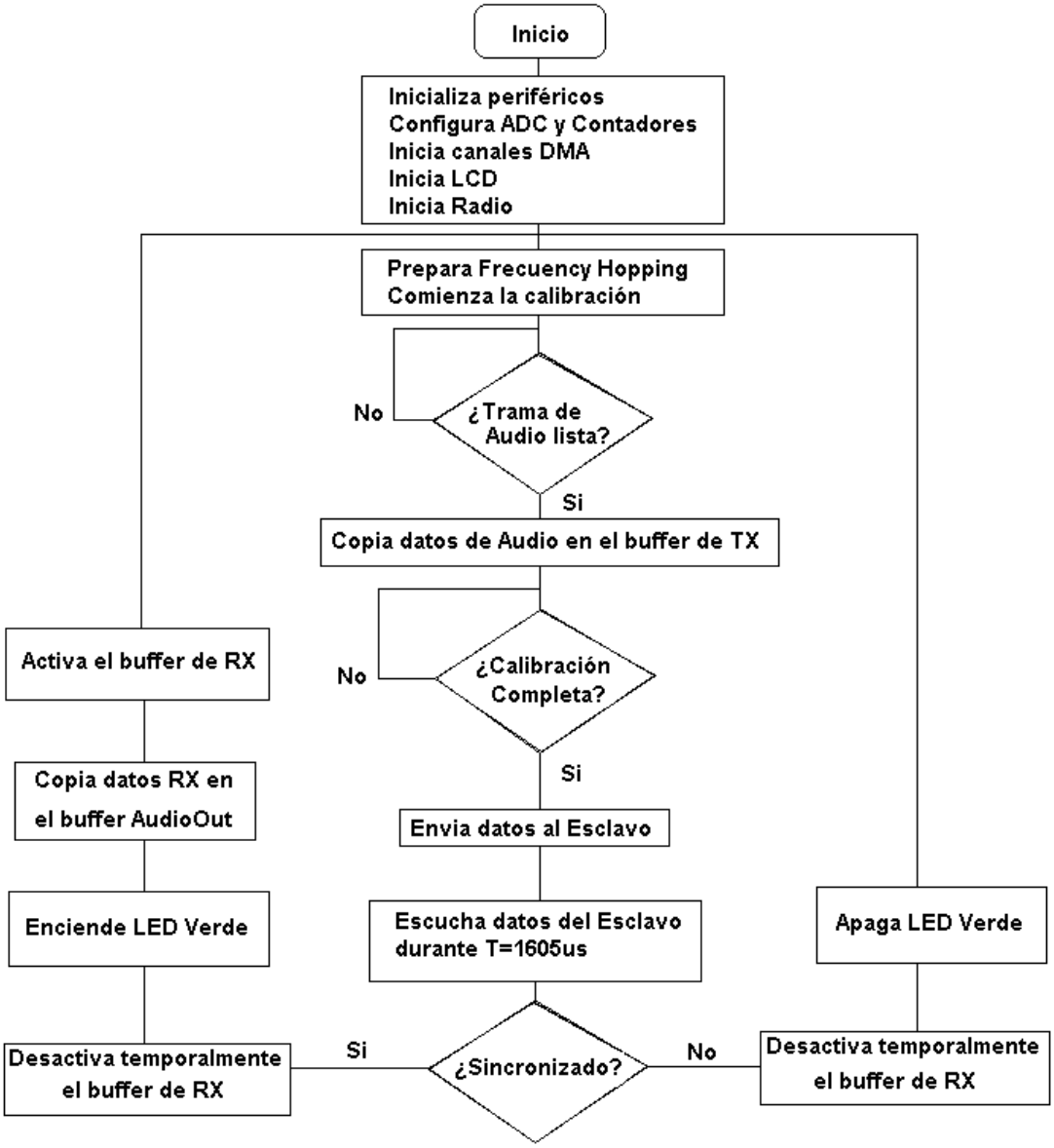
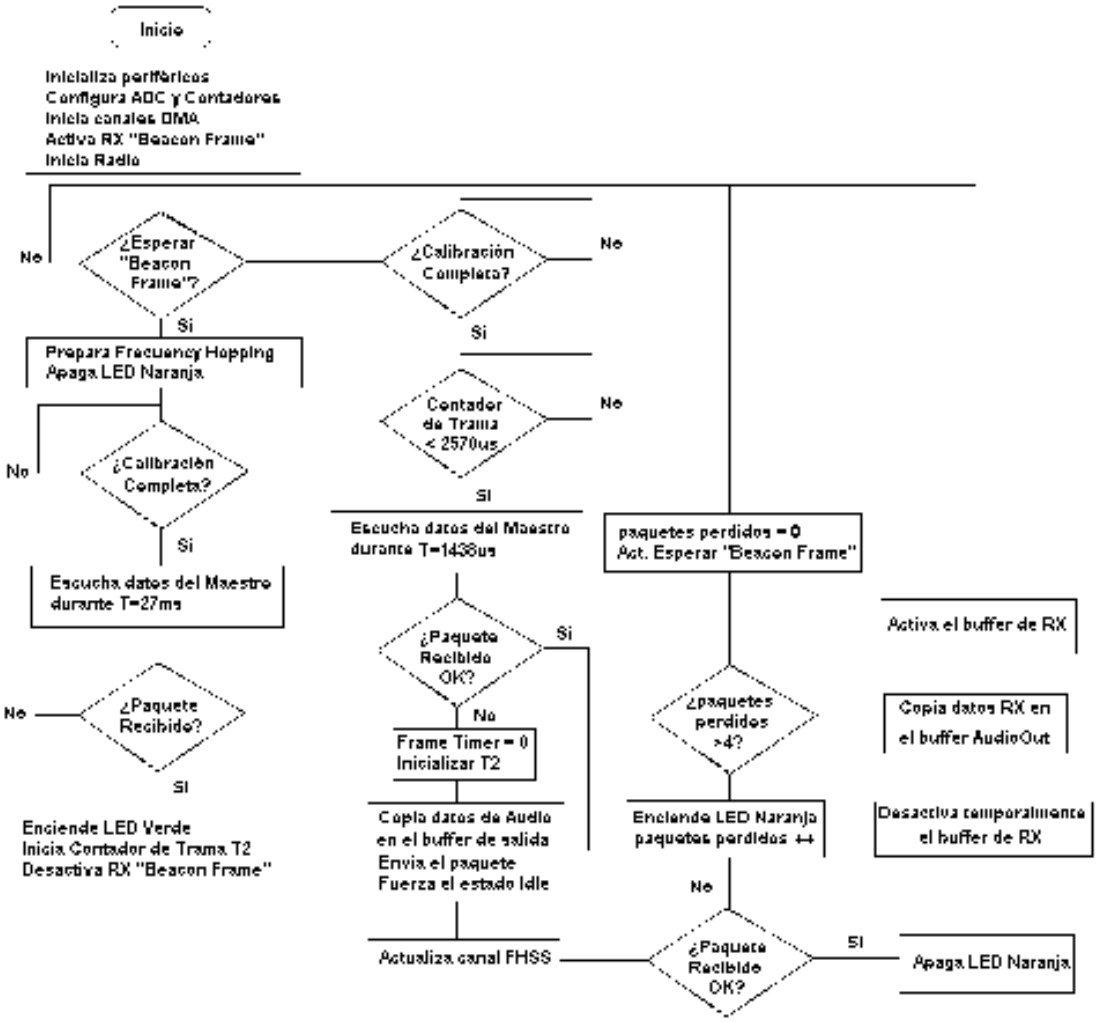
Anexo I: Pseudocódigo

Maestro



Esclavo


Inicio
Inicializa periféricos
Configura ADC y Contadores
Inicia canales DMA
Activa RX "Beacon Frame"
Inicia Radio
¿Esperar "Beacon Frame"?
No
Si
Prepara Frecuency Hopping
Apaga LED Naranja
¿Calibración Completa?
No
Si
Escucha datos del Maestro durante T=27ms
¿Paquete Recibido?
No
Si
Enciende LED Verde
Inicia Contador de Trama T2
Desactiva RX "Beacon Frame"
¿Calibración Completa?
No
Si
Contador de Trama < 2570us
No
Si
Escucha datos del Maestro durante T=1438us
¿Paquete Recibido OK?
Si
No
Frame Timer = 0
Inicializar T2
Copia datos de Audio en el buffer de salida
Envia el paquete
Fuerza el estado Idle
Actualiza canal FHSS
paquetes perdidos = 0
Act. Esperar "Beacon Frame"
¿paquetes perdidos >4?
Enciende LED Naranja
paquetes perdidos ++
No
¿Paquete Recibido OK?
Si
Apaga LED Naranja
Activa el buffer de RX
Copia datos RX en el buffer AudioOut
Desactiva temporalmente el buffer de RX

Anexo II: Código C

com_main.c

```
/*
*******************************************************************************
* com_main.c                                                                  *
* Compilador:           IAR C Compiler for 8051                               *
* SoC:                  Chipcon CC2510                                        *
* Placa de Desarrollo:  SmartRF04EB                                           *
* Autor:                MFP                                                   *
* Ultima actualizacion: 12-4-2010                                             *
*******************************************************************************/

/*******************************************************************************
*                                    INCLUDES
*******************************************************************************/

#include "voz.h"
#include "lcd.h"
#include "RF04EB.h"


/*******************************************************************************
*                             VARIABLES GLOBALES                              *
*******************************************************************************/

UINT8 __xdata activeIn = 0;                 // Activar entrada Audio (0|1)
BYTE  __xdata audioIn[2][AF_BUF_SIZE];      // Buffer de Entrada Audio
UINT8 __xdata activeOut = 0;                // Activar salida Audio (0|1)
BYTE  __xdata audioOut[2][AF_BUF_SIZE];     // Buffer de Salida Audio
UINT8 __xdata rxBufferAvailable = 0;        // El bit se activa cuando el
// buffer esta disponible

UINT8  masterstatus = 0;
UINT8  slavestatus = 0;
BOOL   waitingforbeacon = TRUE;
UINT8  lostpkts = 0;
UINT16 packets = 0;

UINT8 __xdata  activeTable[FH_ACTIVE_TABLE_LEN]; // Secuencia para el Salto de
// Frecuencia (FHSS)
UINT8 __xdata  ActiveChIdx;


/*******************************************************************************
*                             FUNCIONES LOCALES                               *
*******************************************************************************/
```

```
void initIO(void) {                         // Inicializa los puertos E/S


// Puerto 0
// P0_0  Microfono Entrada Analógica al ADC
// P0_1  Boton S1 Entrada Digital (Actualmente no se usa)
// P0_5  Joystick Entrada Digital (Actualmente no se usa)

P0SEL = 0x00;
P0DIR |= 0xEE;
P0 = 0x00;

// Puerto 1
// P1_0  E/S   LED VERDE
// P1_1  Auriculares Salida de Audio
// P1_3  E/S   LED NARANJA

P1SEL = 0xA2;
P1DIR |= 0xFF;
P1 = 0x00;
PERCFG = 0x40;

// Puerto 2

P2SEL = 0x00;
P2DIR = 0x00;
P2 = 0x00;

}

/*****************************************************************************
* @fn          main
*
* @brief       Programa Principal
*
* @param       ninguno
*
* @return      0
*****************************************************************************/

void main(void) {

UINT8 i;                                    // Variable auxiliar

// Inicializaciones
CLKCON = 0x80;                              // Selecciona Xosc como fuente
// de reloj a max velocidad
while( CLKCON != 0x80 );                    // Espera que Xosc esté estable
SLEEP |= 0x04;                              // Apaga el oscilador HSRC

initIO();                                   // Inicia Puertos E/S
initDsm();                                  // Configura el Timer 1 para DSM
I2SCFG0 = 0x00;                             // Resetea ENAB
initAdc();                                  // Configura el ADC
initDma();                                  // Inicia el admin DMA (resetea
                                            // los canales 1-4 del DMA)
initLcd();                                  // Inicia las funciones del LCD

for (i = 0; i < AF_BUF_SIZE; i++)
{
```

```
audioIn[0][i] = audioIn[1][i] = 0;
audioOut[0][i] = audioOut[1][i] = 0;
}

// Configura los canales radio y reserva canales DMA para los buffer RX/TX
initRf(RADIO_BASE_FREQ);

macTimer3Init();

activeTable[0] = CHANNEL_1;
activeTable[1] = CHANNEL_2;
activeTable[2] = CHANNEL_3;
activeTable[3] = CHANNEL_4;

INT_GLOBAL_ENABLE(INT_ON);

/*****************************************************************************/
/*                          MODO MAESTRO                                     */
/*****************************************************************************/

#ifdef MASTER


MAStxData.macPayloadLen = TX_PAYLOAD_LEN;
MAStxData.macField = MAC_ADDR;
lcdUpdate("MAESTRO", "0X00B8");

while (1)  {                            // bulce programa principal Maestro

setChannel(activeTable[ActiveChIdx]);  // Prepara FHSS
ActiveChIdx++;
if (ActiveChIdx == FH_ACTIVE_TABLE_LEN)
ActiveChIdx = 0;

SCAL();                                 // Comienza la calibración en
                                        // un nuevo canal

while (audioFrameReady == FALSE) {  // Espera que la trama de audio
                                        // este lista
}

audioFrameReady = FALSE;
MAStxData.statusField = masterstatus;
MAStxData.channel[0] = activeTable[0];
MAStxData.channel[1] = activeTable[1];
MAStxData.channel[2] = activeTable[2];
MAStxData.channel[3] = activeTable[3];

for (i = 0; i < AF_BUF_SIZE; i++) {
if (activeIn == 1)
MAStxData.payload[i] = audioIn[0][i];
else
MAStxData.payload[i] = audioIn[1][i];
}

while (MARCSTATE != MARCSTATE_IDLE);       // Espera a calibracion completa

rfSendPacket(MASTER_TX_TIMEOUT_WO_CALIB);

// Esperando datos del Esclavo
```

```
rfReceivePacket(&MASrxData, MASTER_RX_TIMEOUT_WO_CALIB, RX_PAYLOAD_LEN);

if (rxPacketStatus == TIMEOUT_ERROR) {      // Si no se sincroniza...

LED_GREEN_REG |= LED_GREEN;              // Apaga el Led Verde

if (activeOut == 0)
rxBufferAvailable &= ~BUFFERA;          // Buffer A NO Disponible
else
rxBufferAvailable &= ~BUFFERB;          // Buffer B NO Disponible
}
else {                                              // Si se recibe un paquete...
LED_GREEN_REG &= ~LED_GREEN;             // Enciende el Led Verde

// a continuación marca ambos buffer como no disponible

if (activeOut == 0)
rxBufferAvailable &= ~BUFFERA;          // Buffer A NO Disponible
else
rxBufferAvailable &= ~BUFFERB;          // Buffer B NO Disponible

// Introduce los datos de audio al buffer

for (i = 0; i < AF_BUF_SIZE; i++)
audioOut[activeOut][i] = MASrxData.payload[i];

// Vuelve a poner disponible el buffer de RX

if (activeOut == 0)
rxBufferAvailable |= BUFFERA;            // Buffer A Disponible
else
rxBufferAvailable |= BUFFERB;            // Buffer B Disponible
}     // fin del 'else' TIMEOUT_ERROR (Paquete Recibido)

activeOut ^= 0x01;                              // En cualquier caso conmuta a
                                                // activeOut

}   // fin del bucle Principal del Maestro
}
#endif

/*****************************************************************************/
/*                          MODO ESCLAVO                                     */
/*****************************************************************************/


#ifdef SLAVE


// Inicia Timer 2 (Contador de Trama)
// Timer 2 es un contador descendente de 8 bits con un prescaler de 18 bits
// El periodo es T = T2PR * Val(T2CTL.TIP) ciclos de reloj, donde
// Val = 64 if T2CTL bits 1:0 = 0
// Val = 128 if T2CTL bits 1:0 = 1
// Val = 256 if T2CTL bits 1:0 = 2
// Val = 1024 if T2CTL bits 1:0 = 3

lcdUpdate("ESCLAVO", "0X00BD");
T2CT = 0;                               // Detiene el contador
```

```
T2PR = 6;                               // Multiplicador
T2CTL = 0x01;                           // Configura el T=128*T2PR
                                        // ciclos (128*6/26.000MHz
                                        // = 29.538 usec)
                                        // Se desactivan las interrupciones

SLVtxData.macPayloadLen = TX_PAYLOAD_LEN;
SLVtxData.macField = MAC_ADDR;

while (1)  {                            // bucle modo Esclavo
if (waitingforbeacon) {
LED_ORANGE_REG &= ~LED_ORANGE;                  // Enciende el LED verde

setChannel(0);                                  // Escucha en el canal 0
SCAL();                                         // Comienza la calibración en
// un nuevo canal
while (MARCSTATE != MARCSTATE_IDLE);            // Espera a calibracion completa

rxPacketStatus == PKT_STATUS_UNKNOWN;

while (rxPacketStatus != PKT_OK) {              // Permanece en este estado
// hasta recibir un paquete

ListenforMaster(&SLVrxData, LISTEN_FOR_BEACON_TIMEOUT, RX_PAYLOAD_LEN);

}   // si se sincroniza sale de este estado y...


LED_ORANGE_REG |= LED_ORANGE;                   // Apaga el LED Naranja
lostpkts = 0;                                   // Resetea el nºpaq perdidos
ActiveChIdx = 1;                                // Configura el canal siguiente
// como activo
waitingforbeacon = FALSE;                       // Sale del modo sincr.
T2CT = FRAME_TIMEOUT_DEFAULT;                   // Resetea el contador de trama
}     // se sale de este modo si...

else {                          // Está sincronizado

// Escuchando al Maestro

while (MARCSTATE != MARCSTATE_IDLE) {           // Espera a completar la
                                                // calibracion. Estado=ocioso
}

while (T2CT > LISTENFORMASTER) {        // Espera a poder escuchar
}

rfReceivePacket(&SLVrxData, SLAVE_RX_TIMEOUT_WO_CALIB, RX_PAYLOAD_LEN);

if (rxPacketStatus == TIMEOUT_ERROR)
while (T2CT > 0) {                              // deja q el contador llegue a 0
}

T2CT = FRAME_TIMEOUT_DEFAULT;                   // Resetea el contador de trama

}     // fin del Estado Sincronizado.

// Enviando datos al Maestro

SLVtxData.statusField = slavestatus;
```

```
for (i = 0; i < AF_BUF_SIZE; i++) {
if (activeIn == 1) {
SLVtxData.payload[i] = audioIn[0][i];
}
else {
SLVtxData.payload[i] = audioIn[1][i];
}
}


rfSendPacket(SLAVE_TX_TIMEOUT_WO_CALIB);

SIDLE();                          // Fuerza estado ocioso en RX

// Configura el siguiente canal

setChannel(activeTable[ActiveChIdx]);       // SetChannel establece
// MARCSTATE a IDLE(ocioso)
ActiveChIdx++;
if (ActiveChIdx == FH_ACTIVE_TABLE_LEN)
ActiveChIdx = 0;
SCAL();                           // Comienza la calibración en
// un nuevo canal

if (rxPacketStatus == TIMEOUT_ERROR) {      // Error en RX

LED_GREEN_REG |= LED_GREEN;               // Apaga el LED verde

if (activeOut == 0)
rxBufferAvailable &= ~BUFFERA;          // Buffer A NO Disponible
else
rxBufferAvailable &= ~BUFFERB;          // Buffer B NO Disponible

lostpkts++;
if (lostpkts > 4) {                       // Si se pierden 4 paquetes
lostpkts = 0;
waitingforbeacon = TRUE;                  // entra en modo re-sincronismo
}
}
else {                              // Se reciben paquetes
LED_GREEN_REG &= ~LED_GREEN;              // Enciende el LED verde
lostpkts = 0;

// Marca temporalmente el Buffer de RX como NO Disponible

if (activeOut == 0)
rxBufferAvailable &= ~BUFFERA;          // Buffer A NO Disponible
else
rxBufferAvailable &= ~BUFFERB;          // Buffer B NO Disponible

// Introduce los datos en el buffer de audio

for (i = 0; i < AF_BUF_SIZE; i++)
audioOut[activeOut][i] = SLVrxData.payload[i];

// Vuelve a poner disponible el Buffer de Audio

if (activeOut == 0)
rxBufferAvailable |= BUFFERA;             // Buffer A Disponible
```

```
else
rxBufferAvailable |= BUFFERB;          // Buffer B Disponible
}     // fin del estado TIMEOUT_ERROR si se rx paquetes

activeOut ^= 0x01;                     // En cualquier caso activeOut
}
// Fin del bucle principal del Modo Esclavo
}
#endif
```

voz.h

```
/*
 ********************************************************************************************
 * voz.h                                                                                    *
 * Compilador:           IAR C Compiler for 8051                                            *
 * SoC:                  Chipcon CC1110                                                     *
 * Placa de Desarrollo:  SmartRF04EB                                                        *
 * Autor:                MFP                                                                *
 * Ultima Actualización: 12-4-2010                                                          *
 ********************************************************************************************
*/

#ifndef VOZ_H
#define VOZ_H


/*********************************************************************************************
*                                  INCLUDES                                                  *
*********************************************************************************************/

#include "hal.h"
#include "iocc1110.h"

/*********************************************************************************************
*                             CONFIGURACION DE MODOS                                         *
*********************************************************************************************/

// Elija el MODO Maestro des-comentando la primera linea, en otro caso se
// configura como Esclavo

//#define MASTER
#ifndef MASTER
#define SLAVE
#endif


// Incluir control Automatico de ganancia
#define AGC

// Descomentando la siguiente linea se mezcla el audio d salida con el rx para
// poder oirnos a nosotros mismos
#define LOCALFEEDBACK

// Uncomment the following #define to generate a missed packet ever 'howoften' packets


/*********************************************************************************************
*                               DEFINICIONES GLOBALES                                        *
*********************************************************************************************/

#define MAC_ADDR                0xC3      // Primer byte. dir MAC
//#define RADIO_BASE_FREQ           2406000 // Radiofrecuencia (kHz) Canal 0 CC2510
#define RADIO_BASE_FREQ           433000   // Radiofrecuencia (kHz) Canal 0
```

```
// Number of ADC samples per period
#define AF_LEN  54

// Audio frame and buffer size
#define AF_BUF_SIZE  AF_LEN                // Tamaño buffer de audio = numero de muestras
ADC

// Ajustes tasa de muestreo
// El contador 1 se basa en la fecuencia Xtal 26.000MHz, un ciclo son 38.46ns

#define NORMAL_SAMPLE_PERIOD    3249          // 3249 * 38.46 ns -> ~125 us (8 kHz tasa
de muestreo)
#define CLOCK_SKEW_STEP_SIZE       2          //    2 * 38.46 ns -> ~77 ns

#define SPEEDUP   AF_LEN/2 - 5
#define SLOWDOWN  AF_LEN/2 + 5

// Formato de los paquetes (bytes)
#define PHY_FIELD_LEN              1       // Longitud byte
#define MAC_FIELD_LEN              1       // Direccion MAC
#define STATUS_FIELD_LEN           1       // Bits de Estado finales
#define CHANNEL_TABLE_LEN          4       // Tabla de canales
#define DF_LEN                     AF_LEN  // Longitud del campo de datos = Longitud trama de
audio
#define MAC_APPENDED_STATUS_LEN    2       // Bytes de estado añadidos por PHY
(CC2510)

#ifdef MASTER
#define TX_PAYLOAD_LEN  (MAC_FIELD_LEN + STATUS_FIELD_LEN +
CHANNEL_TABLE_LEN + DF_LEN)
#define RX_PAYLOAD_LEN  (MAC_FIELD_LEN + STATUS_FIELD_LEN + DF_LEN)
#else
#define RX_PAYLOAD_LEN  (MAC_FIELD_LEN + STATUS_FIELD_LEN +
CHANNEL_TABLE_LEN + DF_LEN)
#define TX_PAYLOAD_LEN  (MAC_FIELD_LEN + STATUS_FIELD_LEN + DF_LEN)
#endif

// Asignacion de canales DMA
#define DMA_RX  1
#define DMA_TX  2

// 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

// Banderas Contador
#define T3OVFIF                    0x01

// Modos RX Off
#define RXOFFMODE_IDLE             0x00
#define RXOFFMODE_FSTXON           0x04
#define RXOFFMODE_TX               0x08
#define RXOFFMODE_RX               0x0C

// Estados
#define MARCSTATE_TX               0x13
#define MARCSTATE_RX               0x0D
```

```
#define MARCSTATE_IDLE              0x01

// Definiciones bits de regitros PKTSTATUS
#define SFD                         0x08

// Definiciones T2 (Contador de Trama)
// El periodo de T2 es 128*6/26.000 MHz = 29.538 usec
#define FRAME_TIMEOUT_DEFAULT       229     // (T2CT) Tiempo de espera = 229 *
29.538 us = 6.764 ms
#define LISTENFORMASTER             87      // Empieza a escuchar al Maestro 2570 usec
antes del
                                    // final de la trama o 300 usec antes del paquete esperado

// T3 Definiciones de los Tiempos de Espera (Tiempo de espera 4.923076923 us,
// Multiplicador porque es un contador de solo 8 bits)
// El periodo de T3 es 128/26 = 4.923076923 us
#define MASTER_TX_TIMEOUT_WO_CALIB    234, 2    // Tiempo de espera = 234 *
4.923076923 us * 2 = 2304.0 us
#define MASTER_RX_TIMEOUT_WO_CALIB    163, 2    // Tiempo de espera = 163 *
4.923076923 us * 2 = 1604.9 us
#define SLAVE_TX_TIMEOUT_WO_CALIB     220, 2    // Tiempo de espera = 220 *
4.923076923 us * 2 = 2166.4 us
#define SLAVE_RX_TIMEOUT_WO_CALIB     146, 2    // Tiempo de espera = 146 *
4.923076923 us * 2 = 1437.5 us
#define LISTEN_FOR_BEACON_TIMEOUT     229, 24   // Tiempo de espera = 229 *
4.923076923 us * 24 = 27.057 msec

// Estado del Paquete en RX
#define PKT_OK                      0x01
#define CRC_ERROR                   0x02
#define TIMEOUT_ERROR               0x04
#define PKT_ERROR                   0x08
#define DMA_ERROR                   0x10
#define PKT_STATUS_UNKNOWN          0xFF

// Valores de Bits del Estado de los Paquetes
#define CRC_OK_MASK     0x80
#define RSSI_VAL_IDX    0
#define CRC_LQI_VAL_IDX 1

// LEDs


#define LED_ORANGE_REG P1
#define LED_ORANGE 0x08   // P1_3
#define LED_GREEN_REG P1
#define LED_GREEN  0x01   // P1_0


/*****************************************************************************
*                     DECLARACION DE ESTRUCTURAS                             *
*****************************************************************************/

#ifdef MASTER

// TX struct
typedef struct{
  BYTE macPayloadLen;           // Tamaño bytes
  BYTE macField;                // Direccion MAC
  BYTE statusField;             // Byte de Estado
```

```
  BYTE channel[CHANNEL_TABLE_LEN];      // Tabla de canales
  BYTE payload[DF_LEN];                 // Muestras de Audio
} __xdata TX_MASTER_STRUCT;

// RX struct
typedef struct{
  BYTE macPayloadLen;                   // Tamaño bytes
  BYTE macField;                    // Direccion MAC
  BYTE statusField;                 // Byte de Estado
  BYTE payload[DF_LEN];                 // Muestras de Audio
  BYTE append[MAC_APPENDED_STATUS_LEN]; // Byte de estado (RSSI, LQI)
} __xdata RX_MASTER_STRUCT;

#endif

#ifdef SLAVE

// RX struct
typedef struct{
  BYTE macPayloadLen;                   // Tamaño bytes
  BYTE macField;                    // Direccion MAC
  BYTE statusField;                 // Byte de Estado
  BYTE channel[CHANNEL_TABLE_LEN];      // Channel table
  BYTE payload[DF_LEN];                 // Muestras de Audio
  BYTE append[MAC_APPENDED_STATUS_LEN]; // Bytes de estado (RSSI, LQI)
} __xdata RX_SLAVE_STRUCT;

// TX struct
typedef struct{
  BYTE macPayloadLen;                   // Tamaño bytes
  BYTE macField;                    // Direccion MAC
  BYTE statusField;                 // Byte de Estado
  BYTE payload[DF_LEN];                 // Muestras de Audio
} __xdata TX_SLAVE_STRUCT;

#endif

/*****************************************************************************
*    DECLARACION DE FUNCIONES PROTOTIPO DEFINIDAS EN HAL    *
*****************************************************************************/

// tw_dma.c
void initDma(void);
void dmaToRadio(WORD length, WORD srcAddr);
void dmaFromRadio(WORD length, WORD dstAddr);

// tw_adc.c
void initAdc(void);

// tw_dsm.c
void initDsm(void);

// tw_rf.c
BOOL initRf(UINT32 frequency);
BOOL rfSendPacket(UINT8 timeout, UINT8 multiplier);
#ifdef MASTER
void rfReceivePacket(RX_MASTER_STRUCT __xdata * rxData, UINT8 timeout, UINT8
t3_multiplier, UINT8 packetlen);
#else
```

```
void rfReceivePacket(RX_SLAVE_STRUCT __xdata * rxData, UINT8 timeout, UINT8
t3_multiplier, UINT8 packetlen);
void ListenforMaster(RX_SLAVE_STRUCT __xdata * rxData, UINT8 timeout, UINT8
t3_multiplier, UINT8 packetlen);
#endif

// tw_mac.c
void initFrameTimer(void);
void startFrameTimer(UINT8 cnt);
void macTimer3Init(void);
void macsetT3TimeoutAndWait(UINT8 timeout, UINT8 multiplier);
void setChannel (UINT8 ch);

/*********************************************************************************************
*                    VARIABLES GLOBALES EXTERNAS                                             *
*********************************************************************************************/

#ifdef MASTER
extern TX_MASTER_STRUCT __xdata MAStxData;      // Empaquetador de datos
extern RX_MASTER_STRUCT __xdata MASrxData;
#endif

#ifdef SLAVE
extern RX_SLAVE_STRUCT __xdata SLVrxData;       // Empaquetador de datos
extern TX_SLAVE_STRUCT __xdata SLVtxData;
#endif

extern DMA_DESC __xdata *dmaDescTx;             // Puntero al descriptor DMA de TX
extern DMA_DESC __xdata *dmaDescRx;             // Puntero al descriptor DMA de RX

extern volatile UINT8 __xdata rxPacketStatus;

extern UINT8 __xdata activeTable[FH_ACTIVE_TABLE_LEN];

extern BYTE __xdata audioIn[2][AF_BUF_SIZE];      // Buffer de Entrada Audio
extern UINT8 __xdata activeIn;                    // Activar entrada Audio (0|1)
extern BYTE __xdata audioOut[2][AF_BUF_SIZE];     // Buffer de Salida Audio
extern UINT8 __xdata activeOut;                   // Activar salida Audio (0|1)
extern UINT8 __xdata rxBufferAvailable;           // El bit se activa cuando el buffer esta
disponible

#define BUFFERA 0x40          // Bit 6 se activa siempre que el Buffer RX A está disponible
#define BUFFERB 0x80          // Bit 7 se activa siempre que el Buffer RX B está disponible
extern UINT8 audioInindx;
extern UINT8 audioOutindx;
extern UINT16 timerSampleRate;
extern BOOL __xdata audioFrameReady;
extern UINT8 sampleCount;
extern UINT8 __xdata ActiveChIdx;
extern BOOL sfdDetected;
extern BOOL dmaDone;
#endif
```