Índice

1. Códigos principales del sistema

En este apartado se exponen los programas de ambas unidades.

1.1. Código del nodo central

1.1.1. Inclusiones de subrutinas

```
#include <p18f4520.h>
#include "ConfigurationBits.h" //Configuration bits
#include "LCD1602.h"
#include "DS18B20.h"
#include "1wire.h"
#include "UPC_nRF24L01.h"
#include <stdio.h>
#include <string.h>
#include <delays.h>
#include <timers.h>
#define s0 PORTBbits.RB0
#define s1 PORTBbits.RB1
```

1.1.2. Inicialización de variables

```
unsigned char direccionenvio[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
unsigned char direccion1[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
unsigned char direccion2[5]={0x11, 0x11, 0x11, 0x11, 0x11};
unsigned char direccion3[5]={0x41, 0x41, 0x41, 0x41, 0x41};
unsigned char direccion4[5]={0x71, 0x71, 0x71, 0x71, 0x71};
unsigned char direccion5[5]={0xD1, 0xD1, 0xD1, 0xD1, 0xD1};
unsigned char direccion6[5]={0xF1, 0xF1, 0xF1, 0xF1, 0xF1};
unsigned char direccion7[5]={0x21, 0x21, 0x21, 0x21, 0x21};
unsigned char direccion8[5]={0x81, 0x81, 0x81, 0x81, 0x81};
unsigned char direccion9[5]={0x91, 0x91, 0x91, 0x91, 0x91};
unsigned char direccion10[5]={0xA1, 0xA1, 0xA1, 0xA1, 0xA1};

unsigned char informe;
unsigned char informe2;

unsigned int j=0;
unsigned int s=1;
unsigned int c=0;
unsigned int w=0;

unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
unsigned char T[2];
char buffer[12];
unsigned char cambio=0;
```

```
unsigned char encendido[10]={0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};
unsigned char activacion[1]={0b10000000};
unsigned char act[10]={0b10000000,0b01000000,0b00100000,0b00010000,
    0b00001000,0b00000100,0b00000010,0b00000001,0b11000000,0b10100000};

unsigned char s00;
unsigned char s11;


struct temperatura
{
    char T[2];
    char TM[2];
    char Tm[2];
};

struct temperatura sensor[10];
```

1.1.3. Programa principal

```
void InterruptHandlerHigh();
void main()
{

for(w=0;w<10;w=w+1)
    {sensor[w].Tm[0]=0xFF;
    sensor[w].Tm[1]=0xFF;}

for(w=0;w<10;w=w+1)
    {sensor[w].TM[0]=0x00;
    sensor[w].TM[1]=0x00;}

ADCON0bits.ADON=0;  // Disable the Analog-to-Digital (A/D) Converter module
TRISCbits.TRISC4 = 0; // RS_PIN for LCD
TRISCbits.TRISC5 = 0; // RW_PIN for LCD
TRISCbits.TRISC6 = 0; // E_PIN for LCD
TRISB=0xFF; // botones
InitLCD();
TRISD=0xFF;
OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_2);
INTCON2bits.TMR0IP = 1; // High priority for Timer0
RCONbits.IPEN = 1; // Enable priority levels
INTCONbits.GIEH = 1; // Enable interrupts
```

```
while(1)
{

if (cambio==1)
    {
    strcpy(sensor[c].T,T);


    if ( ((T[0]&0x80)!=0) && ((sensor[c].Tm[0]&0x80)!=0))
        {if (strcmp(T,sensor[c].Tm)>0)
            strcpy(sensor[c].Tm,T);}
    if ( ((T[0]&0x80)!=0) && ((sensor[c].Tm[0]&0x80)==0))
        {strcpy(sensor[c].Tm,T);}
    if ( ((T[0]&0x80)==0) && ((sensor[c].Tm[0]&0x80)==0))
        {if (strcmp(T,sensor[c].Tm)<0)
            strcpy(sensor[c].Tm,T);}

    if ( ((T[0]&0x80)!=0) && ((sensor[c].TM[0]&0x80)!=0))
        {if (strcmp(T,sensor[c].TM)<0)
            strcpy(sensor[c].TM,T);}
    if ( ((T[0]&0x80)==0) && ((sensor[c].TM[0]&0x80)!=0))
        {strcpy(sensor[c].TM,T);}
    if ( ((T[0]&0x80)==0) && ((sensor[c].TM[0]&0x80)==0))
        {if (strcmp(T,sensor[c].TM)>0)
            strcpy(sensor[c].TM,T);}


    if (strcmp(direccionenvio,direccion10)==0)
        {strcpy(direccionenvio,direccion1);}
    else if (strcmp(direccionenvio,direccion1)==0)
        {strcpy(direccionenvio,direccion2);}
    else if (strcmp(direccionenvio,direccion2)==0)
        {strcpy(direccionenvio,direccion3);}
    else if (strcmp(direccionenvio,direccion3)==0)
        {strcpy(direccionenvio,direccion4);}
    else if (strcmp(direccionenvio,direccion4)==0)
        {strcpy(direccionenvio,direccion5);}
    else if (strcmp(direccionenvio,direccion5)==0)
        {strcpy(direccionenvio,direccion6);}
    else if (strcmp(direccionenvio,direccion6)==0)
        {strcpy(direccionenvio,direccion7);}
    else if (strcmp(direccionenvio,direccion7)==0)
        {strcpy(direccionenvio,direccion8);}
    else if (strcmp(direccionenvio,direccion8)==0)
        {strcpy(direccionenvio,direccion9);}
    else
        {strcpy(direccionenvio,direccion10);}
```

```
ClearLCD();
gotoxy(0,0);
while(BusyXLCD()); // Wait while LCD is busy
putrsXLCD("Sensor "); // Send a new character to the LCD// Slave present on the bus
while(BusyXLCD()); // Wait while LCD is busy
putcXLCD(j+48); // Send a new character to the LCD// Slave present on the bus
gotoxy(1,0);
if (encendido[j]==0x01)
    {if(s==1)
        {DS18B20_decode_temp(sensor[j].T,buffer);
        while(BusyXLCD());
        putrsXLCD("T= ");
        while(BusyXLCD());
        putsXLCD(sensor[j].T);}

    if(s==2)
        {DS18B20_decode_temp(sensor[j].TM,buffer);
        while(BusyXLCD());
        putrsXLCD("TM= ");
        while(BusyXLCD());
        putsXLCD(sensor[j].TM);}

    if(s==3)
        {DS18B20_decode_temp(sensor[j].Tm,buffer);
        while(BusyXLCD());
        putrsXLCD("Tm= ");
        while(BusyXLCD());
        putsXLCD(sensor[j].Tm);}
    }

else if (encendido[j]==0x00)
    {while(BusyXLCD());
    putrsXLCD("Apagado");}

cambio=0;
}
```

1.1.4. Interrupciones

```
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh()
{
    if(INTCONbits.TMR0IF) // Has Timer0 overflow causing an interrupt?
    {
            c=c+1;
            if (c==10)
                {c=0;}

            LATAbits.LATA5=0;
            TRISD=0xFF; // RF

            activacion[0]=act[c];

            SPI_Start(0b10);
            nRF24L01_Ports_Start();
            Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10, 1, 1);
            Send_Data_TX_Mode_nRF24L01(1, 0b11, direccionenvio, 1, activacion);
            informe2=Check_Data_Sent_TX_Mode_nRF24L01();
            Finish_nRF24L01_Operation();
            Finish_SPI_Operation();


            SPI_Start(0b10);
            nRF24L01_Ports_Start();
            Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion1,
                direccion2, 0x00, 0x00, 0x00, 0x00, 1, 2, 2, 0, 0, 0, 0);
            informe=Receive_Data_RX_Mode_nRF24L01(1, 1, 2, T);
            Finish_nRF24L01_Operation();
            Finish_SPI_Operation();

            if (s00=1 && s0==0)
                {j=j+1;
                if(j==10)
                    {j=0;}
                }
            if (s11=1 && s1==0)
                {s=s+1;
                if(s==4)
                    {s=1;}
                }

            s11=s1;
            s00=s0;

            if (informe2==0b00000001)
                {encendido[c]=0b00000001;}

            else
                {encendido[c]=0b00000000;}
```

```
            if (c==9)
                {LATAbits.LATA5=1;

                TRISCbits.TRISC4 = 0; // RS_PIN for LCD
                TRISCbits.TRISC5 = 0; // RW_PIN for LCD
                TRISCbits.TRISC6 = 0; // E_PIN for LCD
                TRISD=0x00; // 8-bit data port for LCD
                InitLCD(); // This function calls OpenXLCD function from Microchip,
                // which configures the I/O ports specified in LCD1602.h
                // file
                cambio=1;}


            // INTERRUPCION
            INTCONbits.TMR0IF = 0;  // Clear Interrupt Flag, do not forget it!
            // User code attending the event...
    }
}
```

1.2. Código del nodo sensor

1.2.1. Inclusiones de subrutinas

```
#include <p18f4520.h>
#include "ConfigurationBits.h" //Configuration bits
#include "LCD1602.h"
#include "DS18B20.h"
#include "1wire.h"
#include "UPC_nRF24L01.h"
#include <stdio.h>
#include <delays.h>
#include <timers.h>
```

1.2.2. Inicialización de variables

```
unsigned char direccion1[5]={0xB2, 0xB2, 0xB3, 0xB4, 0x01};
unsigned char direccion2[5]={0x11, 0x11, 0x11, 0x11, 0x11};
unsigned char direccion3[5]={0x41, 0x41, 0x41, 0x41, 0x41};
unsigned char direccion4[5]={0x71, 0x71, 0x71, 0x71, 0x71};
unsigned char direccion5[5]={0xD1, 0xD1, 0xD1, 0xD1, 0xD1};
unsigned char direccion6[5]={0xF1, 0xF1, 0xF1, 0xF1, 0xF1};
unsigned char direccion7[5]={0x21, 0x21, 0x21, 0x21, 0x21};
unsigned char direccion8[5]={0x81, 0x81, 0x81, 0x81, 0x81};
unsigned char direccion9[5]={0x91, 0x91, 0x91, 0x91, 0x91};
unsigned char direccion10[5]={0xA1, 0xA1, 0xA1, 0xA1, 0xA1};
unsigned char informe;

unsigned int i=20;

unsigned char direccion_transmision[5] = {0xB2, 0xB2, 0xB3, 0xB4, 0x01};
unsigned char T[9];
unsigned char temperatura[2];
unsigned char activacion[1];
```

1.2.3. Programa principal

```
void InterruptHandlerHigh();
void main()
{

TRISB=0x00;
TRISD=0xFF;
OpenTimer0(TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_2);
INTCON2bits.TMR0IP = 1; // High priority for Timer0
RCONbits.IPEN = 1; // Enable priority levels
INTCONbits.GIEH = 1; // Enable interrupts

if(!Detect_Slave_Device()) // Is slave present?
{}

else
{
while(1)
{
DS18B20_convert_temperature();
if(DS18B20_read_scratchpad(T)==1)
    {
    temperatura[0]=T[0];
    temperatura[1]=T[1];
    }
OSCCONbits.IDLEN=0;
}
}
}
```

1.2.4. Interrupciones

```
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh()
{
    if(INTCONbits.TMR0IF) // Has Timer0 overflow causing an interrupt?
    {
    i=i+1;

    if (i==10 || i>15)
        {
        SPI_Start(0b10);
        nRF24L01_Ports_Start();
        Start_RX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, direccion1,
            direccion2, 0x00, 0x00, 0x00, 0x00, 1, 1, 0, 0, 0, 0, 0);
        informe=Receive_Data_RX_Mode_nRF24L01(1, 100, 1, activacion);
        Finish_nRF24L01_Operation();
        Finish_SPI_Operation();

        if (activacion[0]==0b10000000)
            {
            SPI_Start(0b10);
            nRF24L01_Ports_Start();
            Start_TX_Mode_nRF24L01(0b11, 0b1000000, 0, 0b11, 1, 1, 0b0000, 10, 1, 2);
            Send_Data_TX_Mode_nRF24L01(1, 0b11, direccion_transmision, 2, temperatura);
            informe=Check_Data_Sent_TX_Mode_nRF24L01();
            Finish_nRF24L01_Operation();
            Finish_SPI_Operation();
            i=1;
            }
        }
    // INTERRUPCION
    INTCONbits.TMR0IF = 0;  // Clear Interrupt Flag, do not forget it!
    // User code attending the event...
    }
}
```

2. Subrutinas utilizadas

En este apartado se exponen las subrutinas utilizadas por los programas principales para la realización de la aplicación.

2.2. Subrutinas del sensor DS18B20

```
#include "ds18b20.h"
#include "LCD1602.h"
#include "1wire.h"
#include <stdio.h>

unsigned char Detect_Slave_Device(void)
{
    if (!OW_reset_pulse())
        return HIGH;
    else
        return LOW;
}

void DS18B20_convert_temperature(void)
{
    OW_reset_pulse();                    // Reset pulse
    OW_write_byte(CMD_18B20_SKIP_ROM);   // Skip ROM command
    OW_write_byte(CMD_18B20_CONV_TEMP); // Send a command to convert temperature
}

unsigned char DS18B20_read_scratchpad(unsigned char *sp)
{
    unsigned char i,crc;

    // Sequence to read scratchpad
    OW_reset_pulse();
    OW_write_byte(CMD_18B20_SKIP_ROM);
    OW_write_byte(CMD_18B20_READ_SPAD);
    for(i=0;i<9;i++)
    {
        sp[i] = OW_read_byte();
    }
    // Check CRC
    crc = calc_crc(sp,8);
    if(crc == sp[8])          // CRC OK from scratchpad
    {
        return(1);
    }
    else
    {
        return(0);            // CRC wrong from scratchpad
    }
}
```

```
unsigned char DS18B20_read_laser_rom_code(unsigned char *lrc)
{
    unsigned char i;
    unsigned char crc;
    OW_reset_pulse();
    OW_write_byte(CMD_18B20_READ_ROM);
    for(i=0;i<8;i++)
    {
        lrc[i]=OW_read_byte();
    }
    crc=calc_crc(lrc,7);
    if (crc==lrc[7])
    {
        while(BusyXLCD());
        putrsXLCD("CRC ok");
    }
    else
    {
        while(BusyXLCD());
        putrsXLCD("CRC KO");
    }

void DS18B20_decode_temp(unsigned char *sp, char *buffer)
{
unsigned int decimal;
unsigned char s;
unsigned int largo;
unsigned char signe;
unsigned char enter;
signe=sp[1]>>3;
if (signe==0)
{
decimal=(sp[0]&0x0F)*625;
enter=((sp[1]&0x07)<<4)|(sp[0]>>4);
}
else
{
largo=0x0000;
largo=((largo+sp[1])<<8)|sp[0];
largo=(~largo)+1;
decimal=((largo&0x00FF)&0x000F)*625;
enter=(((largo>>8)&0x0007)<<4)|((largo&0x00FF)>>4);
}
if (signe==0)
{s='+';}
else
{s='-';}
sprintf(buffer, "%c%u.%04u C", s, enter, decimal);      // To be completed by students

}


unsigned char BusyDS18B20(void)
{
    return(!OW_read_bit());
}
```

```
unsigned char calc_crc(unsigned char buff[], unsigned char num_vals)
{
   unsigned char shift_reg=0, data_bit, sr_lsb, fb_bit, i, j;

   for (i=0; i<num_vals; i++) /* for each byte */
   {
      for(j=0; j<8; j++)   /* for each bit */
      {
         data_bit = (buff[i]>>j)&0x01;
         sr_lsb = shift_reg & 0x01;
         fb_bit = (data_bit ^ sr_lsb) & 0x01;
         shift_reg = shift_reg >> 1;
         if (fb_bit)
         {
            shift_reg = shift_reg ^ 0x8c;
         }
      }
   }
   return(shift_reg);
}
```

2.3. Subrutinas de la pantalla LCD1602

```
#include <delays.h>
#include "lcd1602.h"


void InitLCD(void)
{
    OpenXLCD(EIGHT_BIT & LINES_5X7);            // Use 8 bit Data, 5x7 pixel Matrix per character
    ClearLCD();                                 // Clear display
    while(BusyXLCD());                          // Wait till LCD finishes executing command
    WriteCmdXLCD(DON & CURSOR_OFF & BLINK_OFF); // Display ON, Cursor OFF, Blink OFF
    while(BusyXLCD());                          // Wait till LCD finishes executing command
    WriteCmdXLCD(RETURNHOME);                   // Command home
    Delay10KTCYx(1);                            // Extra delay
}


void ClearLCD(void)
{
     while(BusyXLCD());                 // Wait till LCD finishes executing command
     WriteCmdXLCD(CLEARDISPLAY);        // Clear display
}
```

```
void gotoxy(unsigned char row , unsigned char col)
{
    if(row == 0)
    {
        while(BusyXLCD());           // Wait till LCD finishes executing command
        WriteCmdXLCD(LINE1 | col);   // Cursor on the first line
    }
    else
    {
        while(BusyXLCD());           // Wait till LCD finishes executing command
        WriteCmdXLCD(LINE2 | col);   // Cursor on the second line
    }
}

void OpenXLCD(unsigned char lcdtype)
{
        // The data bits must be either a 8-bit port or the upper or
        // lower 4-bits of a port. These pins are made into inputs
#ifdef BIT8                          // 8-bit mode, use whole port
        DATA_PORT = 0;
        TRIS_DATA_PORT = 0x00;
#else                                // 4-bit mode
#ifdef UPPER                         // Upper 4-bits of the port
        DATA_PORT &= 0x0f;
        TRIS_DATA_PORT &= 0x0F;
#else                                // Lower 4-bits of the port
        DATA_PORT &= 0xf0;
        TRIS_DATA_PORT &= 0xF0;
#endif
#endif
        TRIS_RW = 0;                 // All control signals made outputs
        TRIS_RS = 0;
        TRIS_E = 0;
        RW_PIN = 0;                  // R/W pin made low
        RS_PIN = 0;                  // Register select pin made low
        E_PIN = 0;                   // Clock pin made low

        // Delay for 15ms to allow for LCD Power on reset
        DelayPORXLCD();
 //-------------------reset procedure through software----------------------
        WriteCmdXLCD(0x30);
        Delay10KTCYx(0x05);

        WriteCmdXLCD(0x30);
        Delay10KTCYx(0x01);

        WriteCmdXLCD(0x32);
        while( BusyXLCD() );
//-------------------------------------------------------------------------------------------
```

```
        // Set data interface width, # lines, font
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(lcdtype);          // Function set cmd

        // Turn the display on then off
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(DOFF&CURSOR_OFF&BLINK_OFF);        // Display OFF/Blink OFF
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(DON&CURSOR_ON&BLINK_ON);           // Display ON/Blink ON

        // Clear display
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(0x01);             // Clear display

        // Set entry mode inc, no shift
        while(BusyXLCD());              // Wait if LCD busy
        WriteCmdXLCD(SHIFT_CUR_LEFT);   // Entry Mode

        // Set DD Ram address to 0
        while(BusyXLCD());              // Wait if LCD busy
        SetDDRamAddr(0x80);                // Set Display data ram address to 0

unsigned char BusyXLCD(void)
{
        RW_PIN = 1;                     // Set the control bits for read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock in the command
        DelayFor18TCY();
#ifdef BIT8                             // 8-bit interface
        if(DATA_PORT&0x80)                      // Read bit 7 (busy bit)
        {                               // If high
                E_PIN = 0;              // Reset clock line
                RW_PIN = 0;             // Reset control line
                return 1;               // Return TRUE
        }
        else                            // Bit 7 low
        {
                E_PIN = 0;              // Reset clock line
                RW_PIN = 0;             // Reset control line
                return 0;               // Return FALSE
        }
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        if(DATA_PORT&0x80)
#else                                   // Lower nibble interface
        if(DATA_PORT&0x08)
#endif
        {
                E_PIN = 0;              // Reset clock line
                DelayFor18TCY();
                E_PIN = 1;              // Clock out other nibble
                DelayFor18TCY();
                E_PIN = 0;
                RW_PIN = 0;             // Reset control line
                return 1;               // Return TRUE
        }
        else                            // Busy bit is low
```

```
        {
                E_PIN = 0;              // Reset clock line
                DelayFor18TCY();
                E_PIN = 1;              // Clock out other nibble
                DelayFor18TCY();
                E_PIN = 0;
                RW_PIN = 0;             // Reset control line
                return 0;               // Return FALSE
        }
#endif
}

void putrsXLCD(const rom char *buffer)
{
        while(*buffer)                  // Write data to LCD up to null
        {
                while(BusyXLCD());      // Wait while LCD is busy
                WriteDataXLCD(*buffer); // Write character to LCD
                buffer++;               // Increment buffer
        }
        return;
}

void putsXLCD(char *buffer)
{
        while(*buffer)                  // Write data to LCD up to null
        {
                while(BusyXLCD());      // Wait while LCD is busy
                WriteDataXLCD(*buffer); // Write character to LCD
                buffer++;               // Increment buffer
        }
        return;
}
```

```
unsigned char ReadAddrXLCD(void)
{
        char data;                      // Holds the data retrieved from the LCD

#ifdef BIT8                             // 8-bit interface
        RW_PIN = 1;                     // Set control bits for the read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data out of the LCD controller
        DelayFor18TCY();
        data = DATA_PORT;               // Save the data in the register
        E_PIN = 0;
        RW_PIN = 0;                     // Reset the control bits
#else                                   // 4-bit interface
        RW_PIN = 1;                     // Set control bits for the read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data out of the LCD controller
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data = DATA_PORT&0xf0;          // Read the nibble into the upper nibble of data
#else                                   // Lower nibble interface
        data = (DATA_PORT<<4)&0xf0;     // Read the nibble into the upper nibble of data
#endif
        E_PIN = 0;                      // Reset the clock
        DelayFor18TCY();
        E_PIN = 1;                      // Clock out the lower nibble
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data |= (DATA_PORT>>4)&0x0f;    // Read the nibble into the lower nibble of data
#else                                   // Lower nibble interface
        data |= DATA_PORT&0x0f;         // Read the nibble into the lower nibble of data
#endif
        E_PIN = 0;
        RW_PIN = 0;                     // Reset the control lines

#endif
        return (data&0x7f);              // Return the address, Mask off the busy bit
}
```

```
char ReadDataXLCD(void)
{
        char data;

#ifdef BIT8                             // 8-bit interface
        RS_PIN = 1;                     // Set the control bits
        RW_PIN = 1;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the data out of the LCD
        DelayFor18TCY();
        data = DATA_PORT;               // Read the data
        E_PIN = 0;
        RS_PIN = 0;                     // Reset the control bits
        RW_PIN = 0;
#else                                   // 4-bit interface
        RW_PIN = 1;
        RS_PIN = 1;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the data out of the LCD
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data = DATA_PORT&0xf0;          // Read the upper nibble of data
#else                                   // Lower nibble interface
        data = (DATA_PORT<<4)&0xf0;     // read the upper nibble of data
#endif
        E_PIN = 0;                      // Reset the clock line
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the next nibble out of the LCD
        DelayFor18TCY();
#ifdef UPPER                            // Upper nibble interface
        data |= (DATA_PORT>>4)&0x0f;    // Read the lower nibble of data
#else                                   // Lower nibble interface
        data |= DATA_PORT&0x0f;         // Read the lower nibble of data

#endif
        E_PIN = 0;
        RS_PIN = 0;                     // Reset the control bits
        RW_PIN = 0;
#endif
        return(data);                   // Return the data byte
}
```

```
void SetCGRamAddr(unsigned char CGaddr)
{
#ifdef BIT8                                     // 8-bit interface
        TRIS_DATA_PORT = 0;                     // Make data port ouput
        DATA_PORT = CGaddr | 0b01000000;        // Write cmd and address to port
        RW_PIN = 0;                             // Set control signals
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
        DelayFor18TCY();
        TRIS_DATA_PORT = 0xff;                  // Make data port inputs
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;                 // Make nibble input
        DATA_PORT &= 0x0f;                      // and write upper nibble
        DATA_PORT |= ((CGaddr | 0b01000000) & 0xf0);
#else                                           // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;                 // Make nibble input
        DATA_PORT &= 0xf0;                      // and write upper nibble
        DATA_PORT |= (((CGaddr |0b01000000)>>4) & 0x0f);
#endif
        RW_PIN = 0;                             // Set control signals
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                    // Upper nibble interface
        DATA_PORT &= 0x0f;                      // Write lower nibble
        DATA_PORT |= ((CGaddr<<4)&0xf0);
#else                                           // Lower nibble interface
        DATA_PORT &= 0xf0;                      // Write lower nibble
        DATA_PORT |= (CGaddr&0x0f);


#endif
        DelayFor18TCY();
        E_PIN = 1;                              // Clock cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                    // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;                 // Make inputs
#else                                           // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;                 // Make inputs
#endif
#endif
        return;
}
```

```
void SetDDRamAddr(unsigned char DDaddr)
{
#ifdef BIT8                                     // 8-bit interface
        TRIS_DATA_PORT = 0;                     // Make port output
        DATA_PORT = DDaddr | 0b10000000;        // Write cmd and address to port
        RW_PIN = 0;                             // Set the control bits
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
        DelayFor18TCY();
        TRIS_DATA_PORT = 0xff;                  // Make port input
#else                                           // 4-bit interface
#ifdef UPPER                                    // Upper nibble  interface
        TRIS_DATA_PORT &= 0x0f;                 // Make port output
        DATA_PORT &= 0x0f;                      // and write upper nibble
        DATA_PORT |= ((DDaddr | 0b10000000) & 0xf0);
#else                                           // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;                 // Make port output
        DATA_PORT &= 0xf0;                      // and write upper nibble
        DATA_PORT |= (((DDaddr | 0b10000000)>>4) & 0x0f);
#endif
        RW_PIN = 0;                             // Set control bits
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                              // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                    // Upper nibble interface
        DATA_PORT &= 0x0f;                      // Write lower nibble
        DATA_PORT |= ((DDaddr<<4)&0xf0);

#else                                             // Lower nibble interface
        DATA_PORT &= 0xf0;                        // Write lower nibble
        DATA_PORT |= (DDaddr&0x0f);
#endif
        DelayFor18TCY();
        E_PIN = 1;                                // Clock the cmd and address in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                                      // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;                   // Make port input
#else                                             // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;                   // Make port input
#endif
#endif
        return;
}
```

```
void WriteCmdXLCD(unsigned char cmd)
{
#ifdef BIT8                             // 8-bit interface
        TRIS_DATA_PORT = 0;             // Data port output
        DATA_PORT = cmd;                // Write command to data port
        RW_PIN = 0;                     // Set the control signals
        RS_PIN = 0;                     // for sending a command
        DelayFor18TCY();
        E_PIN = 1;                      // Clock the command in
        DelayFor18TCY();
        E_PIN = 0;
        DelayFor18TCY();
        TRIS_DATA_PORT = 0xff;          // Data port input
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;
        DATA_PORT &= 0x0f;
        DATA_PORT |= cmd&0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;
        DATA_PORT &= 0xf0;
        DATA_PORT |= (cmd>>4)&0x0f;
#endif
        RW_PIN = 0;                     // Set control signals for command
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock command in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;
        DATA_PORT |= (cmd<<4)&0xf0;
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;
        DATA_PORT |= cmd&0x0f;


#endif
        DelayFor18TCY();
        E_PIN = 1;                      // Clock command in
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Make data nibble input
        TRIS_DATA_PORT |= 0xf0;
#else
        TRIS_DATA_PORT |= 0x0f;
#endif
#endif
        return;
}
```

```
void WriteDataXLCD(char data)
{
#ifdef BIT8                             // 8-bit interface
        TRIS_DATA_PORT = 0;             // Make port output
        DATA_PORT = data;               // Write data to port
        RS_PIN = 1;                     // Set control bits
        RW_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock data into LCD
        DelayFor18TCY();
        E_PIN = 0;
        RS_PIN = 0;                     // Reset control bits
        TRIS_DATA_PORT = 0xff;          // Make port input
#else                                   // 4-bit interface
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT &= 0x0f;
        DATA_PORT &= 0x0f;
        DATA_PORT |= data&0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT &= 0xf0;
        DATA_PORT &= 0xf0;
        DATA_PORT |= ((data>>4)&0x0f);
#endif
        RS_PIN = 1;                     // Set control bits
        RW_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock nibble into LCD
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        DATA_PORT &= 0x0f;
        DATA_PORT |= ((data<<4)&0xf0);
#else                                   // Lower nibble interface
        DATA_PORT &= 0xf0;
        DATA_PORT |= (data&0x0f);

            _
#endif
        DelayFor18TCY();
        E_PIN = 1;                      // Clock nibble into LCD
        DelayFor18TCY();
        E_PIN = 0;
#ifdef UPPER                            // Upper nibble interface
        TRIS_DATA_PORT |= 0xf0;
#else                                   // Lower nibble interface
        TRIS_DATA_PORT |= 0x0f;
#endif
#endif
        return;
}
```

```
void DelayFor18TCY(void)
{
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
_asm NOP _endasm
}

void DelayPORXLCD(void)
{
    Delay1KTCYx(15);
}

void DelayXLCD(void)
{
    Delay1KTCYx(5);
}
```

2.4. Subrutinas del módulo de radiofrecuencia nRF24L01

```
#include<p18f4520.h>
#include<delays.h>
#include"UPC_nRF24L01.h"
```

```
void SPI_Start (unsigned char Clock_Frequency)
{
    TRISCbits.TRISC3=0;
    TRISCbits.TRISC4=1;
    TRISCbits.TRISC5=0;

    SSPCON1 = (SSPCON1 & 0xF0) | Clock_Frequency;


    SSPCON1bits.CKP=0;
    SSPSTATbits.CKE=1;
    SSPSTATbits.SMP=1;

    PIE1bits.SSPIE=0;
    IPR1bits.SSPIP=0;
    PIR1bits.SSPIF=0;

    SSPCON1bits.SSPEN=1;
}


void nRF24L01_Ports_Start(void)
{
    ADCON1=0x0F;

    TRISAbits.TRISA2=0;
    TRISAbits.TRISA3=0;
    TRISAbits.TRISA4=1;

    CSN=1;
    Delay10TCYx(5);
    CE=0;
}


unsigned char SPI_Transfer(unsigned char byte_to_send)
{
    SSPBUF = byte_to_send;

    while(PIR1bits.SSPIF==0);
    PIR1bits.SSPIF=0;

    return(SSPBUF);
}

unsigned char Read_nRF24L01_Register (unsigned char Register_Address)
{
    unsigned char Register_Content, nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);
    Register_Content=SPI_Transfer(0x00);
    CSN=1;
    Delay10TCYx(5);
    return(Register_Content);
}
```

```
void Write_nRF24L01_Register (unsigned char Register_Address, unsigned char Register_Content)
{
    unsigned char dummydata, nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
    dummydata=SPI_Transfer(Register_Content);
    CSN=1;
    Delay10TCYx(5);

    while(Read_nRF24L01_Register(Register_Address)!=Register_Content)
    {
        CSN=0;
        nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
        dummydata=SPI_Transfer(Register_Content);
        CSN=1;
        Delay10TCYx(5);
    }
}


unsigned char Read_nRF24L01_Status (void)
{
    unsigned char nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(NOP);
    CSN=1;
    Delay10TCYx(5);

    return(nRF24L01_Status);
}

void Write_nRF24L01_Status (unsigned char Register_Content)
{
    unsigned char dummydata, nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(W_REGISTER+STATUS);
    dummydata=SPI_Transfer(Register_Content);
    CSN=1;
    Delay10TCYx(5);
}


void Read_nRF24L01_Address_Register (unsigned char TX_RX_Address_Width,
            unsigned char Register_Address, unsigned char *Register_Content)
{
    unsigned char i, nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(R_REGISTER+Register_Address);
    for(i=0; i<(TX_RX_Address_Width+0b10); i++)
    {
        Register_Content[i]=SPI_Transfer(0x00);
    }
    CSN=1;
    Delay10TCYx(5);
}
```

```
void Write_nRF24L01_Address_Register (unsigned char TX_RX_Address_Width,
            unsigned char Register_Address, unsigned char *Register_Content)
{
    unsigned char dummydata, i, nRF24L01_Status;
    unsigned char Process_Finished;
    unsigned char Address_Verification[5];

    Process_Finished=0;
    while(!Process_Finished)
    {
        CSN=0;
        nRF24L01_Status=SPI_Transfer(W_REGISTER+Register_Address);
        for(i=0; i<(TX_RX_Address_Width+0b10); i++)
        {
            dummydata=SPI_Transfer(Register_Content[i]);
        }
        CSN=1;
        Delay10TCYx(5);

        Read_nRF24L01_Address_Register(TX_RX_Address_Width, Register_Address, Address_Verification);
        Process_Finished=1;
        for(i=0; i<(TX_RX_Address_Width+0b10); i++)
        {
            if(Register_Content[i]!=Address_Verification[i])
            {
                Process_Finished=0;
            }
        }
    }
}

unsigned char Read_nRF24L01_RX_Payload (unsigned char Enable_Checksum,
        unsigned char TX_RX_Payload_Width, unsigned char *RX_Payload)
{
    unsigned char nRF24L01_Status, TX_Checksum;
    unsigned char RX_Checksum, Checksum_Conclusion;
    unsigned char i;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(R_RX_PAYLOAD);
    if(!Enable_Checksum)
    {
        for(i=0; i<TX_RX_Payload_Width; i++)
        {
            RX_Payload[i]=SPI_Transfer(0x00);
        }
        Checksum_Conclusion=1;
    }
    else if(Enable_Checksum)
    {
        RX_Checksum=0b00000000;
        for(i=0; i<TX_RX_Payload_Width; i++)
        {
            RX_Payload[i]=SPI_Transfer(0x00);
            RX_Checksum=RX_Checksum+RX_Payload[i];

        }
        TX_Checksum=SPI_Transfer(0x00);

        if(RX_Checksum==TX_Checksum)
        {
            Checksum_Conclusion=1;
        }
        else if(RX Checksum!=TX Checksum)
```

```
        else if(RX_Checksum!=TX_Checksum)
        {
            Checksum_Conclusion=0;
        }

    }
    CSN=1;
    Delay10TCYx(5);

    return Checksum_Conclusion;
}


void Write_nRF24L01_TX_Payload (unsigned char Enable_Checksum,
        unsigned char TX_RX_Payload_Width, unsigned char *TX_Payload)
{
    unsigned char nRF24L01_Status, TX_Checksum;
    unsigned char i, dummydata;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(W_TX_PAYLOAD);
    if(!Enable_Checksum)
    {
        for(i=0; i<TX_RX_Payload_Width; i++)
        {
            dummydata=SPI_Transfer(TX_Payload[i]);
        }
    }
    else if(Enable_Checksum)
    {
        TX_Checksum=0b00000000;
        for(i=0; i<TX_RX_Payload_Width; i++)
        {
            dummydata=SPI_Transfer(TX_Payload[i]);
            TX_Checksum=TX_Checksum+TX_Payload[i];
        }
        dummydata=SPI_Transfer(TX_Checksum);
    }
    CSN=1;
    Delay10TCYx(5);
}
```

```
void Reset_nRF24L01_Status_and_nRF24L01_Payloads (void)
{
    unsigned char nRF24L01_Status;

    CSN=0;
    nRF24L01_Status=SPI_Transfer(FLUSH_TX);
    CSN=1;
    Delay10TCYx(5);

    CSN=0;
    nRF24L01_Status=SPI_Transfer(FLUSH_RX);
    CSN=1;
    Delay10TCYx(5);

    Write_nRF24L01_Status(RX_DR);
    Write_nRF24L01_Status(TX_DS);
    Write_nRF24L01_Status(MAX_RT);
}


void Start_RX_Mode_nRF24L01 (   unsigned char TX_RX_Address_Width,  unsigned char Frequency_Channel,
                                unsigned char RF_Data_Rate,         unsigned char RF_Output_Power,
                                unsigned char LNA_Gain,             unsigned char CRC_Setup,
                                unsigned char *RX_Pipe0_Address,    unsigned char *RX_Pipe1_Address,
                                unsigned char RX_Pipe2_Address,     unsigned char RX_Pipe3_Address,
                                unsigned char RX_Pipe4_Address,     unsigned char RX_Pipe5_Address,
                                unsigned char Enable_Checksum,      unsigned char RX_P0_Payload_Width,
                                unsigned char RX_P1_Payload_Width,  unsigned char RX_P2_Payload_Width,
                                unsigned char RX_P3_Payload_Width,  unsigned char RX_P4_Payload_Width,
                                unsigned char RX_P5_Payload_Width   )

{
    CE=0;

    Reset_nRF24L01_Status_and_nRF24L01_Payloads();

    Write_nRF24L01_Register (EN_AA, 0b00111111);
    Write_nRF24L01_Register (EN_RXADDR, 0b00111111);
    Write_nRF24L01_Register (SETUP_AW, TX_RX_Address_Width);
    Write_nRF24L01_Register (RF_CH, Frequency_Channel);
    Write_nRF24L01_Register (RF_SETUP, RF_Data_Rate*0b1000 + RF_Output_Power*0b10 + LNA_Gain);

    Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0, RX_Pipe0_Address);
    Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P1, RX_Pipe1_Address);
    Write_nRF24L01_Register (RX_ADDR_P2, RX_Pipe2_Address);
    Write_nRF24L01_Register (RX_ADDR_P3, RX_Pipe3_Address);
    Write_nRF24L01_Register (RX_ADDR_P4, RX_Pipe4_Address);
    Write_nRF24L01_Register (RX_ADDR_P5, RX_Pipe5_Address);

    Write_nRF24L01_Register (RX_PW_P0, RX_P0_Payload_Width+Enable_Checksum);
    Write_nRF24L01_Register (RX_PW_P1, RX_P1_Payload_Width+Enable_Checksum);
    Write_nRF24L01_Register (RX_PW_P2, RX_P2_Payload_Width+Enable_Checksum);
    Write_nRF24L01_Register (RX_PW_P3, RX_P3_Payload_Width+Enable_Checksum);
    Write_nRF24L01_Register (RX_PW_P4, RX_P4_Payload_Width+Enable_Checksum);
    Write_nRF24L01_Register (RX_PW_P5, RX_P5_Payload_Width+Enable_Checksum);

    Write_nRF24L01_Register (CONFIG, EN_CRC + CRC_Setup*0b100 + PWR_UP + PRIM_RX);
    Delay100TCYx(15);
}
```

```
void Start_TX_Mode_nRF24L01 (   unsigned char TX_RX_Address_Width,  unsigned char Frequency_Channel,
                                unsigned char RF_Data_Rate,             unsigned char RF_Output_Power,
                                unsigned char LNA_Gain,             unsigned char CRC_Setup,
                                unsigned char Auto_Retransmit_Delay,    unsigned char Max_Auto_Retransmit,
                                unsigned char Enable_Checksum,      unsigned char TX_RX_Payload_Width    )

{
    CE=0;

    Reset_nRF24L01_Status_and_nRF24L01_Payloads();

    Write_nRF24L01_Register (EN_AA, 0b00111111);
    Write_nRF24L01_Register (EN_RXADDR, 0b00111111);
    Write_nRF24L01_Register (SETUP_AW, TX_RX_Address_Width);
    Write_nRF24L01_Register (SETUP_RETR, Auto_Retransmit_Delay*0b10000 + Max_Auto_Retransmit);
    Write_nRF24L01_Register (RF_CH, Frequency_Channel);
    Write_nRF24L01_Register (RF_SETUP, RF_Data_Rate*0b1000 + RF_Output_Power*0b10 + LNA_Gain);

    Write_nRF24L01_Register (RX_PW_P0, TX_RX_Payload_Width+Enable_Checksum);

    Write_nRF24L01_Register (CONFIG,EN_CRC + CRC_Setup*0b100 + PWR_UP + 0*PRIM_RX);
    Delay100TCYx(15);
}


unsigned char Receive_Data_RX_Mode_nRF24L01 (unsigned char Enable_Checksum,
        unsigned char Max_Waiting_ds, unsigned char TX_RX_Payload_Width, unsigned char *RX_Payload)
{
    unsigned char nRF24L01_Status;
    unsigned char Origin_Pipe;
    unsigned char Checksum_Conclusion, IRQ_Error;
    unsigned int i;
    unsigned char j;

    IRQ_Error=0;
    CE=1;
    Delay10TCYx(13);
    Delay10TCYx(1);

    i=0; j=0;
    while(IRQ)
    {
        i++;
        if(i==3333)
        {
            i=0; j++;
        }
        if(j==Max_Waiting_ds)
        {
            IRQ_Error=1;
            break;
        }
    }
    CE=0;
    Delay10TCYx(1);


    nRF24L01_Status=Read_nRF24L01_Status();
    Origin_Pipe=(nRF24L01_Status & RX_P_NO);
    Checksum_Conclusion=Read_nRF24L01_RX_Payload(Enable_Checksum,TX_RX_Payload_Width, RX_Payload);
    Write_nRF24L01_Status(RX_DR);

    return (IRQ_Error*0b10000+Origin_Pipe+Checksum_Conclusion);
}
```

```
void Send_Data_TX_Mode_nRF24L01(unsigned char Enable_Checksum, unsigned char TX_RX_Address_Width,
        unsigned char *TX_Address, unsigned char TX_RX_Payload_Width, unsigned char *TX_Payload)
{
    Write_nRF24L01_Address_Register (TX_RX_Address_Width, TX_ADDR, TX_Address);
    Write_nRF24L01_Address_Register (TX_RX_Address_Width, RX_ADDR_P0, TX_Address);

    Write_nRF24L01_TX_Payload (Enable_Checksum, TX_RX_Payload_Width, TX_Payload);

    CE=1;
    Delay10TCYx(13);
    Delay10TCYx(1);
    CE=0;
    Delay10TCYx(1);
}


 unsigned char Check_Data_Sent_TX_Mode_nRF24L01 (void)
 {
     unsigned char TX_Operation_Result, TX_Retransmit_Counter;
     unsigned char nRF24L01_Status;
     unsigned int i;
     unsigned char j;

     i=0; j=0;
     while(IRQ)
     {
         i++;
         if(i==333)
         {
             i=0; j++;
         }
         if(j==7)
         {
             TX_Operation_Result=0b11;
             break;
         }
     }

     while(!IRQ)
     {
         nRF24L01_Status=Read_nRF24L01_Status();

         if(nRF24L01_Status & MAX_RT)
         {
             TX_Operation_Result=0b00;
             TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;
             Write_nRF24L01_Status(MAX_RT);
         }

        else if(nRF24L01_Status & TX_DS)
        {
            TX_Operation_Result=0b01;
            TX_Retransmit_Counter=Read_nRF24L01_Register(OBSERVE_TX) & 0x0F;
            Write_nRF24L01_Status(TX_DS);
        }
    }

    return (TX_Retransmit_Counter*0b100 + TX_Operation_Result);
}
```

```
void Finish_nRF24L01_Operation (void)
{
    unsigned char previous_config_register;
    previous_config_register=Read_nRF24L01_Register(CONFIG);
    Write_nRF24L01_Register(CONFIG, previous_config_register & 0b11111101);
}


void Finish_SPI_Operation (void)
{
    SSPCON1bits.SSPEN=0;
}


void Easy_Start_RX_Mode_nRF24L01(unsigned char Frequency_Channel,
        unsigned char RF_Output_Power, unsigned char *RX_Pipe0_Address)
{
        unsigned char Void_Address[5] = {0x00, 0x00, 0x00, 0x00, 0x00};

        SPI_Start(0b10);
        nRF24L01_Ports_Start();
        Start_RX_Mode_nRF24L01(0b11, Frequency_Channel, 0, RF_Output_Power, 1, 1,
                RX_Pipe0_Address, Void_Address, 0x00, 0x00, 0x00, 0x00, 1, 8, 0, 0, 0, 0, 0);
}


void Easy_Start_TX_Mode_nRF24L01(unsigned char Frequency_Channel,
         unsigned char RF_Output_Power, unsigned char Max_Auto_Retransmit)
{
         SPI_Start(0b10);
         nRF24L01_Ports_Start();
         Start_TX_Mode_nRF24L01(0b11, Frequency_Channel, 0, RF_Output_Power,
                 1, 1, 0b0000, Max_Auto_Retransmit, 1, 8);
}


unsigned char Easy_Receive_Data_RX_Mode_nRF24L01 (unsigned char *RX_Payload)
{
    return(Receive_Data_RX_Mode_nRF24L01(1, 100, 8, RX_Payload));
}


void Easy_Send_Data_TX_Mode_nRF24L01 (unsigned char *TX_Address, unsigned char *TX_Payload)
{
    Send_Data_TX_Mode_nRF24L01(1, 0b11, TX_Address, 8, TX_Payload);
}
```

2.5. Subrutinas del bus 1wire

```
#include "1wire.h"


void drive_OW_low (void)
{
    OW_WRITE_PIN = LOW;
    OW_PIN_DIRECTION = OUTPUT;
}


void drive_OW_high (void)
{
    OW_WRITE_PIN = HIGH;
    OW_PIN_DIRECTION = OUTPUT;
}


unsigned char read_OW (void)
{
    OW_WRITE_PIN = INPUT;

     if (HIGH == OW_READ_PIN)
     {
        return(SET);
     }
     else
     {
        return(CLEAR);
     }
}


unsigned char OW_reset_pulse(void)
{
    unsigned char presence_detect;

    drive_OW_low();                 // Drive the bus low
    DELAY_240Us;                    // Delay 480 microseconds
    DELAY_240Us;
    drive_OW_high ();               // Release the bus
    DELAY_70Us;                     // Delay 70 microseconds
    presence_detect = read_OW();    // Sample for presence pulse from slave
    DELAY_205Us;                    // Delay 410 microseconds
    DELAY_205Us;
    drive_OW_high ();               // Release the bus
    return presence_detect;         // Return the presence pulse
}
```

```
void OW_write_bit (unsigned char write_bit)
{
    if (write_bit)
    {
        //writing a bit '1'
        drive_OW_low();          // Drive the bus low
        DELAY_6Us;               // Delay 6 microseconds
        drive_OW_high ();        // Release the bus
        DELAY_64Us;              // Delay 64 microseconds
    }
    else
    {
        //writing a bit '0'
        drive_OW_low();          // Drive the bus low
        DELAY_60Us;              // Delay 60 microseconds
        drive_OW_high ();        // Release the bus
        DELAY_10Us;              // Delay 10 microseconds for recovery
    }
}

unsigned char OW_read_bit (void)
{
    unsigned char read_data;
    //reading a bit
    drive_OW_low();                  // Drive the bus low
    DELAY_6Us;                       // delay 6 microseconds
    drive_OW_high ();                // Release the bus
    Nop();                           // delay 3 microseconds
    Nop();
    Nop();
    read_data = read_OW();           // Read the status of OW_PIN
    DELAY_55Us;                      // Delay 55 microseconds
    return read_data;
}

void OW_write_byte (unsigned char write_data)
{
    unsigned char loop;
    for (loop = 0; loop < 8; loop++)
    {
        OW_write_bit(write_data & 0x01);
        write_data >>= 1;
    }
}
```

```
unsigned char OW_read_byte (void)
{
    unsigned char loop, result=0;
    for (loop = 0; loop < 8; loop++)
    {

        result >>= 1;
        if (OW_read_bit())
        result |= 0x80;
    }
    return result;
}
```