7 Anexo I: Código fuente

```
// Fichero de cabecera de NYDIANET
// Alpha 2.0 - 31 Julio 2006

/*********************************************************************
SALIDAS:  Relacion entre salidas y filas de la tabla "operacion"

   B1    B2    B3    B4    B5    B6    B7    A4
   0     1     2     3     4     5     6     7

ENTRADAS:  Relacion entre entradas y bits del byte "entrada actual"

   A0    A1    A2    A3    A5    C0
   0     1     2     3     4     5

*********************************************************************/

/*****************************************************
   Includes
*****************************************************/
// Libreria basica del PIC18F2620
#include <18F2620.h>

// Uso de ICD para programacion
#device ICD=TRUE

// Canales A/D
#device adc=8

// Configuracion de la velocidad de reloj (Hz)
#use delay(clock=20000000)

/*****************************************************
   Configuracion Micro
*****************************************************/
#FUSES NOWDT       // Sin Watch Dog Timer
#FUSES HS          // Oscilador de mas de 4 MHz
#FUSES NOPROTECT   // Codigo no protegido de lectura
#FUSES IESO        // Modo "Internal External Switch Over" activado
#FUSES BROWNOUT    // Reset tras deteccion de fallo de alimentacion
#FUSES BORV21      // Fallo de alimentacion establecido bajo 2.1V
#FUSES NOPUT       // Sin Power Up Timer
#FUSES NOCPD       // Sin proteccion EE
#FUSES STVREN      // Desbordamiento y vaciado de pila causa reset
#FUSES NODEBUG     // No se usa modo de depuracion con ICD
#FUSES NOLVP       // No se usa programacion de voltaje bajo
#FUSES NOWRT       // Memoria de programa no protegida contra escritura
#FUSES NOWRTD      // EEPROM de datos no protegida
#FUSES NOEBTR      // Memoria no protegida de lecturas de tabla
#FUSES NOCPB       // Sector de arranque no protegido
#FUSES NOEBTRB     // Sector de arranque no protegido contra escritura
#FUSES NOWRTC      // Registros de configuracion no protegidos contra escritura
#FUSES NOWRTB      // Sector de arranque no protegido contra escritura
#FUSES FCMEN       // Monitorizacion de reloj activa
#FUSES NOXINST     // Direccionamiento extendido e indexado desactivados
#FUSES NOPBADEN    // Puerto B configurado como E/S digital tras un reset
#FUSES LPT1OSC     // Timer 1 configurado para operacion de bajo consumo
#FUSES MCLR        // Pin de reseteo activado

/*****************************************************
   Configuracion Bus
*****************************************************/
// 19200 Baudios, sin paridad, pin de txon: C6, pin de rxon: C7,
```

```
//   8 bits, pin enable: C5, nombre: RS485_BUS
#use rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7,
           bits=8,stream=RS485_BUS,enable=PIN_C5)

// Número máximo de mensajes en los buffer
#define MAXMENSAJES 5

// Longitud máxima de las tramas
#define MAXLONGITUD 64

// Tamaño de la cabecera
#define HEADER_SIZE 7

// Máximo numero de retransmisiones
#define MAX_RETX 2

/*******************************************************
   Configuracion E/S
*******************************************************/
// Modo rapido para puertos A y B
#use fast_io(A)
#use fast_io(B)

// Mapeo de puertos
#byte PORTA = 0xF80
#byte PORTB = 0xF81
#byte PORTC = 0xF82

/*******************************************************
   Registros
*******************************************************/
// Mapeo de registro de periodo de Timer 2
#byte PR2 = 0xFCB

/*******************************************************
   Codificacion de las entradas en la configuracion
*******************************************************/
#define C_IN_AN0 0
#define C_IN_AN1 1
#define C_IN_AN2 2
#define C_IN_AN3 3
#define C_IN_AN4 4
#define C_IN_RC0 5

/*******************************************************
   Codificacion de las salidas en la configuracion
*******************************************************/
#define C_OUT_B1 0
#define C_OUT_B2 1
#define C_OUT_B3 2
#define C_OUT_B4 3
#define C_OUT_B5 4
#define C_OUT_B6 5
#define C_OUT_B7 6
#define C_OUT_A4 7

/*******************************************************
   Codificacion de los modos en la configuracion
*******************************************************/
// Modo local digital normal conmutacion
#define M_LOC_DIG_NOR_TOG 0b00000001

// Modo local digital normal pulsador
#define M_LOC_DIG_NOR_PUL 0b00000010

// Modo local digital normal regulacion
```

```
#define M_LOC_DIG_NOR_DIM 0b00000011

// Modo remoto digital normal conmutacion
#define M_REM_DIG_NOR_TOG 0b10000001

// Modo remoto digital normal pulsador
#define M_REM_DIG_NOR_PUL 0b10000010

// Modo remoto digital normal regulacion
#define M_REM_DIG_NOR_DIM 0b10000011

/*****************************************************
   Codificacion de las operaciones en la configuracion
*****************************************************/
// Operacion local de activacion de salida
#define O_LOC_ASAL 0b00000010

// Operacion local de desactivacion de salida
#define O_LOC_DSAL 0b00000011

// Operacion remota de activacion de salida
#define O_REM_ASAL 0b10000010

// Operacion remota de desactivacion de salida
#define O_REM_DSAL 0b10000011

/*****************************************************
   Campos de las tramas
*****************************************************/
// Flag de inicio de trama
#define T_FLAG       0b01111110

// Direccion propia (solo para configuracion por defecto)
#define T_DIR_PROP   0b00000010

//   Direccion de Difusión
#define T_DIR_DIF    0b00000000

//   Indicador de trama de datos
#define T_TDAT       0b00000000

//   Indicador de trama de asentimiento
#define T_TACK       0b10000000

//   Indicador de trama de control
#define T_TCTL       0b01000000

//   Tipo de trama de datos: Datos de Configuración
#define T_TDAT_DCFG  0b00000000

//   Tipo de trama de control: Comprobar Dirección
#define T_TCTL_CDIR  0b00000000

//   Tipo de trama de control: Indicación de Dirección Repetida
#define T_TCTL_IDRE  0b00000001

//   Tipo de trama de control: Activar/Encender Salida
#define T_TCTL_ASAL  0b00000010

//   Tipo de trama de control: Desactivar/Apagar Salida
#define T_TCTL_DSAL  0b00000011

//   Tipo de trama de control: Conmutar Salida
#define T_TCTL_CSAL  0b00000100

//   Tipo de trama de control: Fijar Potencia Dimmer
```

```
#define T_TCTL_FPOT  0b00000101

//    Tipo de trama de control: Cambiar Potencia Dimmer
#define T_TCTL_CPOT  0b00000110

/*****************************************************
   Definicion de constantes
*****************************************************/
// Valor de desbordamiento de lso timers de 16 bits
#define OVERFLOW 0xFFFF

// Factor de conversion de valor de potencia a valor del timer
#define DIMMER_SCALE 0xC4

// Valor maximo de la potencia de dimmer (se corresponde con minima intensidad)
#define DIMMER_POT_VMAX 210

// Valor minimo de la potencia de dimmer (se corresponde con maxima intensidad)
#define DIMMER_POT_VMIN 0

// Umbral de potencia minima y maxima
#define DIMMER_POT_GAP 10

// Byte todo a 0
#define NULL_BYTE 0x00

// Byte todo a 1
#define FULL_BYTE 0xFF

// Mascara para el nibble menos significativo
#define LS_4BITS 0x0F

// Mascara para el nibble mas significativo
#define MS_4BITS 0xF0

// Duracion de la pulsacion larga
#define PULS_EXTRA_LARGA 40

// Duracion de la pulsacion corta
#define PULS_CORTA 10

// Número máximo de nodos de la red
#define MAX_NODOS 32

// Número máximo de filas en configuración
#define MAX_ENTRADAS 32

// Numero de salidas
#define NUM_SALIDAS 8

// Numero de entradas
#define NUM_ENTRADAS 6

// Número de ciclos del timeout de asentimiento
#define ACKTOUT_CICLOS 10

// Número de ciclos del timeout de recepción
#define RXTOUT_CICLOS 10

// Direcciones en EEPROM donde se almacenan los estados de las salidas
#define S1_DIR 0x03F8
#define S2_DIR 0x03F9
#define S3_DIR 0x03FA
#define S4_DIR 0x03FB
#define S5_DIR 0x03FC
#define S6_DIR 0x03FD
```

```
#define S7_DIR 0x03FE
#define S8_DIR 0x03FF

/*******************************************************
   Sistema Operativo en Tiempo Real (RTOS)
*******************************************************/
// Declaracion de timer a usar
#use RTOS(timer=0)

///  Definición de tareas

// Tarea de captura de entradas
#task (rate = 25ms)
void CapturaEntradas();

// Tarea de activacion de salidas
#task (rate = 25ms)
void ActivaSalidas();

// Tarea de interpretacion de tramas recibidas
#task (rate = 100ms)
void InterpretaTramas();

// Tarea de envio de tramas del buffer de salida
#task (rate = 100ms)
void EnviaTramas();

// Tarea de extraccion de tramas de prebuffer de entrada
#task (rate = 25ms)
void RecogeTrama();

// Tarea de gestión de contadores
#task (rate = 50ms)
void Contadores();

/*******************************************************
   Prototipos de funciones
*******************************************************/
// Inicialización de variables globales de longitud variable
// Parametros: Ninguno
// Entrada: Variables globales y constantes definidas en cabecera
// Salida: Variables globales inicializadas
void Inicializacion();

// Conmprobación de la existencia de configuración en EEPROM
// Parametros: Ninguno
// Entrada: Dirección 0 de la EEPROM
// Salida: 1 si existe, 0 si no existe
int1 ExisteConfig();

// Lectura de configuración de EEPROM
// Parametros: Ninguno
// Entrada: Contenido de la EEPROM
// Salida: A través de variables globales
void LeeConfig();

// Creación de configuración por defecto
// Parametros: Ninguno
// Entrada: Configuración por defecto definida como constante
// Salida: Escritura en EEPROM de la configuracion
void CreaConfigDefecto();

// Cambio cíclico de la potencia del dimmer
// Parámetros: Dimmer a cambiar (0,1) y cantidad a cambiar (1 o 5)
// Entrada: Parámetro
// Salida: Escritura en EEPROM de la configuracion
```

```
void CambiarPotDimmer(int dimmer, int cantidad);

// Creación de trama en buffer de salida
// Parametros: Dirección destino, puntero a campo de
//    control, Longitud y puntero a datos
// Entrada: Parámetros
// Salida: 1 si es posible, 0 si no. Escritura en buffer de salida de la trama
int1 CreaTrama(int8 dirdest, int8 *control, int8 longitud, int8 *datos);

// Inserción de dirección y resto de campos en la
//    tabla de direcciones si esta no existe
// Parametros: Dirección a insertar
// Entrada: Parámetros
// Salida: A través de variables globales
void InsertaDir(int8 Dir);

// Busqueda de dirección en la tabla de direcciones
// Parametros: Dirección a buscar
// Entrada: Parámetros
// Salida: Índice de la tabla donde se encontró la dirección.
int BuscaDir(int8 Dir);

/*******************************************************
   Interrupciones
*******************************************************/
// Interrupción externa
// Ocurrencia: Paso por cero de la señal de alimentación de la red eléctrica
#int_EXT
void PasoPorCero();

// Desbordamiento del timer 1
// Ocurrencia: Instante en que debe conmutarse
//    el triac para el control de potencia
#int_TIMER1
void Dimmer1();

// Desbordamiento del timer 3
// Ocurrencia: Instante en que debe conmutarse
//    el triac para el control de potencia
#int_TIMER3
void Dimmer2();

// Byte listo en el buffer (hardware) de entrada
// Ocurrencia: Instante en que se recibe un byte por el bus de datos
#int_RDA
void ByteRecibido();
```

```
// Codigo de funciones de NYDIANET
// Alpha 2.0 - 31 Julio 2006

#include "cab20.h"

/*****************************************************
   Variables
*****************************************************/
// Estados del bus de datos
static enum estados_bus {BUS_OCIOSO=0xF9, BUS_ESP_DEST=0xFA,
                         BUS_ESP_ORIG=0xFB, BUS_ESP_CTRL1=0xFC,
                         BUS_ESP_CTRL2=0xFD, BUS_ESP_LONG=0xFE,
                         BUS_ESP_CKSM=0xFF} EstadoBus = BUS_OCIOSO;

// Tabla de entradas
static int8 Ent[MAX_ENTRADAS];

// Tabla de salidas
static int8 Sal[MAX_ENTRADAS];

// Tabla de modos
static int8 Mod[MAX_ENTRADAS];

// Tabla de operaciones
static int8 Ope[MAX_ENTRADAS];

// Tabla de direcciones
static int8 Dir[MAX_ENTRADAS];

// Configuación por defecto
static int8 const DefConfig[27] =
   {T_DIR_PROP, 0x19, C_IN_AN0, C_OUT_B1, M_LOC_DIG_NOR_DIM, 0, 0,
                      C_IN_AN1, C_OUT_B2, M_LOC_DIG_NOR_DIM, 0, 0,
                      C_IN_AN2, C_OUT_B5, M_LOC_DIG_NOR_TOG, 0, 0,
                      C_IN_RC0, C_OUT_B5, M_LOC_DIG_NOR_TOG, 0, 0,
                      C_IN_RC0, C_OUT_B6, M_LOC_DIG_NOR_TOG, 0, 0};

// Almacena el estado de las entradas en el ciclo de trabajo
static int8 EntradaActual=0xFF;

// Almacena el estado de las entradas en el anterior ciclo
static int8 EntradaAnterior=0xFF;

// Contadores de duracion de pulsacion
static int8 ContPulsacion[NUM_ENTRADAS];

// Operaciones con los contadores
static enum eOpCont {Nop=0x00, Dec=0x01, Dec5=0x02, Res=0x03}
   OpCont[NUM_ENTRADAS];

// Operaciones con las salidas
static enum eOperacion {ConmutarL=0x01, ApagarL=0x02, EncenderL=0x03,
          CPotenciaL=0x04, FPotenciaL=0x05, ConmutarR=0x06,
          ApagarR=0x07, EncenderR=0x08,CPotenciaR=0x09,
          FPotenciaR=0x0A, CPotenciaL5=0x0B, Ninguna=0x00}
          Operacion[NUM_SALIDAS];

// Estado de las salidas locales
static enum eEstado {Apagado=0x00, Encendido=0xFF} Estado[NUM_SALIDAS];

// Sentido del cambio del potencia del dimmer (para que sea ciclico)
static int8 SentidoCambio=0x03;

// Potencia actual de cada dimmer
static int8 PotDimmer[2]={0,0};
```

```
// Buffer de Entrada de tramas
static int8 BufferEntrada[MAXMENSAJES][MAXLONGITUD];

// Almacena tramas temporalmente mientras se reciben
static int8 PreBufferEntrada[MAXLONGITUD];

// Buffer de Salida de tramas
static int8 BufferSalida[MAXMENSAJES][MAXLONGITUD];

// Trama pendiente de asentir
static int8 Ventana[MAXLONGITUD];

// Indice para usar el buffer de Entrada
static int8 SiguienteHuecoE = 0;

// Indice para usar el buffer de Salida
static int8 SiguienteHuecoS = 0;

// Flag de buffer de entrada lleno
static int1 BufferLleno = FALSE;

// Flag para indicar trama recibida
static int1 TramaRecibida = FALSE;

// Contador de bytes de datos a recibir por cada trama
static int8 UltimoByte;

// Direccion de envio para instrucciones remotas
static int8 DirEnvio[NUM_SALIDAS];

// Checksum para tramas recibidas
static int8 Checksum;

// Campo control para tramas enviadas
static int8 Control[NUM_SALIDAS][2];

// Longitud para tramas enviadas
static int8 Longitud[NUM_SALIDAS];

// Datos para tramas enviadas
static int8 Info[NUM_SALIDAS][MAXLONGITUD-HEADER_SIZE];

// Enable del timeout de recepcion
static int1 RxToutCEnable = FALSE;

// Enable del timeout de asentimiento tras envio
static int1 AckToutCEnable = FALSE;

// Contador del timeout de recepcion
static int8 RxToutCValue = 0;

// Contador del timeout de aasentimiento
static int8 AckToutCValue = 0;

// Numero de nodos detectados hasta el momento en la red
static int8 NumeroNodos = 0;

// Numero de entradas leidas de la configuracion
static int8 NumeroEntradas = 0;

// Dir, Ultimo NT recibido, NR esperado, TipoUltima
static int8 Direcciones[MAX_NODOS][4];

// Flag de ventana llena
static int1 VentanaLlena = FALSE;
```

```
// Contador de retransmisiones
static int8 Retransmisiones = 0;

// Flag para saber cuando un envio es una retransmision
static int1 EsReTx = FALSE;

// Flag para captura de direcciones de tramas no propias
static int1 SoloCapturaDirOrig = FALSE;

// Direccion a capturar de tramas no propias
static int8 DirAComprobar = 0;

// Flag de indicación de operacion con salida remota
static int8 RemoteOp[NUM_SALIDAS];

// Direccion propia leida de EEPROM
static int8 DireccionPropia = 0;



/*******************************************************
   Funciones (ver detalles en cab20.h)
*******************************************************/
void Inicializacion() // Inicialización
{
   int i;

   // Inicializamos las tablas relacionadas con entradas
   for(i=0; i<NUM_ENTRADAS; i++)
   {
      ContPulsacion[i]=0;
      OpCont[i]=Nop;
   }

   // Inicializamos las tablas relacionadas con salidas
   for(i=0; i<NUM_SALIDAS; i++)
   {
      RemoteOp[i]=FALSE;
   }
}
int1 ExisteConfig() // Comprueba si existe config en EEPROM
{
   int1 res;
   int8 dat;

   // EEPROM[0] debe ser la dirección propia
   dat = read_eeprom(0);
   if(dat == NULL_BYTE || dat == FULL_BYTE)
      res = FALSE; // Si es 0 o 255 es que no existe configuración
   else
      res = TRUE;

   return res;
}

void CreaConfigDefecto() // Crea configuracion por defecto en EEPROM
{
   int i;

   for(i=0;i<DefConfig[1]+2;i++) // Copiamos la configuracion en la EEPROM
   {
      write_eeprom(i,DefConfig[i]);
   }
   write_eeprom(S1_DIR, 0x00); // Inicialmente las salidas estarán apagadas
   write_eeprom(S2_DIR, 0x00);
   write_eeprom(S3_DIR, 0x00);
```

```
   write_eeprom(S4_DIR, 0x00);
   write_eeprom(S5_DIR, 0x00);
   write_eeprom(S6_DIR, 0x00);
   write_eeprom(S7_DIR, 0x00);
   write_eeprom(S8_DIR, 0x00);
}

void LeeConfig() // Lee configuracion de la EEPROM
{
   int i, j, longconf;

   NumeroEntradas = 0; // Reseteamos contador
   DireccionPropia = read_eeprom(0); // Dirección propia en EEPROM[0]
   // Longitud de la configuración restante en EEPROM[1]
   longconf = read_eeprom(1);

   for(i=2,j=0; i<longconf+2; i+=5, j++) // Resto de la configuración
   {
      Ent[j] = read_eeprom(i);  // Entrada
      Sal[j] = read_eeprom(i+1);// Salida
      Mod[j] = read_eeprom(i+2);// Modo
      Ope[j] = read_eeprom(i+3);// Operacion
      Dir[j] = read_eeprom(i+4);// Direccion
      NumeroEntradas++; // Actualizamos contador de entradas
   }
   Estado[0] = read_eeprom(S1_DIR); // Cargamos estado de las salidas
   Estado[1] = read_eeprom(S2_DIR);
   Estado[2] = read_eeprom(S3_DIR);
   Estado[3] = read_eeprom(S4_DIR);
   Estado[4] = read_eeprom(S5_DIR);
   Estado[5] = read_eeprom(S6_DIR);
   Estado[6] = read_eeprom(S7_DIR);
   Estado[7] = read_eeprom(S8_DIR);
   // Ordenamos el encendido de las que deban estar en dicho estado
   for(i=0; i< NUM_SALIDAS; i++)
   {
      if(Estado[i] == 0xFF)
         Operacion[i] = EncenderL;
      else
         Operacion[i] = Ninguna;
   }
}

// Crea una trama nueva en el buffer
int1 CreaTrama(int8 dirdest, int8 *control, int8 longitud, int8 *datos)
{
   int1 res;
   int i=0;
   int8 CS;
   int16 DirBase;

   if(SiguienteHuecoS<MAXMENSAJES)
   {
      // Para optimizar el indexado
      DirBase=_mul(SiguienteHuecoS, MAXLONGITUD);
      *(BufferSalida+DirBase)=T_FLAG; // Inserción del Flag
      *(BufferSalida+DirBase+1)=dirdest; // Dirección destino
      CS = (dirdest ^ FULL_BYTE); // Comienza cálculo del Checksum
      *(BufferSalida+DirBase+2)=DireccionPropia; // Dirección origen
      CS^=DireccionPropia;
      *(BufferSalida+DirBase+3)=control[0]; // Primer byte de control
      CS^=control[0];
      *(BufferSalida+DirBase+4)=control[1]; // Segundo byte de control
      CS^=control[1];
      *(BufferSalida+DirBase+5)=longitud; // Longitud de los datos
      CS^=longitud;
```

```
      for(i=0; i < longitud; i++) // Datos
      {
         *(BufferSalida+DirBase+6+i)=datos[i];
         CS^=datos[i];
      }
      *(BufferSalida+DirBase+6+longitud)=CS; // Checksum
      SiguienteHuecoS++; // Actualizamos siguiente hueco libre
      res=TRUE; // Devolvemos TRUE si se ha podido crear la trama
   }
   else
      res=FALSE; // FALSE si no se ha podido crear

   return res;
}

// Introduce una direccion nueva en la lista de direcciones
void InsertaDir(int8 Dir)
{
   int i;
   int1 existe;

   existe = FALSE; // Inicializamos la variable local
   for(i=0; i<NumeroNodos; i++) // Recorremos la lista de direcciones
   {
      if(Direcciones[i][0]==Dir) // Si la encuentra lo notifica
         existe = TRUE;
   }
   if(existe == FALSE) // Si no la ha encontrado la introduce
   {
      Direcciones[NumeroNodos][0]=Dir;  // Dirección
      Direcciones[NumeroNodos][1]=0x00; // NT ultimo recibido
      Direcciones[NumeroNodos][2]=0x00; // NR esperado (ultimo NT enviado)
      // Tipo ultima trama enviada a esa direccion
      Direcciones[NumeroNodos][3]=T_TACK;
      NumeroNodos++; // Nuevo nodo detectado
   }
}

int BuscaDir(int8 Dir) // Busca una direccion en la lista de direcciones
{
   int i;
   int1 encontrado;

   i = 0;
   encontrado = FALSE; // Inicializamos las variables locales

   while(encontrado == FALSE)// Hasta que no se encuentre (esta funcion se
            // llamará siempre tras la de inserción de
            // direcciones, por lo que siempre la
            // encontrará)
   {
      if(Direcciones[i][0]==Dir) // Cuando la encuentre
         encontrado = TRUE;
      else
         i++;
   }
   return i; // Devuelve la posicion donde esta la direccion buscada
}

//#task (rate = 25ms)
void CapturaEntradas() // Captura y procesa las entradas activas
{
   int i;
   int8 temp1, temp2, temp3; // Para el procesado de la lectura de puertos
   int8 Ent_i, Sal_i; // Para optimizar los accesos indexados
```

```
temp1 = PORTA; // Puerto A
temp2 = temp1; // Copia de Puerto A
temp3 = PORTC; // Puerto C
// Calculamos "EntradaActual" = -,-,C0,A5,A3,A2,A1,A0;  Un bit por entrada
EntradaActual = ((temp1 & 0b00001111) | ((temp2 & 0b00100000)>>1)
          | ((temp3 & 0b00000001)<<5));

for(i=0; i<6; i++) // Actualizamos los contadores
{
   if(!bit_test(EntradaActual,i)) // Si esta pulsada
      ContPulsacion[i]++; // Contar
}

for(i=0; i<NumeroEntradas; i++) // Para cada entrada
{
   Ent_i = Ent[i]; // Para optimizar el indexado
   Sal_i = Sal[i]; // Para optimizar el indexado

   switch(Mod[i]) // Dependiendo del modo de operacion
   {
      case NULL_BYTE: // Si no hay asignado ningun modo
         break;
      case M_LOC_DIG_NOR_DIM: // Si es local digital normal dimmer
         if(ContPulsacion[Ent_i] >= PULS_EXTRA_LARGA) // Pulsacion larga
         {
            Operacion[Sal_i]=CPotenciaL; // Se encarga el cambio de pot
            OpCont[Ent_i] = Dec; // Se ordena restar un ciclo
         }
         if(!bit_test(EntradaAnterior,Ent_i) &&
               bit_test(EntradaActual,Ent_i)) // Si se suelta
         {
            if(ContPulsacion[Ent_i] <= PULS_CORTA) // Pulsacion corta
               Operacion[Sal_i] = ConmutarL; // Conmutar
               if(ContPulsacion[Ent_i] > PULS_CORTA
               && ContPulsacion[Ent_i] < PULS_EXTRA_LARGA-1) // Larga
            {
               if(Estado[Sal_i]==Encendido) // Larga: Potencia maxima
                  PotDimmer[Sal_i]=5; // Evitamos el cero
               else // Si esta apagada, encender
                  Operacion[Sal_i] = ConmutarL;
            }
            OpCont[Ent_i] = Res; // Tras soltar, contador a cero
         }
         break;
      case M_LOC_DIG_NOR_TOG: // Modo local digital normal conmutacion
         if(!bit_test(EntradaAnterior,Ent_i) &&
               bit_test(EntradaActual,Ent_i)) // Si se suelta
         {
            if(ContPulsacion[Ent_i] <= PULS_CORTA) // Pulsacion corta
               Operacion[Sal_i] = ConmutarL; // Conmutar
               if(ContPulsacion[Ent_i] > PULS_CORTA) // Larga
               Operacion[Sal_i] = ConmutarL; // Hace lo mismo
               OpCont[Ent_i] = Res; // Tras soltar, contador a cero
         }
         break;
      case M_LOC_DIG_NOR_PUL: // Modo local digital normal pulsador
         if(bit_test(EntradaAnterior,Ent_i)
         && !bit_test(EntradaActual,Ent_i))
            // Enciente en el flanco de bajada
            Operacion[Sal_i]=EncenderL;
         if(!bit_test(EntradaAnterior,Ent_i)
         && bit_test(EntradaActual,Ent_i))
            // Apagar en el flaco de subida
            Operacion[Sal_i]=ApagarL;
         break;
      // Los modos remotos son analogos a los locales pero
```

```
// se incluye informacion para las tramas
case M_REM_DIG_NOR_DIM: // Modo remoto digital normal regulado
   if(ContPulsacion[Ent_i] >= PULS_EXTRA_LARGA) // Pulsacion larga
   {
      Operacion[Sal_i] = CPotenciaR; // Cambiar potencia remota
      DirEnvio[Sal_i] = Dir[i]; // Dirección destino
      Longitud[Sal_i] = 1; // Longitud de los datos
      Info[Sal_i][0] = Sal_i; // Datos
      OpCont[Ent_i] = Dec5; // Se restan 5 ciclos
      RemoteOp[Sal_i] = TRUE; // Avisamos de operación remota
   }
   if(!bit_test(EntradaAnterior,Ent_i) &&
         bit_test(EntradaActual,Ent_i)) // Al soltar
   {
      if(ContPulsacion[Ent_i] <= PULS_CORTA) // Pulsacion corta
      {
         Operacion[Sal_i] = ConmutarR; // Conmutacion remota
         DirEnvio[Sal_i] = Dir[i];
         Longitud[Sal_i] = 1;
         Info[Sal_i][0]= Sal_i;
         RemoteOp[Sal_i] = TRUE;
      }
      if(ContPulsacion[Ent_i] > PULS_CORTA
      && ContPulsacion[Ent_i] < PULS_EXTRA_LARGA-5) // Larga
      {
         Operacion[Sal_i] = FPotenciaR; // Fijar potencia remota
         DirEnvio[Sal_i] = Dir[i];
         Longitud[Sal_i] = 2;
         Info[Sal_i][0]= Sal_i;
         Info[Sal_i][1]= 5;
         RemoteOp[Sal_i] = TRUE;
      }
   }
   OpCont[Ent_i] = Res; // Tras soltar, contador a cero
   break;
case M_REM_DIG_NOR_TOG: // Modo remoto digital normal conmutado
   if(!bit_test(EntradaAnterior,Ent_i) &&
         bit_test(EntradaActual,Ent_i)) // Al soltar el pulsador
   {
      if(ContPulsacion[Ent_i] <= PULS_CORTA) // Pulsacion corta
      {
         Operacion[Sal_i] = ConmutarR; // Conmutación remota
         DirEnvio[Sal_i] = Dir[i];
         Longitud[Sal_i] = 1;
         Info[Sal_i][0]= Sal_i;
         RemoteOp[Sal_i] = TRUE;
      }
      if(ContPulsacion[Ent_i] > PULS_CORTA) // Larga
      {
         Operacion[Sal_i] = ConmutarR; // Actua igual
         DirEnvio[Sal_i] = Dir[i];
         Longitud[Sal_i] = 1;
         Info[Sal_i][0]= Sal_i;
         RemoteOp[Sal_i] = TRUE;
      }
      OpCont[Ent_i] = Res; // Tras soltar, contador a cero
   }
   break;
case M_REM_DIG_NOR_PUL: // Modo remoto digital normal pulsador
   if(bit_test(EntradaAnterior,Ent_i)
   && !bit_test(EntradaActual,Ent_i))
   {
      // En el flanco de bajada se envia encendido
      Operacion[Sal_i] = EncenderR;
      DirEnvio[Sal_i] = Dir[i];
      Longitud[Sal_i] = 1;
```

```
            Info[Sal_i][0]= Sal_i;
            RemoteOp[Sal_i] = TRUE;
         }
         if(!bit_test(EntradaAnterior,Ent_i)
         && bit_test(EntradaActual,Ent_i))
         {
            // En el flanco de subida se envia apagado
            Operacion[Sal_i] = ApagarR;
            DirEnvio[Sal_i] = Dir[i];
            Longitud[Sal_i] = 1;
            Info[Sal_i][0]= Sal_i;
            RemoteOp[Sal_i] = TRUE;
         }
         break;
   }
   switch(Ope[i]) // Dependiendo de la operación asignada
   {
      case NULL_BYTE: // Si no hay ninguna asignada
         break;
      case O_LOC_ASAL: // Operación local de activacion de salida
         if(bit_test(EntradaAnterior,Ent_i)
         && !bit_test(EntradaActual,Ent_i))
            Operacion[Sal_i]=EncenderL;
         break;
      case O_LOC_DSAL: // Operación local de desactivacion de salida
         if(bit_test(EntradaAnterior,Ent_i)
         && !bit_test(EntradaActual,Ent_i))
            Operacion[Sal_i]=ApagarL;
         break;
      case O_REM_ASAL: // Operación remota de activación de salida
         if(bit_test(EntradaAnterior,Ent_i)
         && !bit_test(EntradaActual,Ent_i))
         {
            Operacion[Sal_i] = EncenderR; // Encendido remoto
            DirEnvio[Sal_i] = Dir[i];
            Longitud[Sal_i] = 1;
            Info[Sal_i][0]= Sal_i;
            RemoteOp[Sal_i] = TRUE;
         }
         break;
      case O_REM_DSAL: // Operación  remota de desactivación de salida
         if(!bit_test(EntradaAnterior,Ent_i)
         && bit_test(EntradaActual,Ent_i))
         {
            Operacion[Sal_i] = ApagarR; // Apagado remoto
            DirEnvio[Sal_i] = Dir[i];
            Longitud[Sal_i] = 1;
            Info[Sal_i][0]= Sal_i;
            RemoteOp[Sal_i] = TRUE;
         }
         break;
   }
}

for(i=0; i<NUM_ENTRADAS; i++) // Modificamos los contadores
{
   switch(OpCont[i]) // Segun la operacion encargada
   {
      case Dec: // Decrementar (para los dimmer locales)
         ContPulsacion[i]--;
         OpCont[i] = Nop;
         break;
      case Dec5: // Decrementar 5 (para los dimmer remotos)
         ContPulsacion[i]-=5;
         OpCont[i] = Nop;
         break;
```

```
         case Res: // Poner a cero
            ContPulsacion[i] = 0;
            OpCont[i] = Nop;
            break;
         case Nop: // Ninguna operacion
            break;
      }
   }
   // Guardamos estado actual de las entradas
   EntradaAnterior = EntradaActual;
}

//#task (rate = 25ms)
void ActivaSalidas() // Realiza las operaciones indicadas para cada salida
{
   int i, indiceDir[NUM_SALIDAS];
   int8 NTSig[NUM_SALIDAS], NR[NUM_SALIDAS];
   // Para cada salida miramos si tiene asiganada una operacion remota
   for(i=0; i<NUM_SALIDAS; i++)
   {
      if(RemoteOp[i] == TRUE) // Si existe alguna operación remota
      {
         InsertaDir(DirEnvio[i]);  // Metemos la direccion en la tabla por
                                   // si es la primera vez que se envía
                                   // una trama a esa dirección
         indiceDir[i] = BuscaDir(DirEnvio[i]);  // Extraemos su indice y
                                    // actualizamos los NT y NR
         NTSig[i] = Direcciones[indiceDir[i]][2]+0x10; // NR esperado + 1
         NR[i] = Direcciones[indiceDir[i]][1];
         if(NTSig[i]==0x00) // El NT debe ser ciclico
            NTSig[i]=0x10;
         RemoteOp[i]=FALSE; // Desactivamos el flag
      }
   }

   for(i=0; i<NUM_SALIDAS; i++) // Para cada salida
   {
      switch(Operacion[i]) // En funcion de la operación
      {
         case ConmutarL: // Conmutar localmente
            if(Estado[i] == Apagado)
            {
               Estado[i] = Encendido;
               write_eeprom(i+0x03F8, 0xFF);
               if(i<7 && i>1) // Salidas sin dimmer en PORTB
                  bit_set(PORTB, i+1);
               else if(i==7) // Salida por PORTA
                  bit_set(PORTA, 4);
            }
            else
            {
               Estado[i] = Apagado;
               write_eeprom(i+0x03F8, 0x00);
               if(i<7 && i>1) // Salidas sin dimmer en PORTB
                  bit_clear(PORTB, i+1);
               else if(i==7) // Salida por PORTA
                  bit_clear(PORTA, 4);
            }
            Operacion[i] = Ninguna;
            break;
         case EncenderL: // Encender localmente
            if(i<7 && i>1) // Salidas sin dimmer en PORTB
               bit_set(PORTB, i+1);
            else if(i==7) // Salida por PORTA
               bit_set(PORTA, 4);
            Estado[i] = Encendido;
```

```
      write_eeprom(i+0x03F8, 0xFF);
      Operacion[i] = Ninguna;
      break;
   case ApagarL: // Apagar localmente
      if(i<7 && i>1) // Salidas sin dimmer en PORTB
         bit_clear(PORTB, i+1);
      else if(i==7) // Salida por PORTA
         bit_clear(PORTA, 4);
      Estado[i] = Apagado;
      write_eeprom(i+0x03F8, 0x00);
      Operacion[i] = Ninguna;
      break;
   case CPotenciaL: // Cambiar potencia localmente en un nivel
      CambiarPotDimmer(i,1);
      Operacion[i] = Ninguna;
      break;
   case CPotenciaL5: // Cambiar potencia localmente en 5 niveles
      CambiarPotDimmer(i,5);
      Operacion[i] = Ninguna;
      break;
   case ConmutarR: // Conmutar remotamente
      Control[i][0]=T_TCTL|T_TCTL_CSAL; // Generamos campos de trama
      Control[i][1]=(NTSig[i])|(NR[i]>>4);
      Direcciones[indiceDir[i]][2]=NTSig[i];
      Direcciones[indiceDir[i]][3]=T_TCTL;
      if(CreaTrama(DirEnvio[i], Control[i],
         Longitud[i], Info[i])==TRUE)
         Operacion[i] = Ninguna;
      break;
   case ApagarR: // Apagar remotamente
      Control[i][0]=T_TCTL|T_TCTL_DSAL; // Generamos campos de trama
      Control[i][1]=(NTSig[i])|(NR[i]>>4);
      Direcciones[indiceDir[i]][2]=NTSig[i];
      Direcciones[indiceDir[i]][3]=T_TCTL;
      if(CreaTrama(DirEnvio[i], Control[i],
         Longitud[i], Info[i])==TRUE)
         Operacion[i] = Ninguna;
      break;
   case EncenderR: // Encender remotamente
      Control[i][0]=T_TCTL|T_TCTL_ASAL; // Generamos campos de trama
      Control[i][1]=(NTSig[i])|(NR[i]>>4);
      Direcciones[indiceDir[i]][2]=NTSig[i];
      Direcciones[indiceDir[i]][3]=T_TCTL;
      if(CreaTrama(DirEnvio[i], Control[i],
         Longitud[i], Info[i])==TRUE)
         Operacion[i] = Ninguna;
      break;
   case CPotenciaR: // Cambiar potencia remotamente
      Control[i][0]=T_TCTL|T_TCTL_CPOT; // Generamos campos de trama
      Control[i][1]=(NTSig[i])|(NR[i]>>4);
      Direcciones[indiceDir[i]][2]=NTSig[i];
      Direcciones[indiceDir[i]][3]=T_TCTL;
      if(CreaTrama(DirEnvio[i], Control[i],
         Longitud[i], Info[i])==TRUE)
         Operacion[i] = Ninguna;
      break;
   case FPotenciaR: // Fijar potencia remotamente
      Control[i][0]=T_TCTL|T_TCTL_FPOT; // Generamos campos de trama
      Control[i][1]=(NTSig[i])|(NR[i]>>4);
      Direcciones[indiceDir[i]][2]=NTSig[i];
      Direcciones[indiceDir[i]][3]=T_TCTL;
      if(CreaTrama(DirEnvio[i], Control[i],
         Longitud[i], Info[i])==TRUE)
         Operacion[i] = Ninguna;
      break;
   case Ninguna:
```

```
            break;
         }
      }
   }

//#task (rate = 100ms)
void InterpretaTramas() // Ejecuta instrucciones y procesa datos de cada trama
{
   // Variables locales para almacenar los campos de la trama
   int8 DirOrig; // Dirección origen
   int8 ControlIn[2]; // Campo Control
   int8 ControlAux[2]; // Campos Control Auxiliar (para enviar ACK's)
   int8 LongitudIn; // Campo longitud
   int8 LongitudAux; // Campo longitud auxiliar (para enviar ACK's)
   int8 DatosIn[MAXLONGITUD-HEADER_SIZE]; // Datos de entrada
   int8 DatosAux[1]; // Datos para tramas enviadas (Solo se crean tramas ACK)
   int8 TipoTrama; // Tipo de trama que se está procesando
   int8 i; // Contador
   int8 indiceDir; // Indice para buscar la direccion origen en la tabla
   int1 ProcesarOk = FALSE; // Flag para indicar si una trama  debe procesarse
   int8 NTRecibido, NRRecibido; // Numeros de secuencia recibidos
   int8 NTGuardado; // NT Guardado en la trabla
   int8 NTEsperado, NREsperado; // Valores esperados

   while(SiguienteHuecoE==0) // Mientras el buffer este vacio esperamos
      rtos_yield(); // Devuelve el control al despachador de tareas

   // Extraemos los datos de la trama
   DirOrig=BufferEntrada[SiguienteHuecoE-1][0]; // Origen
   ControlIn[0]=BufferEntrada[SiguienteHuecoE-1][1]; // Control
   ControlIn[1]=BufferEntrada[SiguienteHuecoE-1][2];
   LongitudIn=BufferEntrada[SiguienteHuecoE-1][3]; // Longitud

   for(i=0; i<LongitudIn; i++) // Datos
      DatosIn[i] = BufferEntrada[SiguienteHuecoE-1][4+i];


   InsertaDir(DirOrig); // Metemos la dirección origen si no está
   indiceDir=BuscaDir(DirOrig); // La buscamos en la tabla

   if(DirOrig != T_DIR_DIF) // Las tramas de difusión se saltan este proceso
   {
    TipoTrama=ControlIn[0]&0b11000000; // Extraemos el tipo de trama
    NTRecibido = (ControlIn[1]&MS_4BITS); // Extraemos el NT de la trama
    NRRecibido = ((ControlIn[1]&LS_4BITS)<<4); // Extraemos el NR de la trama
    NTGuardado = Direcciones[indiceDir][1]; // Buscamos valores guardados
    NREsperado = Direcciones[indiceDir][2]; // tanto para NT como para NR
    NTEsperado = NTGuardado + 0x10; // NT esperado = siguiente al guardado

    if(NTEsperado == 0x00) // NT siempre deber ser ciclico
      NTEsperado=0x10;

    if(AckToutCEnable == TRUE) // Si espero ACK
    {
      if(TipoTrama == T_TACK) // Si he recibido TACK
      {
         if(NRRecibido == NREsperado) // Si el NR es correcto
         {
            AckToutCEnable = FALSE; // Paro el timeout de asentimiento
            VentanaLlena = FALSE; // Ventana vacía
            Retransmisiones = 0; // Reseteo contador de retransmisiones
         }
      }
      else // Si he recibido TDAT o TCTL
      {
         if(NTRecibido == NTEsperado) // Si el NT es correcto
```

```
{
   Direcciones[indiceDir][1]=NTRecibido; // Inserto el nuevo NT
   if(NRRecibido == NREsperado) // Si el NR es correcto
   {
      AckToutCEnable = FALSE; // Paro el timeout de asentimiento
      VentanaLlena = FALSE; // Ventana vacía
      ProcesarOk = TRUE; // La trama debe procesarse
      Retransmisiones = 0; // Reset contador de retransmisiones
      // Si hay tramas por enviar las aprovecho para enviar ACK
      if(SiguienteHuecoS!=0)
         BufferSalida[SiguienteHuecoS-1][4]|=(NTRecibido>>4);
      else // Crear trama Ack con el NTRecibido
      {
         ControlAux[0]=T_TACK;  // Generamos campos de trama
         // ACK no incrementa el NT
         ControlAux[1]=(NREsperado)|(NTRecibido>>4);
         LongitudAux=0; // Sin datos
         // Indicamos que hemos enviado ACK
         Direcciones[indiceDir][3]=T_TACK;
         CreaTrama(DirOrig, ControlAux, LongitudAux, DatosAux);
      }
   }
   else // Si el NR es incorrecto
   {
      // Como NT es correcto, se procesa la trama y se asiente
      ProcesarOk = TRUE;
      // Si hay tramas por enviar las aprovecho para enviar ACK
      if(SiguienteHuecoS!=0)
         BufferSalida[SiguienteHuecoS-1][4]|=(NTRecibido>>4);
      else // Crear trama Ack con el NTRecibido
      {
         ControlAux[0]=T_TACK; // Generamos campos de trama
         // ACK no incrementa el NT
         ControlAux[1]=(NREsperado)|(NTRecibido>>4);
         LongitudAux=0; // Sin datos
         // Indicamos que hemos enviado ACK
         Direcciones[indiceDir][3]=T_TACK;
         CreaTrama(DirOrig, ControlAux, LongitudAux, DatosAux);
      }
   }
}
else // Si el NT es incorrecto
{
   ProcesarOk = FALSE; // No se procesa la trama
   if(Direcciones[indiceDir][3]==T_TCTL) // Si envie TDAT o TCTL
   {
      // Si el buffer de salida está lleno espero
      while(SiguienteHuecoS>=MAXMENSAJES)
         rtos_yield();

   // Copiamos la ventana en el buffer de salida para retransmitir
      for(i=0; i < Ventana[5]+7;i++)
         BufferSalida[SiguienteHuecoS][i]=Ventana[i];

      SiguienteHuecoS++; // Actualizamos indice de salida
      EsReTx = TRUE; // Indicamos que es una retransmisión
      Retransmisiones++; // Incrementamos el contador
      if(Retransmisiones>=MAX_RETX) // Si se alcanza el limite
      {
         AckToutCEnable = FALSE; // Desactivo la espera de ACK
         VentanaLlena = FALSE; // Descarto el contenido
         Retransmisiones = 0; // Reseteo del contador
      }
   }
   else // Si envie TACK
   {
```

```
            // Reenviamos el ACK, generandolo nuevamente
            ControlAux[0]=T_TACK;
            ControlAux[1]=(NREsperado)|(NTRecibido>>4);
            LongitudAux=0;
            Direcciones[indiceDir][3]=T_TACK;
            EsReTx = TRUE; // Indicamos que es una retransmisión
            CreaTrama(DirOrig, ControlAux, LongitudAux, DatosAux);
         }
      }
   }
  }
  else // Si no espero ACK
  {
   if(TipoTrama!=T_TACK) // Si he recibido TDAT o TCTL
   {
      if(NTRecibido == NTEsperado) // Si el NT es correcto
      {
         ProcesarOk = TRUE; // Se procesara la trama
         Direcciones[indiceDir][1]=NTRecibido; // Inserto el NT recibido
         // Si hay tramas por enviar las aprovecho para enviar el ACK
         if(SiguienteHuecoS!=0)
         {
            BufferSalida[SiguienteHuecoS-1][4]|=(NTRecibido>>4);
         }
         else // Crear trama Ack con el NTRecibido
         {
            ControlAux[0]=T_TACK;
            ControlAux[1]=(NREsperado)|(NTRecibido>>4);
            LongitudAux=0;
            Direcciones[indiceDir][3]=T_TACK;
            CreaTrama(DirOrig, ControlAux, LongitudAux, DatosAux);
         }
      }
      else // Si el NT es incorrecto
      {
         ProcesarOk = FALSE; // No se procesa la trama
         // Si se envió ACK, se reenvía
         if(Direcciones[indiceDir][3]==T_TACK)
         {
            ControlAux[0]=T_TACK;
            ControlAux[1]=(NREsperado)|(NTGuardado>>4);
            LongitudAux=0;
            Direcciones[indiceDir][3]=T_TACK;
            CreaTrama(DirOrig, ControlAux, LongitudAux, DatosAux);
         }
      }
   }
  }
 }
 else
    ProcesarOk = TRUE; // Las tramas de difusion siempre se procesan

 if(ProcesarOk == TRUE) // Si debo procesar la trama
 {
    switch((ControlIn[0]&0b11000000)) // Miramos el campo control
    {
       case T_TDAT: // Trama de datos de configuracion
          for(i=0;i<DatosIn[1]+2;i++)
             write_eeprom(i,DatosIn[i]); // Se meten en EEPROM
          break;
       case T_TCTL: // Trama de instruccion
          switch(ControlIn[0]&0b00111111)
          {
             case T_TCTL_ASAL: // Activar salida
             // En todos los casos se programa la accion localmente
                Operacion[DatosIn[0]]=EncenderL;
```

```
                  break;
               case T_TCTL_DSAL: // Apagar salida
                  Operacion[DatosIn[0]]=ApagarL;
                  break;
               case T_TCTL_CSAL: // Conmutar salida
                  Operacion[DatosIn[0]]=ConmutarL;
                  break;
               case T_TCTL_CPOT: // Cambiar potencia
                  Operacion[DatosIn[0]]=CPotenciaL5;
                  break;
               case T_TCTL_FPOT: // Fijar potencia
                  PotDimmer[DatosIn[0]]=DatosIn[1];
                  break;
            }
            break;
      }
      SiguienteHuecoE--; // Actualizamos en índice del buffer
      BufferLleno=FALSE; // Desactivamos Flags
      ProcesarOk = FALSE;
   }
}

//#task (rate = 100ms)
void EnviaTramas() // Toma una trama del buffer de salida y la envía
{
   int i;
   int8 byteaux;
   int8 tipo;

   // Se espera mientras el buffer esté vacio o la ventana llena, a menos
   // que se necesite retransmitir
   while((SiguienteHuecoS == 0 || VentanaLlena == TRUE) && EsReTx == FALSE)
      rtos_yield();

   disable_interrupts(INT_RDA); // Desactiva la recepción durante el envío

   for(i=0; i < BufferSalida[SiguienteHuecoS-1][5] + 7; i++)
   {
      byteaux = BufferSalida[SiguienteHuecoS-1][i];
      fputc(byteaux, RS485_BUS); // Enviamos cada byte
      Ventana[i]=byteaux; // Lo almacenamos en la ventana
   }

   enable_interrupts(INT_RDA);

   // Extraemos el tipo enviado
   tipo = (BufferSalida[SiguienteHuecoS-1][3])&0b11000000;

   // Si no se trata de un ACK ni de una retransmision
   if(tipo!=T_TACK && EsReTx == FALSE)
   {
      AckToutCEnable = TRUE; // Activamos el timeout de asentimiento
      // Espera recibir NR = NT enviado durante X ms
      AckToutCValue = ACKTOUT_CICLOS;
      VentanaLlena = TRUE; // Solo si se envia TDAT o TCTL
   }
   SiguienteHuecoS--; // Actualizamos el índice del buffer de salida
   EsReTx = FALSE; // Desactivamos el flag de reenvío
}

//#task (rate = 25ms)
void RecogeTrama() // Extrae tramas del prebuffer y las mete en el buffer
{
   int i;

   while(TramaRecibida==FALSE) // Mientras no se reciban tramas, esperar
```

```
      rtos_yield();

   for(i=0; i < PreBufferEntrada[3]+4;i++) // Copiamos la trama recibida
      BufferEntrada[SiguienteHuecoE][i]=PreBufferEntrada[i];

   SiguienteHuecoE++; // Actualizamos el índice

   if(SiguienteHuecoE>=MAXMENSAJES) // Si el buffer está lleno lo indicamos
      BufferLleno=TRUE;

   TramaRecibida = FALSE; // Desactivamos el flag
}

//#task (rate = 50ms)
void Contadores() // Manejador de contadores
{
   int i;

   if(RxToutCEnable==TRUE) // Timeout de recepción
   {
      if(--RxToutCValue==0) // Si expira
      {
         EstadoBus=BUS_OCIOSO; // Dejamos de esperar el resto de la trama
         RxToutCEnable=FALSE; // Lo desactivamos
         RxToutCValue=RXTOUT_CICLOS; // Reiniciamos para próxima ejecucion
      }
   }
   if(AckToutCEnable==TRUE) // Timeout de asentimiento
   {
      if(--AckToutCValue==0) // Si expira
      {
         for(i=0; i < Ventana[5]+7;i++) // Reenviamos lo que haya en ventana
            BufferSalida[SiguienteHuecoS][i]=Ventana[i];
         SiguienteHuecoS++; // Actualizamos el índice
         EsReTx = TRUE; // Indicamos que se trata de una retransmisión
         Retransmisiones++; // Incrementamos el contador de retransmisiones
         AckToutCValue=ACKTOUT_CICLOS; // Reiniciamos contador
         if(Retransmisiones>=MAX_RETX) // Si se alcanza el limite de retx
         {
            AckToutCEnable = FALSE; // Dejamos de esperar el asentimiento
            VentanaLlena = FALSE; // Descartamos la trama
            Retransmisiones = 0; // Reseteamos el contador
         }
      }
   }
}

// Cambia cíclicamente el valor de los dimmer
void CambiarPotDimmer(int dimmer, int cantidad)
{
   int cambio;

   cambio = cantidad;
   if(cambio != 1 && cambio != 5)
      cambio = 1;  // Si se introduce un valor erroneo
                   // por defecto se cambia en una unidad
   if(bit_test(SentidoCambio,dimmer)) // Miramos el bit de sentido de cambio
      PotDimmer[dimmer]+=cambio; // Incremento
   else
      PotDimmer[dimmer]-=cambio; // Decremento
   if(PotDimmer[dimmer] == DIMMER_POT_VMAX) // Si se alcanza el valor máximo
      bit_clear(SentidoCambio,dimmer); // Indicamos que debemos decrementar
   if(PotDimmer[dimmer] == DIMMER_POT_VMIN) // Si se alcanza el valor mínimo
      bit_set(SentidoCambio,dimmer); // Indicamos que debemos incrementar
}
```

```
#int_EXT
void PasoPorCero() // Se ejecuta en los pasos por cero de la señal electrica
{
   // Dimmer 1
   if(Estado[0] == Encendido && PotDimmer[0] < DIMMER_POT_VMIN+DIMMER_POT_GAP)
   // Si la potencia es proxima al valor mínimo, encendemos
      output_high(PIN_B1);
   if(Estado[0] == Apagado || PotDimmer[0] > DIMMER_POT_VMAX-DIMMER_POT_GAP)
   // Si la potencia es proxima al valor máximo, apagamos
      output_low(PIN_B1);
   if(Estado[0] == Encendido
   && (PotDimmer[0] >= DIMMER_POT_VMIN+DIMMER_POT_GAP)
   && (PotDimmer[0] <= DIMMER_POT_VMAX-DIMMER_POT_GAP)) // En otro caso
   {
      output_low(PIN_B1); // Apagamos el triac
      // Configuramos el tiempo de espera
      set_timer1(OVERFLOW-(_mul(PotDimmer[0],DIMMER_SCALE)));
      clear_interrupt(INT_EXT); // Limpiamos la interrupción externa
      enable_interrupts(INT_TIMER1); // Activamos el timer overflow
   }
   // Dimmer2 (análogo al dimmer 1)
   if(Estado[1] == Encendido && PotDimmer[1] < DIMMER_POT_VMIN+DIMMER_POT_GAP)
      output_high(PIN_B2);
   if(Estado[1] == Apagado || PotDimmer[1] > DIMMER_POT_VMAX-DIMMER_POT_GAP)
      output_low(PIN_B2);
   if(Estado[1] == Encendido
   && (PotDimmer[1] >= DIMMER_POT_VMIN+DIMMER_POT_GAP)
   && (PotDimmer[1] <= DIMMER_POT_VMAX-DIMMER_POT_GAP))
   {
      output_low(PIN_B2);
      set_timer3(OVERFLOW-(_mul(PotDimmer[1],DIMMER_SCALE)));
      clear_interrupt(INT_EXT);
      enable_interrupts(INT_TIMER3);
   }
}

#int_TIMER1
void Dimmer1() // Para el control de intensidad
{
   output_high(PIN_B1); // Apagamos el triac
   set_timer1(0); // Reseteamos el timer
   disable_interrupts(INT_TIMER1); // Desactivamos la interrupción
   clear_interrupt(INT_TIMER1); // Limpiamos el flag de interrupción
}

#int_TIMER3
void Dimmer2() // Para el control de intensidad
{
   output_high(PIN_B2); // Apagamos el triac
   set_timer3(0); // Reseteamos el timer
   disable_interrupts(INT_TIMER3); // Desactivamos la interrupción
   clear_interrupt(INT_TIMER3); // Limpiamos el flag de interrupción
}

#int_RDA
// Se ejecuta cuando se recibe un byte en el buffer hardware del micro
void ByteRecibido()
{
   int8 Dato;

   // Desactivamos la recepción de mas bytes mientras procesamos este
   disable_interrupts(INT_RDA);

   // Si hay sitio en el buffer y prebuffer
   if(BufferLleno == FALSE && TramaRecibida == FALSE)
   {
```

```
if(kbhit()) // Si el byte está listo
{
   Dato = fgetc(RS485_BUS); // Extraemos el dato
   switch(EstadoBus) // Segun en qué estado se encuentre el bus
   {
      case BUS_OCIOSO: // Bus ocioso, esperando inicio de trama
         if(Dato == T_FLAG) // Si el byte recibido es un flag
            EstadoBus=BUS_ESP_DEST; // Espero dirección destino
            RxToutCEnable=TRUE; // Activo el timeout de recepción
            // Cargamos el valor en el contador
            RxToutCValue=RXTOUT_CICLOS;
         break;
      case BUS_ESP_DEST: // Esperando dirección destino
         // Si es direccion propia o difusión
         if(Dato == DireccionPropia || Dato == T_DIR_DIF)
         {
            // Comienzo a calcular el checksum
            Checksum=(Dato ^ FULL_BYTE);
            // No solo capturo la direccion
            SoloCapturaDirOrig = FALSE;
         }
         else // Si la trama no es para este nodo
            SoloCapturaDirOrig = TRUE; // Sólo capturo destino
         EstadoBus=BUS_ESP_ORIG; // Paso a esperar dirección origen
         break;
      case BUS_ESP_ORIG: // Si estoy esperando el origen
         // Si sólo tenemos que capturar la dirección origen
         if(SoloCapturaDirOrig == TRUE)
         {
            DirAComprobar = Dato; // Pasamos la dirección
            // Desactivamos el timeout de recepción
            RxToutCEnable=FALSE;
            RxToutCValue=RXTOUT_CICLOS; // Reiniciamos su valor
            EstadoBus = BUS_OCIOSO; // El bus pasa a estado ocioso
            SoloCapturaDirOrig = FALSE; // Desactivamos el flag
         }
         else // Si es una trama para este nodo
         {
            // La dirección origen es el primer dato almacenado
            PreBufferEntrada[0] = Dato;
            Checksum^=Dato; // Continuamos calculando el checksum
            // Pasamos a esperar el primer byte del campo control
            EstadoBus=BUS_ESP_CTRL1;
         }
         break;
      // Si estoy esperando el primer byte del campo control
      case BUS_ESP_CTRL1:
         PreBufferEntrada[1] = Dato; // Lo insertamos en prebuffer
         Checksum^=Dato; // Cálculo del checkum
         // Pasamos a esperar el segundo byte del campo control
         EstadoBus=BUS_ESP_CTRL2;
         break;
      // Si estoy esperando el segundo byte del campo control
      case BUS_ESP_CTRL2:
         PreBufferEntrada[2] = Dato; // Lo insertamos en prebuffer
         Checksum^=Dato; // Cálculo del checkum
         // Pasamos a esperar el campo longitud
         EstadoBus=BUS_ESP_LONG;
         break;
      case BUS_ESP_LONG: // Si estoy esperando el campo longitud
         if(Dato!=0x00) // Si hay datos
         {
            // Valor del indice total del ultimo byte de datos
            UltimoByte = Dato+3;
            // Usamos la variable de estado para indexar los datos
            EstadoBus = 0x04;
```

```
            }
            else // Si no hay datos
               EstadoBus = BUS_ESP_CKSM; // Esperamos el checksum

            // Metemos la longitud en prebuffer
            PreBufferEntrada[3] = Dato;
            Checksum^=Dato; // Cálculo del checkum
            break;
         // El resto de casos se corresponden con la recepción de datos
         default:
            if(EstadoBus==BUS_ESP_CKSM) // Si estoy esperando checksum
            {
               if(Checksum==Dato) // Si el checksum es correcto
               {
                  // Bus ocioso, trama recibida completa
                  EstadoBus=BUS_OCIOSO;
                  // Activamos el flag de recepción
                  TramaRecibida = TRUE;
                  // Reiniciamos el indice del ultimo byte
                  UltimoByte=0;
                  // Desactivamos el timeout de recepción
                  RxToutCEnable=FALSE;
                  // Reiniciamos su valor
                  RxToutCValue=RXTOUT_CICLOS;

               }
            }
            // Si aún estoy recibiendo datos
            else if(EstadoBus<BUS_OCIOSO)
            {
               // Metemos el dato en el prebuffer
               PreBufferEntrada[EstadoBus] = Dato;
               // Calculamos el checksum
               Checksum^=Dato;
               // Incrementamos el índice
               EstadoBus++;
               if(EstadoBus>UltimoByte) // Si se ha alcanzado el final
               // Pasamos a esperar el checksum
                  EstadoBus=BUS_ESP_CKSM;
            }
            break;
      }
   }
   clear_interrupt(INT_RDA); // Limpiamos y reiniciamos la interrupción
   enable_interrupts(INT_RDA);
}

////////////// MAIN /////////////////
void main (void)
{
   // Inicialización del microcontrolador
   setup_wdt(WDT_OFF);
   setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);
   setup_timer_2(T2_DIV_BY_16,255,12);
   setup_timer_3(T3_INTERNAL|T3_DIV_BY_1);
   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_OFF|ADC_TAD_MUL_0);
   setup_ccp1(CCP_OFF);
   setup_ccp2(CCP_OFF);

   // Configuración de las entradas y salidas
   set_tris_a(0x2F); // 0, 1, 2, 3 y 5 entradas
   set_tris_b(0x01); // Todo salidas menos 0

   // Desactivación de salidas
```

```
   PORTB = 0b00000000;
   bit_clear(PORTA, 5);

   // Inicialización de variables globales
   Inicializacion();

   // Comprobación de existencia de configuración
   if(ExisteConfig()) // Si existe
      LeeConfig(); // Se lee
   else // Si no
   {
      CreaConfigDefecto(); // Se crea una por defecto
      LeeConfig(); // Se lee
   }

   // Cargamos las interrupciones
   enable_interrupts(INT_RDA);
   enable_interrupts(INT_EXT);
   ext_int_edge(L_TO_H);
   enable_interrupts(GLOBAL);

   // Iniciamos el sistema operativo en tiempo real
   rtos_run();
}
```