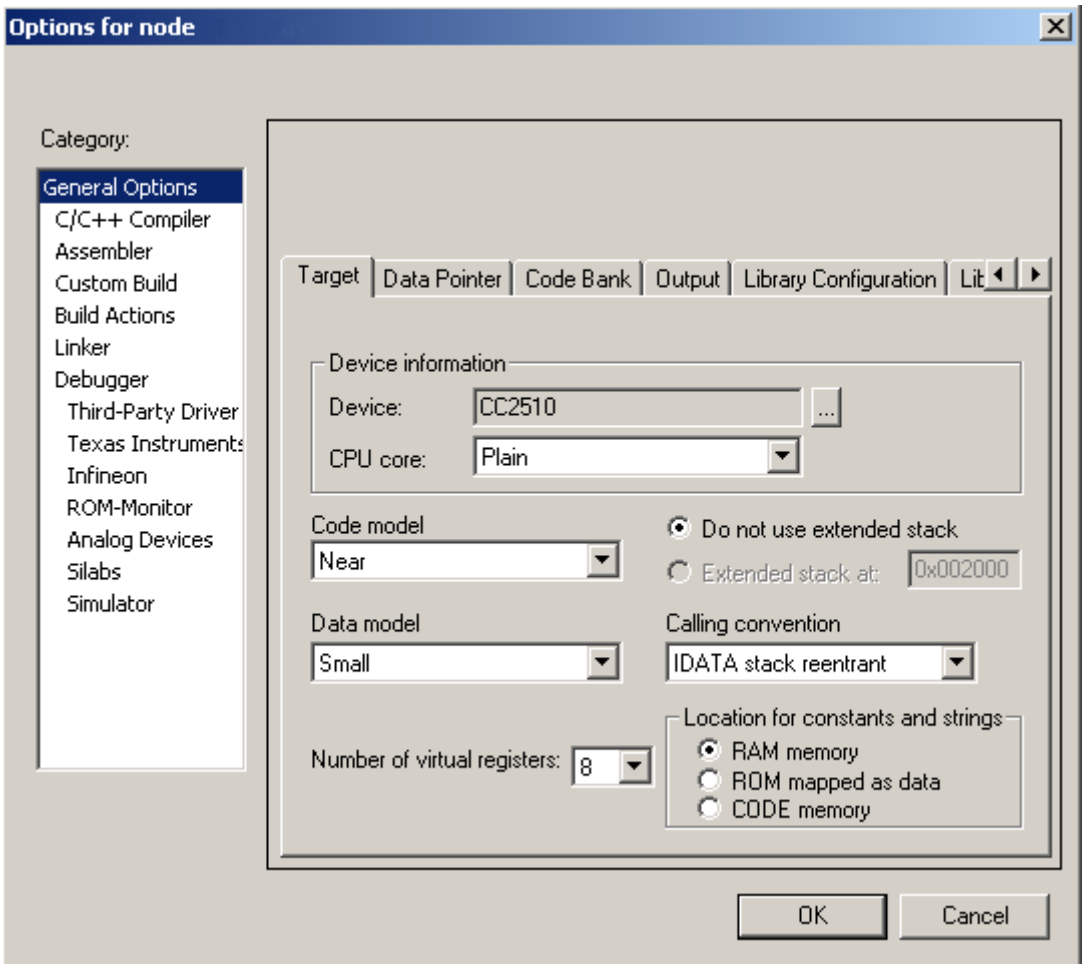

Figura 5.18: Opciones de depuración

Las opciones de depuración del Workspace son clave a la hora de cargar el software en el SoC y comprobar su correcto funcionamiento. En la Figura 5.8 se muestra la ventana de configuración de las opciones generales. Es importante que los campos Code model, Data Model y Calling convention tomen los parámetros adecuados, ya que en caso de no ser así se producen desbordamientos indeseados.

Debido a la importancia de cada parámetro de la configuración que debe tomar el Workspace y a que los parámetros por defecto provocan multitud de errores a la hora de compilar. A continuación se indica el valor que deben tomar la variables para que se pueda cargar el código, ya que en caso de obtener algún error de compilación no se logra iniciar el programa.

Se muestra el procedimiento para el CC2510, pero para el CC1110 es análogo.

1. General Options

Target

Device: CC2510, Chipcon 2510.j51

CPU: Plain

Code Model: Near, Small, 8, do not use extended, IDATA stack, RAM Memory

Data Pointer

1, Size 16

Output

Executable

Library Cont

CLIB

Lib Options

Small

Medium

Stack/Meap

0x80 0xFF

0x80 0xFFF

0x1FF 0xFFF

Misra C

1, 5-9

2. C/C++ Compiler

Languaje

C

Allow IAR ext, Unsigned

Optim

Low

Output

Gen debug

List

Todas desactivadas

Preprocesor

Def symbol CC2510

Chip=2510

3. Assembler

Languaje

User symbols

Output

Generate Debug info

List

Todas desactivadas

Preprocess

$TOOLKITS_DIR$\INC

Diagnostics

Enable

All warnings

Extra Options

Todas desactivadas

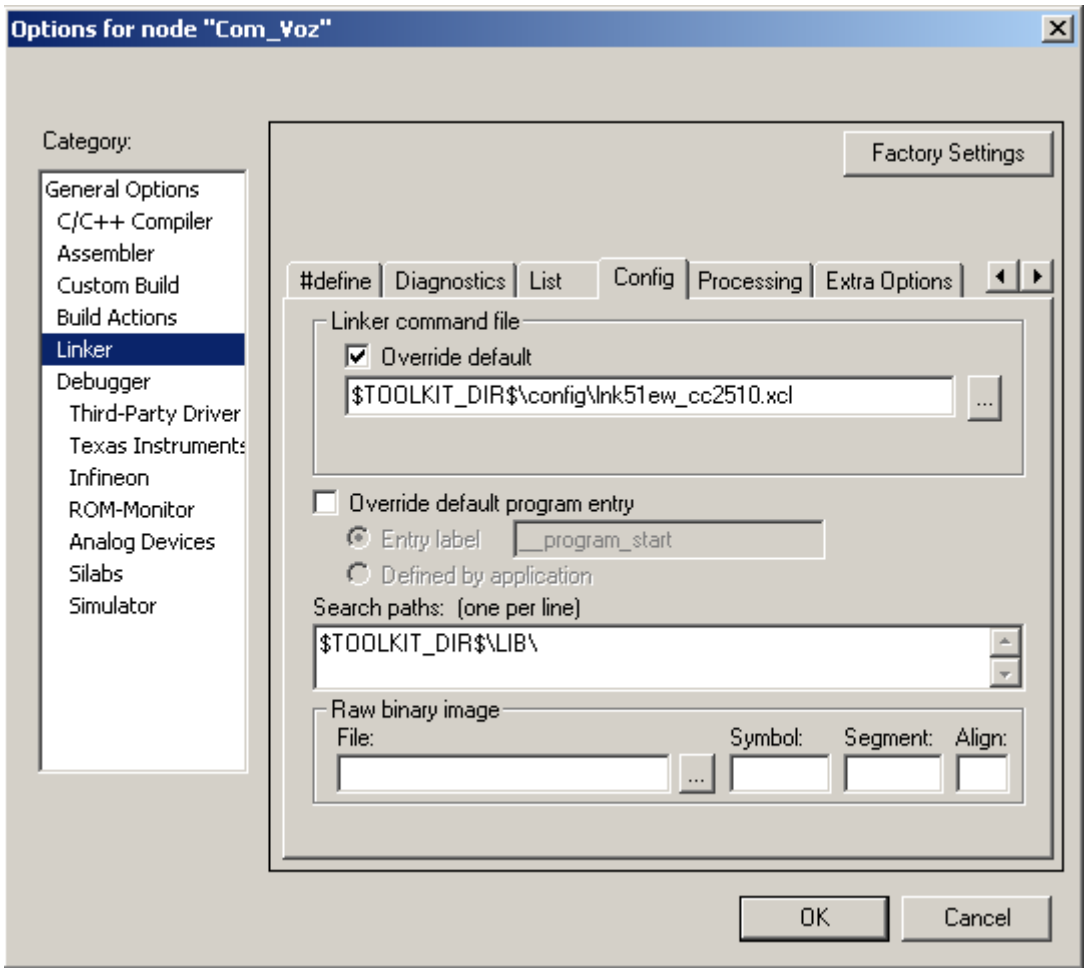

Figura 5.19: Opciones de configuración Linker-Config

4. Linker

Output

Override default, c:\temp\MeasFreqcc2510.hex

Diagnostics

Todas desactivadas salvo Generate errors

Config

√ $TOOLKITS_DIR$\config\lnk51ew_cc2510.xcl

Search path: $TOOLKITS_DIR$\LIB

5. Debugger

Setup

Driver: Texas Instruments Run to: Main

Plugins

√Code

□Orti

√Protilings

√Stacks

Override defaults: $TOOLKITS_DIR$\config\devices\TI\cc2510.ddf

Texas Instruments download

√ Retain unchanged memory

□ Erase flash

Texas Instruments target

√ Enable stack overflow warning

Number of banks: 4

Además de esto se recomienda revisar las opciones de las herramientas, sobre todo las relacionadas con el editor. En la barra de opciones principal en Tools / Options. Debe quedar configurado como se muestra en la Figura 5.7

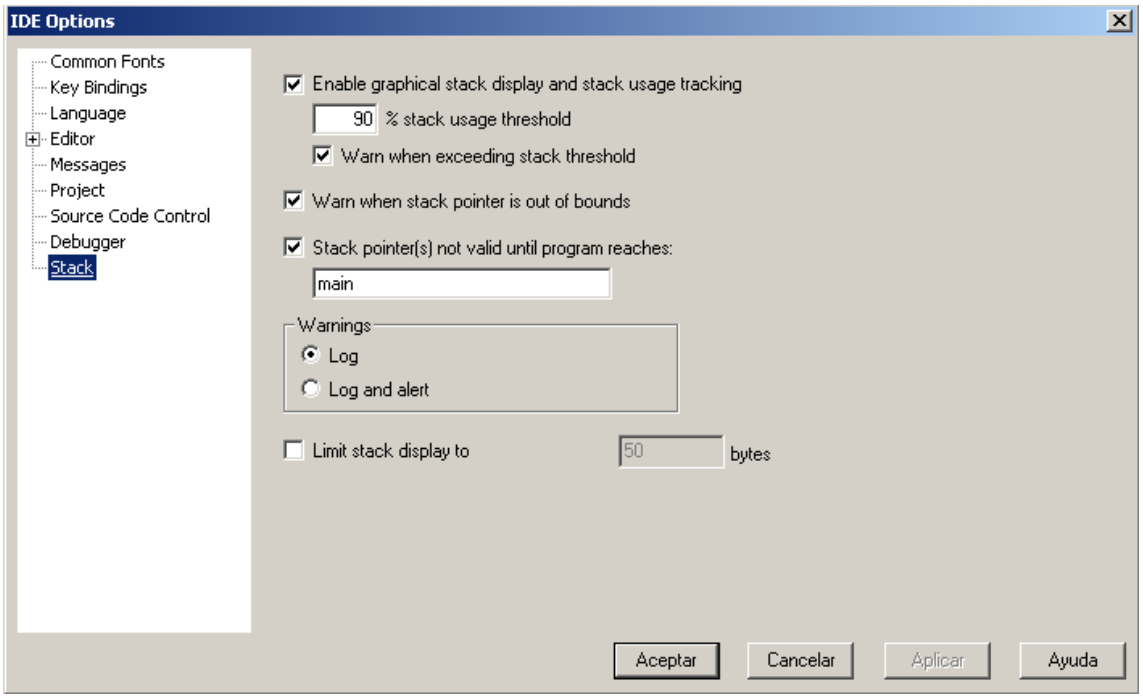

Figura 5.20: Opciones de Herramientas

A continuación, tras conectar una o ambas placas al PC mediante la interfaz USB, se procederá a cargar el código en la memoria de la placa seleccionada. En caso de tener conectadas dos simultáneamente, el entorno de desarrollo pedirá mediante una ventana de diálogo seleccionar con cual se va a trabajar. Por este motivo es muy útil mientras se está en la etapa de depuración, tener visible la dirección física del dispositivo en el LCD de la SmartRF04EB.

Si el código está correctamente compilado, el entorno software lo traduce a lenguaje ensamblador y lo carga automáticamente en el SoC

Una vez cargado el código, utilizando el modo Debug del menú desplegable Project del área de configuración, el entorno de programación de IAR inserta un Breakpoint al comienzo del main en el programa principal. Esto es muy práctico en la etapa de desarrollo y permite ejecutar paso a paso todo el código. El número de Breakpoints que se pueden introducir varía según la versión utilizada, durante el desarrollo del proyecto se han empleado la versión 7.50 y la 7.51 versión de evaluación cuyo número está limitado a tres.

Para comprobar el funcionamiento se ha utilizado un micrófono y unos auriculares estándar, con conectores jack estéreo de 3,5mm de diámetro. Se han elegido estos porque permiten una conexión directa con la placa de desarrollo SmartRF04EB configurando adecuadamente el puerto P0_0 como entrada y el P1_1 como salida de audio.

La interfaz que presenta la SmartRF04EB con el usuario es sencilla e intuitiva, se puede aumentar la información por pantalla, e incluso programar un menú posibles opciones de uso, pero se deja para posibles ampliaciones o diseño de un prototipo comercial. En la pantalla LCD se muestra si el SoC está configurado como Maestro o como Esclavo, así como la dirección física que permite identificarlo y distinguir ambos dispositivos. Se han utilizado los dos LEDs que pueden configurarse utilizando los SoC CC1110 y CC2510, el verde P1_0 y el naranja P1_3. Si todo es correcto el LED verde debe permanecer encendido tanto en el Maestro como en el Esclavo y el LED naranja sólo en el Maestro. En caso de que no estén emparejados en ambos dispositivos sólo permanecerá encendido el LED naranja.