ANEXOS

ANEXO A: Ejemplo de análisis de las tramas del protocolo PLC Lite

Se analizaron las tramas más relevantes que nos permitían cambiar y obtener los valores de los parámetros configurables, así como las tramas que nos permitían mandar y recibir datos.

Como ejemplo, se analiza la trama get_system_info que pregunta al módem la configuración y éste contesta con la información de la configuración. Los datos se han obtenido con el propio software de TI poniendo formato raw en la visualización.

El resultado mostrado por pantalla es el siguiente:

```
2017-04-14 11:49:35.8558:    Message Sent:
       0x01 80 04 00 2A 81 00 00
2017-04-14 11:49:35.8558:    Sending: (0x01) -
SystemInformation.Request:
2017-04-14 11:49:35.8871:    Message Received:
0x01 00 3C 00 4C 36 00 00 05 01     0x00 04 00 00 00 00 00 00 00 00
0x00 00 00 00 00 00 00 00 00 00     0x05 03 00 00 00 00 00 00 00 00
0x00 01 00 00 00 00 00 00 00 00     0x01 00 18 00 01 00 87 00 00 00
0x00 00 00 00
2017-04-14 11:49:35.8871:    Receiving: (0x01) - SystemInformation:
       Firmware Version: Major:4 Minor:0 Revision:1 Build:5
       Serial Number Length: 0
       Serial Number:        0x
       Device Type:          FLEX Lite
       Device Mode:          Point To Point
       Hardware Revision:    0
       Diagnostics Port:     SCI A
       Data Port:            SCI A
       PHY Mode:             ROBO
       Band:                 HalfBand
       Start Tone:           0087
       Address Enable:       False
       Address Size:         1
       Address Offset:       24
       Address One:          0x00
       Address Two:          0x00
       Address Three:        0x00
```

La primera trama, se envía desde el ordenador al módem, los cuatro primeros números hexadecimales son la cabecera y los cuatro siguientes son los códigos CRC. En detalle son:

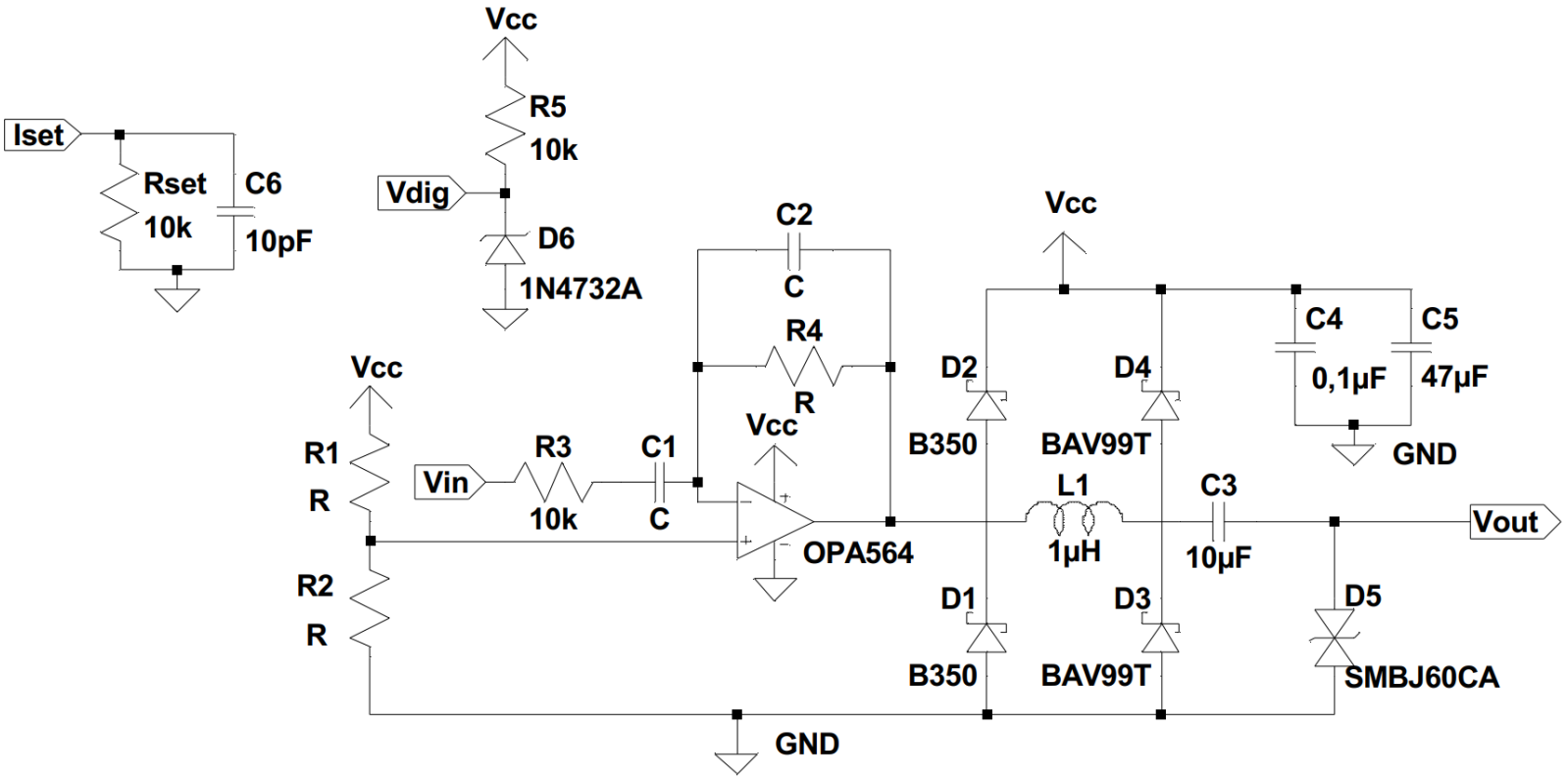
- 0x01: Trama Get System Info
- 0x80: Originada desde el host
- 0x04 00: Longitud de los datos
- 0x2A 81: CRC cabecera
- 0x00 00: CRC datos (si es 0, se ignora en recepción)

La segunda trama, es la contestación del módem a la petición de la configuración del sistema. Los cuatro primeros son también la parte de la cabecera de la trama, los cuatro siguientes los códigos CRC y los demás son los datos. En detalle son:

- 0x01: Trama Get System Info
- 0x00: Originada desde el módem
- 0x3c 00: Longitud de los datos (60 en decimal)
- 0x4C 36: CRC cabecera
- 0x00 00: CRC datos
- 0x05 01 00 04: Firmware versión (Major: 4, Minor: 0, Revision: 1, Build:5)
- 0x00 00: Longitud Serial Number (desactivado)
- 0x00 00 … 00: Serial Number (16 octetos con valor 0)
- 0x05: Tipo de dispositivo
- 0x03: Modo dispositivo (Punto a Punto)
- 0x00 00: Revisión hardware
- 0x00 00 00 00 00 00: Reservado
- 0x00: Puerto (SCI A)
- 0x01: PHY Mode (ROBO)
- 0x00 00 00 00 00 00 00 00: Reservado
- 0x01 00: Tamaño dirección
- 0x18 00: Offset dirección
- 0x87 00: Start tone
- 0x00: Dirección 1
- 0x00: Dirección 2
- 0x00: Dirección 3

ANEXO B: Diseño de una placa con el amplificador OPA564

Esquemático de la placa diseñada con el amplificador OPA564 que nos proporciona la misma salida que el módem para probar los transformadores y así evitar dañar los módems. Para realizar el PCB se uso la herramienta Eagle.



ANEXO C: Código desarrollado

C.1: Programa getsystem.c

```
/* Programa que envía un getsystem al módem, lee y parsea la respuesta
obteniendo los valores configurados */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>

// Constantes
#define BAUDRATE B57600
#define MODEMDEVICE "/dev/ttyUSB0"
#define _POSIX_SOURCE 1

// Funciones
void configuration_serialport(struct termios, int fd);

// Estructura trama get system info
struct struct_getsystem {
    unsigned char header [4];
    unsigned char flags_tx;
    unsigned char flags_rx;
    unsigned char modulation [2];
    unsigned char power_tx [2];
    unsigned char pattern [20];
    unsigned char gain_rx [10];
    unsigned char subbands [10];
};
```

```
int main()
{
int fd;
int i=0;
int bytes_leidos, bytes_escritos;
struct termios newtio;
struct struct_getsystem getsystem;
// Trama GET SYSTEM INFO (0x8A)
unsigned char buf_getsystem[4] = {0x8A, 0x80, 0x00, 0x00};

// Abrir puerto serie, en modo lectura escritura
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);

// Funcion de configuracion puerto serie
configuration_serialport(newtio, fd);

// Escritura trama get system info
bytes_escritos = write(fd,buf_getsystem,sizeof(buf_getsystem));

// Control de error de escritura
if (bytes_escritos != 4) {
   printf("[ERROR] NO se han escrito 4 Bytes por el puerto serie \n");
}

// Lectura resultado get system info
bytes_leidos = read(fd,&getsystem,sizeof(struct struct_getsystem));

// Control errores lectura
if (bytes_leidos != 50) {
   printf("[ERROR] NO se han recibido 50 Bytes\n");
}
```

```
// Procesado de la trama y salida por pantalla
printf("Tipo de trama: GET SYSTEM (%#04x)\n",getsystem.header[0]);
printf("Flags activados en tx %#04x\n",getsystem.flags_tx);
printf("FLAGS:\n");
if ((getsystem.flags_tx & 0x80) == 0x80) {
   printf("\tROBO TX\n");
} if ((getsystem.flags_tx & 0x01) == 0x01) {
   printf("\tFEC\n");
} if ((getsystem.flags_tx & 0x02) == 0x02) {
   printf("\tAGC ON\n");
} if ((getsystem.flags_rx & 0x01) == 0x01) {
   printf("\tROBO RX\n");
}
printf("Flags activados en rx: %#04x \n", getsystem.flags_rx);

switch (getsystem.modulation[0]) {
   case 0:
      printf("Modulacion: DBPSK\n");
      break;
   case 1:
      printf("Modulacion: DQPSK\n");
      break;
   case 3:
      printf("Modulacion: DBPSK + 1/4 rep\n");
      break;
   case 4:
      printf("Modulacion: DBPSK + 1/8 rep\n");
      break;
   default:
      printf("No se reconoce la modulacion configurada\n");
}
```

```
printf("Nivel potencia en transmision: %d \n", getsystem.power_tx[0]);
printf("Nivel AGC en recepcion: %d\n", getsystem.gain_rx[0]);
printf("AGC min: %d\n", getsystem.gain_rx[4]);
printf("AGC max: %d\n", getsystem.gain_rx[6]);
printf("Saltos de AGC: %d\n", getsystem.gain_rx[8]);

return 0;
}

// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
   bzero(&newtio, sizeof(newtio));
   newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las lineas de control del modem y activa receiver
   newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
   newtio.c_oflag = 0;
   newtio.c_lflag = 0; //forma NO canonica
   newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
   newtio.c_cc[VMIN] = 59; //Numero de caracteres mínimo que espera recibir
   tcflush(fd, TCIFLUSH);
   tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}
```

C.2: Programa getinfo.c

```
/* Programa que manda un getinfo al módem, lee y parsea la respuesta
obteniendo los valores configurados */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
```

```
#include <strings.h>

// Constantes
#define BAUDRATE B57600
#define MODEMDEVICE "/dev/ttyUSB0"
#define _POSIX_SOURCE 1

//Funciones
void configuration_serialport(struct termios, int fd);

// Estructura trama get info
struct struct_getinfo {
        unsigned char header [4];
        unsigned char crc_header [2];
        unsigned char crc_payload [2];
        unsigned char firmware [4];
        unsigned char serialNumber_length [2];
        unsigned char serialNumber [16];
        unsigned char device_type;
        unsigned char device_mode;
        unsigned char hw_rev [2];
        unsigned char reserved [4];
        unsigned char port;
        unsigned char phy_mode;
        unsigned char reserved_2 [8];
        unsigned char address_length [2];
        unsigned char address_offset [4];
        unsigned char band [2];
        unsigned char startTone [2];
        unsigned char mac_filter [2];
};
```

```
int main()
{
int fd;
int i=0;
int bytes_leidos, bytes_escritos;
struct termios newtio;
struct struct_getinfo getinfo;
// Trama GET SYSTEM
unsigned char buf_getinfo[8] = {0x01, 0x80, 0x04, 0x00, 0x2A, 0x81, 0x00, 0x00};

// Abrir puerto serie, en modo lectura escritura
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);

// Funcion de configuracion puerto serie
configuration_serialport(newtio, fd);

// Escritura trama get info
bytes_escritos = write(fd,buf_getinfo,sizeof(buf_getinfo));

// Control de error de escritura
if (bytes_escritos != 8) {
        printf("[ERROR] NO se han escrito 8 Bytes por el puerto serie \n");
}

// Lectura resultado get system
bytes_leidos = read(fd,&getinfo,sizeof(getinfo));

// Control errores lectura
if (bytes_leidos != 60) {
        printf("[ERROR] NO se han recibido 60 Bytes\n");
}
```

```
// Procesado de la trama y salida por pantalla
printf("Tipo de trama: GET SYSTEM INFO (%#04x)\n",getinfo.header[0]);

if ((getinfo.device_type == 0x05) && (getinfo.device_mode == 0x03)) {
        printf("Device type: PLC LITE\n");
        printf("Device Mode: P to P Mode\n");
} else {
        printf("No esta en modo Point to Point y no se esta utilizando el protocolo PLC Lite");
}
if (getinfo.port == 0) {
        printf("SCI Port: A\n");
} else {
        printf("SCI Port: B\n");
}
if (getinfo.phy_mode == 0) {
        printf("Modo ROBO: ON \n");
} else {
        printf("Modo ROBO: OFF \n");
}
if (getinfo.band[0] == 1){
        printf("Band: Half-Band\n");
} else {
        printf("Band: Full-Band\n");
}
printf("Start tone: %d \n", getinfo.startTone[0]);

return 0;
}

// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
        bzero(&newtio, sizeof(newtio));
```

```
        newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las
lineas de control del modem y activa receiver
        newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
        newtio.c_oflag = 0;
        newtio.c_lflag = 0; //forma NO canonica
        newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
        newtio.c_cc[VMIN] = 59; //Numero de caracteres mínimo que espera recibir
        tcflush(fd, TCIFLUSH);
        tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}
```

C.3: Programa set_tx.c

```
/* Programa que cambia los parametros de transmisión del módem */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>
#include <stdlib.h>

// Constantes
#define BAUDRATE B57600
#define MODEMDEVICE "/dev/ttyUSB0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

//Funciones
void configuration_serialport(struct termios, int fd);
void printuso(char *progname);
struct struct_set_tx initargs(int argc, char **argv, char *verb, int* robo, int* fec, int* ptx, int*
mod, int* band);
```

```
// Estructura trama set_tx
struct struct_set_tx {
   unsigned char header [4];
   unsigned char flags_tx [2];
   unsigned char modulation [2];
   unsigned char power_tx [2];
   unsigned char subbands [10];
};

//Variables globales estaticas para ancho de banda y ROBO
static int b = 0;
static int r = 1;

int main(int argc, char *argv[])
{
int fd;
int i=0;
int bytes_leidos, bytes_escritos;
struct termios newtio;
struct struct_set_tx set_tx_2;
unsigned char buf_lectura[10]; //Leer el ACK de los cambios

//Array trama para cambiar ancho de banda (byte 13)
unsigned char buf_band [36] = {0x0c, 0x80, 0x1f, 0x00, 0x25, 0x67, 0x68, 0x3b, 0x09, 0x00,
0x04, 0x00, 0x01, 0x00, 0x87, 0x00, 0x0b, 0x00, 0x02, 0x00, 0x01, 0x00, 0x0a, 0x00, 0x09,
0x00, 0x01, 0x00, 0x18, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,0x00};
char verb;
int robo;
int fec;
int ptx;
int mod;
int band;
```

```
//Inicializacion del buffer de lectura
for (i=0; i<sizeof(buf_lectura); i++) {
   buf_lectura[i] = 0x00;
}

// Abrir puerto serie, en modo lectura escritura
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);

// Funcion de configuracion puerto serie
configuration_serialport(newtio, fd);

//Estructura que guarda los cambios de parametros introducido por linea de comandos
set_tx_2 = initargs(argc, argv, &verb, &robo, &fec, &ptx, &mod, &band);

//Configuración del ancho de banda
if (b == 1){
   buf_band [12] = 0x01;
} else if (b == 0){
   buf_band [12] = 0x00;
}
//Configuracion ROBO o no ROBO a nivel global (TX y RX)
if (r == 1){
   buf_band [20] = 0x01;
} else if (r == 0){
   buf_band [20] = 0x00;
}

// Escritura y lectura del cambio de ancho de banda
bytes_escritos= write(fd,buf_band,sizeof(buf_band));
bytes_leidos = read(fd,buf_lectura,sizeof(buf_lectura));
// Escritura y lectura de los demas parametros de configuracion
```

```
bytes_escritos= write(fd,&set_tx_2,sizeof(struct struct_set_tx));
bytes_leidos=read(fd,buf_lectura,sizeof(buf_lectura));


return 0;
}


// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
   bzero(&newtio, sizeof(newtio));
   newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las lineas de control del modem y activa receiver
   newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
   newtio.c_oflag = 0;
   newtio.c_lflag = 0; //forma NO canonica
   newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
   newtio.c_cc[VMIN] = 6; //Numero de caracteres mínimo que espera recibir
   tcflush(fd, TCIFLUSH);
   tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}


// Funcion que indica el uso del programa
void printuso(char *progname) {
   fprintf(stderr,"Uso: %s [-v] -r[ROBO] [-f[FEC]] [-p[Ptx]] [-m[modulation] -b[band]]\n",progname);
   fprintf(stderr, "Si algun parametro no se introduce, se pondra el valor por defecto");
   fprintf(stderr,"  -v\t\tMuestra detalles en salida estándar\n");
   fprintf(stderr,"  -r[ROBO]\t modo ROBO ON (1), modo ROBO OFF (0) (por defecto: 1)\n");
   fprintf(stderr,"  -f[FEC]\t FEC ON (1), FEC OFF (0) (por defecto: 1)\n");
   fprintf(stderr,"  -p[Ptx]\t Nivel de potencia de TX (de 0 a 7), siendo 0 el MAX (por defecto: 2)\n");
   fprintf(stderr,"  -m[modulation]\tModulación a utilizar: (por defecto DBPSK) \n");
   fprintf(stderr,"      1\t\tDBPSK\n");
   fprintf(stderr,"      2\t\tQPSK\n");
```

```
    fprintf(stderr,"      3\t\tDBPSK+1/4\n");
    fprintf(stderr,"      4\t\tDBPSK+1/8\n");
    fprintf(stderr,"  -b[band]\tHalf-Band (1) y Full-Band (0) (por defecto: 1)\n");
}

// Funcion lectura parametros de configuracion por linea de comandos
struct struct_set_tx initargs(int argc, char **argv, char *verb, int* robo, int* fec, int* ptx, int*
mod, int* band) {
    char *progname = *argv;
    struct struct_set_tx set_tx;
    int j = 0;

    //Inicializacion struct con valores por defecto
    set_tx.header [0] = 0x8b; //SET PHY PARAMETERS
    set_tx.header [1] = 0x80;
    set_tx.header [2] = 0x10;
    set_tx.header [3] = 0x00;
    set_tx.flags_tx [0] = 0x81; //ROBO TX y FEC
    set_tx.flags_tx [1] = 0x02; //PRM
    set_tx.modulation [0] = 0x00; //BPSK
    set_tx.modulation [1] = 0x00;
    set_tx.power_tx [0] = 0x02; //Ptx=2 (6dB)
    set_tx.power_tx [1] = 0x00;
    for (j=0;j<10;j++){
        set_tx.subbands[j]=0x00;
    }

  if (argc<2) {
        printuso(progname);
        exit(1);
  }
    //Lectura y procesado de los parametros por linea de configuracion
```

```
for(argc--,argv++; argc > 0; argc--,argv++) {
  if (**argv == '-') {
     switch (*(++*argv)) {
        case 'v':
           *verb=1;
           break;
        case 'r':
           *robo=atoi(++*argv);
           printf("ROBO es:%d\n", (int)*robo);
           if (*robo == 1){
              set_tx.flags_tx [0] = 0x81;
              r = 1;
           } else {
              set_tx.flags_tx [0] = 0x00;
              r = 0;
           }
           printf("Flags 0 ROBO tx es %x\n",set_tx.flags_tx [0]);
           break;

        case 'f':
           *fec=atoi(++*argv);
           if (*fec == 1){
              set_tx.flags_tx [0] = 0x81;
           } else {
              set_tx.flags_tx [0] = 0x00;
           }
           printf("Flags 0 FEC tx es %x\n",set_tx.flags_tx [0]);
           break;

        case 'p':
           *ptx=atoi(++*argv);
           printf("El flag de ptx enviado es:%d\n",*ptx);
```

```
        set_tx.power_tx [0] = (unsigned char)*ptx;
        printf("Flags 0 y 1 Ptx tx es %x\n%x\n",set_tx.power_tx [0], set_tx.power_tx[1]);
        break;

    case 'm':
        *mod=atoi(++*argv);
        if (*mod == 1){
            set_tx.modulation [0] = 0x00;
        } else if (*mod == 2){
                set_tx.modulation [0] = 0x01;
            } else if (*mod == 3){
                set_tx.modulation [0] = 0x03;
            } else if (*mod == 4){
                set_tx.modulation [0] = 0x04;
            }
        printf("Flags 0 modulation tx es %x\n",set_tx.modulation [0]);;
        break;

    case 'b':
        *band=atoi(++*argv);
        if (*band == 1){
            b = 1;
        } else {
            b = 0;
        }
        break;

    default:
        printuso(progname);
        exit(1);
    }
}
```

```
        else {
            printuso(progname);
            exit(1);
        }
    }

    if (*verb) {
        fprintf(stderr,"Valores de parámetros: r=%d, f=%d, p=%d, m=%d,
b=%d\n",*robo,*fec,*ptx,*mod,*band);
    }
    return set_tx;
}
```

C.4: Programa set_rx.c

```
/* Programa que cambia los parametros de recepción del módem */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>
#include <stdlib.h>

// Constantes
#define BAUDRATE B57600
#define MODEMDEVICE "/dev/ttyUSB0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

//Funciones
void configuration_serialport(struct termios, int fd);
void printuso(char *progname);
```

```
struct struct_set_rx initargs(int argc, char **argv, char *verb, int* robo, int* agc, int* grx);

// Estructura trama set_rx
struct struct_set_rx {
  unsigned char header [4];
  unsigned char flags_rx [2];
  unsigned char gain_rx [2];
  unsigned char subbands [10];
};

int main(int argc, char *argv[])
{
int fd;
int i=0;
int bytes_leidos, bytes_escritos;
struct termios newtio;
struct struct_set_rx set_rx_2;
unsigned char buf_lectura[10]; //Leer el ACK de los cambios
char verb;
int robo;
int agc;
int grx;

//Inicializacion del buffer de lectura
for (i=0; i<sizeof(buf_lectura); i++) {
  buf_lectura[i] = 0x00;
}

// Abrir puerto serie, en modo lectura escritura
fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY);

// Funcion de configuracion puerto serie
```

```
configuration_serialport(newtio, fd);

//Estructura que guarda los cambios de parametros introducido por linea de comandos
set_rx_2 = initargs(argc, argv, &verb, &robo, &agc, &grx);

//Escritura y lectura parametros de configuracion recepcion
bytes_escritos= write(fd,&set_rx_2,sizeof(struct struct_set_rx));
printf("Los bytes escritos por el puerto serie son %d \n", bytes_escritos);
bytes_leidos=read(fd,buf_lectura,sizeof(buf_lectura));

return 0;
}

// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
  bzero(&newtio, sizeof(newtio));
  newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las lineas de control del modem y activa receiver
  newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
  newtio.c_oflag = 0;
  newtio.c_lflag = 0; //forma NO canonica
  newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
  newtio.c_cc[VMIN] = 6; //Numero de caracteres mínimo que espera recibir
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}

// Funcion que indica el uso del programa
void printuso(char *progname) {
  fprintf(stderr,"Uso: %s [-v] -r[ROBO] [-a[AGC]] [-g[Grx]]\n",progname);
  fprintf(stderr, "Si algun parametro no se introduce, se pondra el valor por defecto");
  fprintf(stderr,"  -v\t\tMuestra detalles en salida estándar\n");
```

```
  fprintf(stderr,"  -r[ROBO]\t modo ROBO ON (1), modo ROBO OFF (0) (por defecto: 1)\n");
  fprintf(stderr,"  -a[AGC]\t AGC ON (1), AGC OFF (0) (por defecto: 1)\n");
  fprintf(stderr,"  -g[Grx]\tganancia amplificador recepcion (por defecto: AGC ON)\n");
}

// Funcion lectura parametros de configuracion por linea de comandos
struct struct_set_rx initargs(int argc, char **argv, char *verb, int* robo, int* agc, int* grx) {
    char *progname = *argv;
    struct struct_set_rx set_rx;
    int j = 0;

  //Inicializacion struct con valores por defecto
  set_rx.header [0] = 0x8c;//SET PHY PARAMETERS
  set_rx.header [1] = 0x80;
  set_rx.header [2] = 0x0e;
  set_rx.header [3] = 0x00;
  set_rx.flags_rx [0] = 0x02; //ROBO RX y AGC
  set_rx.flags_rx [1] = 0x01;
  set_rx.gain_rx [0] = 0x00; //Prx
  set_rx.gain_rx [1] = 0x00;
  for (j=0;j<10;j++){
    set_rx.subbands[j]=0x00;
  }

  if (argc<2) {
    printuso(progname);
    exit(1);
    }
  for(argc--,argv++; argc > 0; argc--,argv++) {
    if (**argv == '-') {
      switch (*(++*argv)) {
        case 'v':
```

```
  *verb=1;
  break;

case 'r':
  *robo=atoi(++*argv);
  if (*robo == 0){
    set_rx.flags_rx [1] = 0x00;
  } else{
    set_rx.flags_rx [1] = 0x01;
  }
  printf("Flag ROBO es %x\n",set_rx.flags_rx [1]);
  break;

case 'a':
  *agc=atoi(++*argv);
  if (*agc == 0) {
    set_rx.flags_rx [0] = set_rx.flags_rx [0] & (~0x02);
    printf("Flag agc 0 es %#04x\n",set_rx.flags_rx[0]);
  } else {
    set_rx.flags_rx [0] = set_rx.flags_rx [0] | 0x02;
    printf("Flag agc 1 es %#04x\n",set_rx.flags_rx[0]);
  }
  printf("Flag agc despues es %x\n",set_rx.flags_rx[0]);
  break;

case 'g':
  *grx=atoi(++*argv);
  if (*agc == 1){
    set_rx.gain_rx [0] = 0x00;
  } else {
    set_rx.gain_rx [0] = (unsigned char) *grx;
  }
```

```
        printf("Flag gain_rx es %x\n",set_rx.gain_rx[0]);
        break;

      default:
      printuso(progname);
        exit(1);
      }
    }
    else {
    printuso(progname);
      exit(1);
    }
  }

 if (*verb) {
   fprintf(stderr,"Valores de parámetros: r=%d, a=%d, g=%d\n",*robo,*agc,*grx);
 }
 return set_rx;
}
```

C.5: Programa send_bin.c

```
/* Programa que lee un fichero con los datos y los empaqueta en tramas para mandarlos */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <strings.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <stdlib.h>
```

```
// Constantes
#define LONGITUD_MAXIMA 54 //Longitud maxima de datos en una trama
#define BAUDRATE B57600
#define MODEMDEVICE0 "/dev/ttyUSB0"
#define MODEMDEVICE1 "/dev/ttyUSB1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

//Funciones
void CRC16_UpdateChecksum(unsigned short *pcrcvalue, const void *data, int length);
unsigned short CRC16_BlockChecksum(const void *data, int length);
double timeval_diff(struct timeval *a, struct timeval *b);
void configuration_serialport(struct termios, int fd);

// Tabla para calcular CRC
static unsigned short crctable[256] = {
 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
 0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
 0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
 0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
 0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
 0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
 0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
 0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
 0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
 0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
 0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
 0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
 0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
 0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
 0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
 0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
```

```
 0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
 0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
 0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
 0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
 0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
 0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
 0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
 0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
 0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
 0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
 0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
 0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
 0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
 0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
 0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

//Estructura trama de datos
struct frame {
    unsigned char header [4];
    unsigned char CRCheader [2];
    unsigned char CRCdata [2];
    unsigned char mode [2];
    unsigned char dataFixed [26];
    unsigned char data [LONGITUD_MAXIMA];
    unsigned char padding;
};

void main(int argc, char *argv[]){
    struct frame dataFrame;
    struct termios newtio;
```

```
   struct timeval horainicio; // variable para estadísticas finales
   struct timeval tvalBefore, tvalAfter;
   double secs;
   unsigned long ttrans; // tiempo de transmisión a simular
   unsigned long timeout; // tiempo de expiración a simular
   FILE *ptr_fich_tx;
   char nombre_fichero[] = "ca_10.png";
   unsigned char c;
   int fd0, fd1;
   int i,j;
   unsigned char buffer[LONGITUD_MAXIMA + 1];
   int bytes_leidos, bytes_escritos, respuesta;
   int tamchar,tamuchar;
   int bytes_frame;
   int sizeBin;
   int numPages;
   unsigned short crcCab, crcPay;
   unsigned char buf_lectura[100] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
   unsigned char buf_respuesta[100] =
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};


  if (argc != 2){
      printf("Uso: %s nombre_fichero \n", argv[0]);
      exit(1);
   }
```

```
  // Abrir puerto serie, en modo lectura escritura
  if ((fd0 = open(MODEMDEVICE0, O_RDWR | O_NOCTTY)) < 0){
     printf("Ha habido un error al abrir el puerto serie");
  }

// Funcion de configuracion puerto serie
configuration_serialport(newtio, fd0);

  //Tipo de trama -> De datos cabecera fija
  dataFrame.header [0] = 0x00;
  dataFrame.header [1] = 0x80;
  dataFrame.header [3] = 0x00;

  //CRC Payload a 00
  dataFrame.CRCdata [0] = 0x00;
  dataFrame.CRCdata [1] = 0x00;

  //Mode
  dataFrame.mode [0] = 0x01;
  dataFrame.mode [1] = 0x00;

//Datos fijos de la trama
  dataFrame.dataFixed[0]= 0xa0;
  dataFrame.dataFixed[1]= 0xaa;
  dataFrame.dataFixed[12]= 0x01;
  dataFrame.dataFixed[16]= 0x01;

  //Padding
  dataFrame.padding = 0x00;

//Instante de tiempo antes de empezar la transmision
```

```
   gettimeofday (&tvalBefore, NULL);


  //Apertura fichero en modo lectura y control del error
   if ((ptr_fich_tx = fopen(argv[1],"r")) == NULL){
      printf("No se ha podido abrir el fichero %s.\n", argv[1]);
   }
   else {
      clock_t start = clock();


      gettimeofday (&tvalBefore, NULL); //medimos justo antes de empezar a enviar los datos


      //Calculo tamaño del fichero y cuantas páginas serán necesarias en función de la longitud
máxima
      fseek(ptr_fich_tx, 0L, SEEK_END);
      sizeBin = ftell (ptr_fich_tx);
   printf("El archivo %s ocupa: %d\n", argv[1],sizeBin);
   numPages = ((sizeBin / LONGITUD_MAXIMA) + 1);
   printf("Numero de tramas necesarias para enviar fichero:%d\n",numPages);
      dataFrame.dataFixed[16] = numPages;
      printf("El numero de paginas es %02hhx\n",dataFrame.dataFixed[16]);
   fseek(ptr_fich_tx, 0L, SEEK_SET);


      while(!feof(ptr_fich_tx)){
         bytes_leidos = fread(dataFrame.data,sizeof(unsigned char),
LONGITUD_MAXIMA,ptr_fich_tx);


         dataFrame.header [2] = (unsigned char) bytes_leidos + 32; //sumo los datos reales mas
los datos fijos que son 32 (4 CRCs + 2 MODE + 26 fijos)


         //Calculo CRCs cabecera y payload
         crcCab = CRC16_BlockChecksum (dataFrame.header, sizeof(dataFrame.header));
         dataFrame.CRCheader[0] = crcCab;
         dataFrame.CRCheader[1] = crcCab >> 8;
```

```
dataFrame.dataFixed [20] = (unsigned char) bytes_leidos;

if ((argv[1] == "-") && (argv[2] == "v")){
   printf("Longitud trama enviada %d \n", dataFrame.header[2]);
   printf("Longitud datos payload %d\n", (unsigned char)dataFrame.dataFixed[20]);
}

if ((LONGITUD_MAXIMA % 2)!=0){
   bytes_frame=sizeof(struct frame);
} else{
   bytes_frame=sizeof(struct frame) - 1;
}

bytes_escritos = write(fd0,&dataFrame,bytes_frame);

printf("Bytes escritos:%d\n",bytes_escritos);

if ((argv[1] == "-") && (argv[2] == "v")){
   printf("Los bytes escritos en el puerto serie son %d", bytes_escritos);
}
usleep (165000); //necessary for BPSK
//usleep (165000*2.3); //necessary for BPSK+1/4
//usleep (165000*4.2); //necessary for BPSK+1/8
//usleep (100000);

j++;
printf("Numero de pagina enviada:%02hhx\n",dataFrame.dataFixed[12]);

//Controlar el numero de secuencia (cabecera)
dataFrame.header[1] = ((dataFrame.header[1] + 1) & 0x8f);
dataFrame.dataFixed[12] = dataFrame.dataFixed [12] + 1;
```

```
            printf("Numero de secuencia enviada: %02hhx\n",dataFrame.header[1]);

        }

        gettimeofday (&tvalAfter, NULL); //Anotamos hora de fin de la transmision
        secs = timeval_diff(&tvalAfter, &tvalBefore);
        printf("Tiempo: %.7g ms\n", secs*1000);
        printf ("Velocidad TX efectiva conseguida: %.6g bps\n",(sizeBin*8/secs));
    }
}

//Definicion de funciones completas
void CRC16_UpdateChecksum(unsigned short *pcrcvalue, const void *data, int length)
{
 unsigned short crc;
 const unsigned char *buf = (const unsigned char *) data;
 crc = *pcrcvalue;
 while (length--)
 {
 crc = (crc << 8) ^ crctable[(crc >> 8) ^ *buf++];
 }
 *pcrcvalue = crc;
}

unsigned short CRC16_BlockChecksum(const void *data, int length)
{
 unsigned short crc;
 crc = 0;
 CRC16_UpdateChecksum(&crc, data, length);
 return crc;
}
```

```
double timeval_diff(struct timeval *a, struct timeval *b)
{
  return
    (double)(a->tv_sec - (double)b->tv_sec);
}

// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las lineas de control del modem y activa receiver
    newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
    newtio.c_oflag = 0;
    newtio.c_lflag = 0; //forma NO canonica
    newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
    newtio.c_cc[VMIN] = 20; //Numero de caracteres mínimo que espera recibir
    tcflush(fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}
```

C.6: Programa receive_bin.c

```
/* Programa que recibe y procesa los datos recibidos */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <strings.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
```

```
#include <stdlib.h>
#include <signal.h>

// Constantes
#define LONGITUD_MAXIMA 54
#define BAUDRATE B57600
#define MODEMDEVICE1 "/dev/ttyUSB1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

//Funciones
void configuration_serialport(struct termios, int fd);
void CRC16_UpdateChecksum(unsigned short *pcrcvalue, const void *data, int length);
unsigned short CRC16_BlockChecksum(const void *data, int length);
double timeval_diff(struct timeval *a, struct timeval *b);
void terminarbucle();

typedef enum { false, true } bool;
bool fin = false;
// Tabla para calcular CRC
static unsigned short crctable[256] = {
 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
 0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
 0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
 0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
 0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
 0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
 0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
 0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
 0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
 0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
 0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
 0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
```

```
 0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
 0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
 0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
 0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
 0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
 0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
 0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
 0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
 0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
 0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
 0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
 0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
 0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
 0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
 0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
 0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
 0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
 0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
 0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

//Estructura trama de datos
struct frame {
    unsigned char header [4];
    unsigned char CRCheader [2];
    unsigned char CRCdata [2];
    unsigned char mode [2];
    unsigned char dataFixed [26];
    unsigned char data [LONGITUD_MAXIMA];
    unsigned char padding;
};
```

```
void main(int argc, char *argv[]){
    struct frame dataFrame;
    struct termios newtio;
    struct timeval horainicio; // variable para estadísticas finales
    struct timeval tvalBefore, tvalAfter,tvalCond;//medir tiempo de inicio y fin
    double secs, tiempo_secs;
    clock_t t_ini, t_fin;
  unsigned long ttrans; // tiempo de transmisión a simular
    unsigned long timeout; // tiempo de expiración a simular
    unsigned short padding;
    FILE *ptr_fich_rx, *ptr_fich_ruido;
    char nombre_fichero_ruido[] = "test.bin";
    unsigned char c;
    int fd1;
    int i,k;
    int num_rx = 1;
    int n=0;
    unsigned char buf_ruido[LONGITUD_MAXIMA],buf_zero[LONGITUD_MAXIMA];
    int bytes_leidos, bytes_escritos, bytes_datos,bytes_ruido;
    int sizeBin;

    unsigned short crcCab, crcPay;

  if (argc != 2){
        printf("Uso: %s nombre_fichero \n", argv[0]);
        exit(1);
    }

  //Relleno con 0's
  for (k=0; k<sizeof(buf_ruido); k++){
        buf_ruido [k] = 0x00;
```

```
        buf_zero [k] = 0x00;
    }
    signal(SIGALRM,terminarbucle);

    // Abrir puerto serie, en modo lectura escritura
    fd1 = open(MODEMDEVICE1, O_RDWR | O_NOCTTY);

  // Funcion de configuracion puerto serie
  configuration_serialport(newtio, fd1);

  //Instante de tiempo antes de empezar la transmision
    gettimeofday (&tvalBefore, NULL);
    //Ruido blanco
    if ((ptr_fich_ruido = fopen(nombre_fichero_ruido,"r"))==NULL){
        printf("No se ha podido abrir el fichero %s.\n",nombre_fichero_ruido);
    } else {
        bytes_ruido = fread(buf_ruido,sizeof(unsigned char),
LONGITUD_MAXIMA,ptr_fich_ruido);
    }

  //Apertura fichero en modo escritura y control del error
    if ((ptr_fich_rx = fopen(argv[1],"w"))==NULL){
            printf("No se ha podido abrir el fichero %s.\n",argv[1]);
    }
    while (fin==false){

        bytes_leidos = read(fd1,&dataFrame,sizeof(struct frame));
        if (dataFrame.dataFixed[12] == 0x01) {
            gettimeofday (&tvalCond, NULL);

        }
        bytes_datos = bytes_leidos-36;
```

```
        printf("Numero de pagina recibida:%d\n",dataFrame.dataFixed[12]);
        printf("Size payload:%d\n",dataFrame.dataFixed[20]);
        padding=LONGITUD_MAXIMA-dataFrame.dataFixed[20];
        printf("Long padding:%d\n",padding);
        if (dataFrame.dataFixed[20]<LONGITUD_MAXIMA){
           memcpy(dataFrame.data+dataFrame.dataFixed[20], buf_zero, padding);
        }

    //Escritura en el fichero de los datos recibidos
    bytes_escritos = fwrite(dataFrame.data,sizeof(unsigned char),(unsigned
char)LONGITUD_MAXIMA,ptr_fich_rx);

        fclose(ptr_fich_rx);
        ptr_fich_rx = fopen(argv[1],"a");

        num_rx++;
        if (num_rx == 0x70){
           fin = true;
        }

    }

        sizeBin = ftell (ptr_fich_rx);
        printf("El archivo %s ocupa: %d\n", argv[1],sizeBin);
        gettimeofday (&tvalAfter, NULL); //Anotamos hora de fin de la transmision
        secs = timeval_diff(&tvalAfter, &tvalBefore);
        tiempo_secs = timeval_diff(&tvalAfter, &tvalCond);
        printf("Tiempo: %.7g ms\n", secs * 1000.0);
        printf("Tiempo real: %.7g ms\n", tiempo_secs * 1000.0);
        printf ("Velocidad TX efectiva conseguida: %.6g bps\n",(sizeBin*8/secs));
        fclose(ptr_fich_rx);
```

```
}

//Definicion de funciones completas
void CRC16_UpdateChecksum(unsigned short *pcrcvalue, const void *data, int length)
{
 unsigned short crc;
 const unsigned char *buf = (const unsigned char *) data;
 crc = *pcrcvalue;
 while (length--)
 {
 crc = (crc << 8) ^ crctable[(crc >> 8) ^ *buf++];
 }
 *pcrcvalue = crc;
}

unsigned short CRC16_BlockChecksum(const void *data, int length)
{
 unsigned short crc;
 crc = 0;
 CRC16_UpdateChecksum(&crc, data, length);
 return crc;
}
double timeval_diff(struct timeval *a, struct timeval *b)
{
  return
    (double)(a->tv_sec + (double)a->tv_usec/1000000) -
    (double)(b->tv_sec + (double)b->tv_usec/1000000);
}

void terminarbucle (){
    signal(SIGALRM,SIG_IGN);
    fin = true;
```

```
    signal(SIGALRM, terminarbucle);
}


// Funcion configuracion puerto serie
void configuration_serialport(struct termios newtio, int fd) {
    bzero(&newtio, sizeof(newtio));
    newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD; //57600, 8N1, ignora las lineas de control del modem y activa receiver
    newtio.c_iflag = IGNPAR | IGNCR; //Sin paridad y traduce CR en nueva linea
    newtio.c_oflag = 0;
    newtio.c_lflag = 0; //forma NO canonica
    newtio.c_cc[VTIME] = 1; //Tiempo en ms que se queda bloqueada la lectura
    newtio.c_cc[VMIN] = LONGITUD_MAXIMA + 36; //Numero de caracteres mínimo que espera recibir
    tcflush(fd, TCIFLUSH);
    tcsetattr(fd,TCSANOW,&newtio); //Carga el struct termios como nueva configuracion
}
```

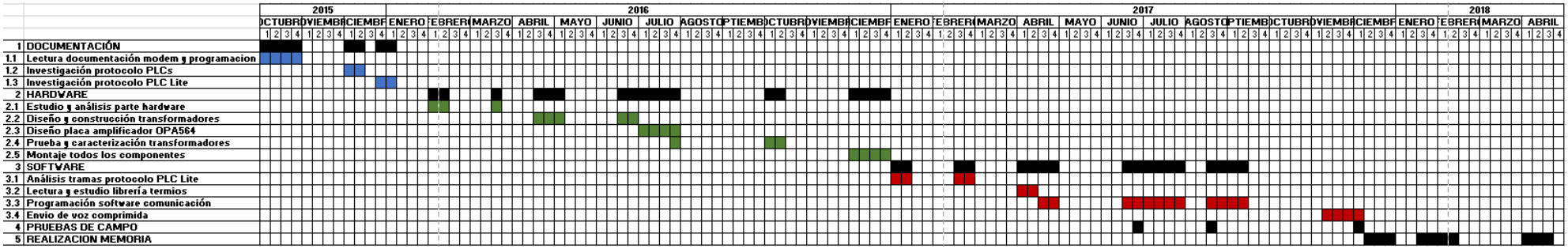
ANEXO D: Presupuesto de montaje de dos prototipos de radio PLC

Presupuesto realizado para un prototipo teniendo en cuenta el precio por unidad de compra:

Concepto	**Precio**
Núcleo transformador EFD15-N87	1,05 €
Carrete plástico EFD-15	1,28 €
bobina hilo de cobre AWG36	17,53 €
Módem PLC TI f28plc83	72,19 €
Convertidor 15V(TRACO-POWER TEN30-1213)	65,63 €
Convertidor 3,3V(TRACO-POWER TSR 1-2433)	7,27 €
TOTAL	**164,95 €**

El precio total de dos prototipos de radio PLC sería 329,9 €.

ANEXO E: Diagrama de Gantt



La estimación en horas para cada uno de los bloques es:

BLOQUE	TIEMPO (Horas)
1. DOCUMENTACION	32
2. HARDWARE	108
3. SOFTWARE	116
4. PRUEBAS DE CAMPO	24
5. RELIZACION MEMORIA	60
TOTAL	340