



UNIVERSITAT POLITÈCNICA DE CATALUNYA  
BARCELONATECH  
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

**Grau en Enginyeria Electrònica Industrial i Automàtica**

**DISSENY I MUNTATGE D' UN SISTEMA D' ADQUISICIÓ DE  
DADES PER UN QUADRICÒPTER**



**Annexos**

**Autor:** Nil Mira Tura  
**Director:** Manuel Andrés Manzanares Brotons  
**Convocatòria:** Juny 2020



## Índex

1.	PROGRAMA PRINCIPAL	1
2.	LLIBRERIA GPS	7
3.	LLIBRERIA DS1337	11
4.	LLIBRERIA 24LC256	13
5.	LLIBRERIA BME280	14
6.	CODI DE L' ESTACIÓ RECEPTORA DE DADES (ARDUINO)	19



# 1. Programa Principal

```
#include <18F46J53.h> //Llibreria microcontrolador PIC 18F46J53
#device adc = 10 //Definim l' ADC intern del microcontrolador de 10 bits
#device PASS_STRINGS = IN_RAM //Pel GPS (Buscar utilitat, sense això error a la seva llibreria)
#use delay(crystal=16000000) //Definim un cristall extern a 16MHz
#fuses NOWDT, HS //Fuses de NO WatchDog Timer i cristall de "High Speed"

##### COMUNICACIONS #####
#use rs232(baud = 9600, xmit=pin_c6, rcv=pin_c7,bits=8, parity=N, stream=BT) //Definim la comunicació UART del bluetooth
#use rs232(baud = 9600, xmit=pin_a1, rcv=pin_a5,bits=8, parity=N, stream=GPS, ERRORS) //Definim la comunicació UART del GPS, afegim els errors en comunicació
#use i2c(MASTER, sda=PIN_B5, scl=PIN_B4) //Definim el bus I2C

##### LLIBRERIES #####
#include <math.h> //Llibreria matemàtica
#include <stdio.h>
#include <GPS_Lib.c> //Llibreria GPS
#include <DS1337.c> //Llibreria rellotge DS1337 (vàlid per DS1307)
#include <I2C_24LC256.c> //Llibreria memòria EEPROM externa 24LC256
#include <floatee.c> //Llibreria per escriure dades del tipus "float" a la EEPROM externa
#include <BME280_Lib.c> //Llibreria sensor temperatura, pressió i humitat

//Inicialitzacions funcions
void init_rs232();
void init_i2c();
void init_adc();
//Temperatura motors
void get_temp();
//GPS
void get_gps();
//DS1337
void init_rtc();
//BMP180
void get_bme();

//Enviar trama
void send_data();
//Emmagatzemar dades
void store_data();
//Printar dades
void read_data();
//Borrar dades
void erase_data();

##### VARIABLES #####
//Temperatures motors
int temp1, temp2, temp3, temp4;
//Hora (i print de les hores, a modificar)
int hr, min, sec;
int day, mth, year, dow; //Dow es el dia de la setmana
long int address=0;
//Recuperar dades
long int address_read=0;
int day_read,mth_read,year_read,hr_read,min_read,sec_read;
float tempBMP_read,presBMP_read,altBMP_read,humBMP_read;
float lat_read,lon_read,altGPS_read;
int sat_read;
int temp1_read,temp2_read,temp3_read,temp4_read;

//Timer EEPROM
int1 flag_eeprom=0; //Flag timer
int16 time_store = 0x85EE; //2 segons
//GPS
float lat, lon, altGPS, spd;
int sat;
//BME280
int32 Tdec;
```

```

int32 Pdec;
float altBMP;
int32 Humdec;
float T_Cent;
float P_mBar;
float H_Rel;
//Timer 1
int count_timer1=0;
##### INTERRUPTIIONS #####
#int_TIMER0
void TIMER0_isr() //Interrupció que escriu a la memòria
{
  flag_eeprom=1;
}

#int_TIMER1 //Fem que un led parpadegi al enviar dades
void TIMER1_isr()
{
  count_timer1++;
  if(count_timer1==4)
  {
    count_timer1=0;
    output_low(PIN_A2);
    disable_interrups(INT_TIMER1);
  }
}

##### PROGRAMA PRINCIPAL #####
void main()
{
  address=0;
  address_read=0;
  delay_us(100);

  init_i2c();
  delay_us(100);
  init_adc();
  delay_us(100);
  BME280_begin(MODE_NORMAL);

  delay_ms(100);
  init_rtc(); //Inicialitzem el rellotge
  while(1)
  {
    if(flag_eeprom) //Interrupció timer 2 segon. Fins que el GPS no ha acabat no pot començar
    {
      flag_eeprom=0;
      store_data();
      read_data();
      set_timer0(time_store);
    }

    if(GPSRead()) //Detecció satèl·lits
    {
      get_gps();
      if(lat != 0 && lon != 0)
        output_high(PIN_A0); //Led de funcionament GPS no dona 0
      else
        output_low(PIN_A0); //GPS dona 0
      //TEMPERATURA MOTORS
      get_temp();
      //DADES BMP180
      get_bme();
    }
  }
}

```

```

        //Data i hora
        rtc_get_time(year,mth,dow,day,hr,min,sec);
        //Enviar trama
        send_data();
        output_high(PIN_A2); //Encenem led
        enable_interrupts(INT_TIMER1);
        set_timer1(0); //Apaguem al cap de 0,5 segons
    }
}
}

##### MÈTODES #####
void init_rs232()
{
    //RS232
    set_tris_c(0b10000000); //Definim el pin C7 com a entrada (Receive del GPS)
    set_tris_a(0b00100000); //Definim el pin A5 com a entrada (Receive del BT)
    output_low(PIN_A0); //Led de funcionament
    output_low(PIN_A2);
    output_low(PIN_A3);
}
void init_i2c()
{
    //Timers
    setup_timer_0(T0_INTERNAL | T0_DIV_256); //Definim el timer 0 amb oscilador intern i prescaler de 256
    set_timer0(time_store); //Valor calcular mitjançant fórmula (Calculat per a 1 segon)
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
    set_tris_b(0b00111111); //Declarem els pins SDA i SCL com a entrada.
    enable_interrupts(INT_TIMER0); //Activem interrupcions del timer 0
    enable_interrupts(INT_TIMER1);
    enable_interrupts(global); //Activem interrupcions globals
}
{
    //ADC
    setup_adc(ADC_CLOCK_INTERNAL); //ADC amb rellotge intern
    setup_adc_ports(sAN8|sAN9|sAN10|sAN12,VSS_VDD); //Definim els ports analítics on hi ha els 4 TMP-35. Ja els hem declarat com a entrada anteriorment
}
void init_rtc()
{
    day=8;
    mth=5;
    year=20;
    dow=5;
    hr=10;
    min=00;
    sec=0;
    rtc_set_time(day,mth,year,dow,hr,min,sec);
}
void get_temp()
{
    //Temperatura motor 1
    set_adc_channel(12);
    delay_us(20);
    temp1=((Read_adc()/1024)*3.3)*100;
    delay_us(20);
    //Temperatura motor 2
    set_adc_channel(10);
    delay_us(20);
    temp2=((Read_adc()/1024)*3.3)*100;
    delay_us(20);
    //Temperatura motor 3
    set_adc_channel(8);
    delay_us(20);
    temp3=((Read_adc()/1024)*3.3)*100;
    delay_us(20);
}

```

```

//Temperatura motor 4
set_adc_channel(9);
delay_us(20);
temp4=((Read_adc()/1024)*3.3)*100;
delay_us(20);
}
void get_gps()
{
  lat = Latitude();
  lon = Longitude();
  altGPS = Altitude();
  sat = Satellites();
  spd = Speed();
}
void get_bme()
{
  BME280_readTemperature(&Tdec);
  BME280_readPressure(&Pdec);
  BME280_readHumidity(&Humdec);

  T_Cent = Tdec/100 + (float)((Tdec%100)/100);
  P_mBar = Pdec/100 + (float)((Pdec%100)/100);
  H_rel = Humdec/1024 + (float)((((Humdec * 100)/1024) % 100)/100);
  altBMP = BMP280Altitude(P_mBar);
}
void store_data()
{
  write_ext_eeprom(address,day);
  address = address+1;
  write_ext_eeprom(address,mth);
  address = address+1;
  write_ext_eeprom(address,year);
  address = address+1;

  write_ext_eeprom(address,hr);
  address = address+1;
  write_ext_eeprom(address,min);
  address = address+1;
  write_ext_eeprom(address,sec);
  address = address+1;
  write_float_ext_eeprom(address,T_Cent);
  address = address+4;
  write_float_ext_eeprom(address,P_mBar);
  address = address+4;
  write_float_ext_eeprom(address,altBMP);
  address = address+4;
  write_float_ext_eeprom(address,H_rel);
  address = address+4;
  write_float_ext_eeprom(address,lat);
  address = address+4;
  write_float_ext_eeprom(address,lon);
  address = address+4;
  write_float_ext_eeprom(address,altGPS);
  address = address+4;
  write_ext_eeprom(address,sat);
  address = address+1;
  write_ext_eeprom(address,temp1);
  address = address+1;
  write_ext_eeprom(address,temp2);
  address = address+1;
  write_ext_eeprom(address,temp3);
  address = address+1;
  write_ext_eeprom(address,temp4);
  address = address+1;
}
void read_data()
{

```

```

day_read = read_ext_eeprom(0+address_read);
mth_read = read_ext_eeprom(1+address_read);
year_read = read_ext_eeprom(2+address_read);
hr_read = read_ext_eeprom(3+address_read);
min_read = read_ext_eeprom(4+address_read);
sec_read = read_ext_eeprom(5+address_read);
tempBMP_read = read_float_ext_eeprom(6+address_read);
presBMP_read = read_float_ext_eeprom(10+address_read);
altBMP_read = read_float_ext_eeprom(14+address_read);
humBMP_read = read_float_ext_eeprom(18+address_read);
lat_read = read_float_ext_eeprom(22+address_read);
lon_read = read_float_ext_eeprom(26+address_read);
altGPS_read = read_float_ext_eeprom(30+address_read);
sat_read = read_ext_eeprom(34+address_read);
temp1_read = read_ext_eeprom(35+address_read);
temp2_read = read_ext_eeprom(36+address_read);
temp3_read = read_ext_eeprom(37+address_read);
temp4_read = read_ext_eeprom(38+address_read);

address_read+=39;

/*fprintf(BT,"DADES MEMORIA: ");
fprintf(BT,"%u", day_read);
fprintf(BT,"%u", mth_read);
fprintf(BT,"%u", year_read);
fprintf(BT,"%u", hr_read);
fprintf(BT,"%u", min_read);
fprintf(BT,"%u", sec_read);
fprintf(BT,"% .1f", tempBMP_read);
fprintf(BT,"% .1f", presBMP_read);
fprintf(BT,"% .1f", altBMP_read);
fprintf(BT,"% .1f", humBMP_read);
fprintf(BT,"% .6f", lat_read);

fprintf(BT,"% .6f", lon_read);
fprintf(BT,"% .1f", altGPS_read);
fprintf(BT,"%d", sat_read);
fprintf(BT,"%d", temp1_read);
fprintf(BT,"%d", temp2_read);
fprintf(BT,"%d", temp3_read);
fprintf(BT,"%dF \n", temp4_read);*/
}
void send_data()
{
    fprintf(BT,"%u", day);
    fprintf(BT,"%u", mth);
    fprintf(BT,"%u", year);
    fprintf(BT,"%u", hr);
    fprintf(BT,"%u", min);
    fprintf(BT,"%u", sec);
    fprintf(BT,"% .1f", T_Cent);
    fprintf(BT,"% .1f", P_mBar);
    fprintf(BT,"% .1f", altBMP);
    fprintf(BT,"% .1f", H_re1);
    fprintf(BT,"% .6f", lat);
    fprintf(BT,"% .6f", lon);
    fprintf(BT,"% .1f", altGPS);
    fprintf(BT,"%d", sat);
    fprintf(BT,"%d", temp1);
    fprintf(BT,"%d", temp2);
    fprintf(BT,"%d", temp3);
    fprintf(BT,"%dF \n", temp4);
}
void erase_data()
{
    for(int i=0;i<=32768;i++)
    {
        write_ext_eeprom(i,0xFF);
    }
}

```



## 2. Llibreria GPS

```

#define _GPRMC_ 1
#define _GPGGA_ 2
#define _OTHER_ 3

#include <stdint.h>
#include <stdlib.h>
#include <string.h>

int1 GPRMC_ok = 0, GPGGA_ok = 0;
uint8_t char_number = 0, SentenceType = 0, Term;
char sentence[6], rawTime[11], rawDate[7], rawSpeed[6], rawCourse[6], rawSatellites[3],
rawLatitude[13], rawLongitude[13], rawAltitude[7], buffer[12];

void stringcpy(char *str1, char *str2, int1 dir = 0) {
uint8_t chr = 0;
do {
str2[chr + dir] = str1[chr];
} while(str1[chr++] != '\0');
}

int1 GPSRead() {
uint8_t c = getc(GPS);

switch(c) {
case '\r': // sentence end
if(SentenceType == _GPRMC_)
GPRMC_ok = 1;
if(SentenceType == _GPGGA_)
GPGGA_ok = 1;
if(GPRMC_ok && GPGGA_ok) {
GPRMC_ok = GPGGA_ok = 0;
return 1;
}
break;

case '$': // sentence start
Term = char_number = 0;
break;

case ',': // term end (new term start)
buffer[char_number] = '\0';
if(Term == 0) {
stringcpy(buffer, sentence);
if(strcmp(sentence, "GPRMC") == 0)
SentenceType = _GPRMC_;
else if(strcmp(sentence, "GPGGA") == 0)
SentenceType = _GPGGA_;
else
SentenceType = _OTHER_;
}

// Time
if(Term == 1 && SentenceType == _GPRMC_) {
stringcpy(buffer, rawTime);
}

// Latitude
if((Term == 3) && (SentenceType == _GPRMC_)) {
stringcpy(buffer, rawLatitude, 1);
}

// Latitude N/S
if((Term == 4) && (SentenceType == _GPRMC_)) {
if(buffer[0] == 'N')
rawLatitude[0] = '0';
else

```

```

    rawLatitude[0] = '-';
}

// Longitude
if((Term == 5) && (SentenceType == _GPRMC_)) {
    strcpy(buffer, rawLongitude, 1);
}
// Longitude E/W
if((Term == 6) && (SentenceType == _GPRMC_)) {
    if(buffer[0] == 'E')
        rawLongitude[0] = '0';
    else
        rawLongitude[0] = '-';
}

// Speed
if((Term == 7) && (SentenceType == _GPRMC_)) {
    strcpy(buffer, rawSpeed);
}

// Course
if((Term == 8) && (SentenceType == _GPRMC_)) {
    strcpy(buffer, rawCourse);
}

// Date
if((Term == 9) && SentenceType == _GPRMC_) {
    strcpy(buffer, rawDate);
}

// Satellites
if((Term == 7) && (SentenceType == _GPGGA_)) {
}

// Altitude
if((Term == 9) && (SentenceType == _GPGGA_)) {
    strcpy(buffer, rawAltitude);
}
Term++;
char_number = 0;
break;

default:
    buffer[char_number++] = c;
    break;
}

return 0;
}

uint8_t GPSSecond() {
    return ((rawTime[4] - '0') * 10 + (rawTime[5] - '0'));
}
uint8_t GPSMinute() {
    return ((rawTime[2] - '0') * 10 + (rawTime[3] - '0'));
}
uint8_t GPShour() {
    return ((rawTime[0] - '0') * 10 + (rawTime[1] - '0'));
}

uint8_t GPSDay() {
    return ((rawDate[0] - '0') * 10 + (rawDate[1] - '0'));
}
uint8_t GPSMonth() {
    return ((rawDate[2] - '0') * 10 + (rawDate[3] - '0'));
}

```

```
}
uint8_t GPSYear() {
    return ((rawDate[4] - '0') * 10 + (rawDate[5] - '0'));
}

float parse_rawDegree(char *term_) {
    float term_value = atof(term_)/100;
    int16_t term_dec = term_value;
    term_value -= term_dec;
    term_value = term_value * 5/3 + term_dec;
    return term_value;
}

float Latitude() {
    return parse_rawDegree(rawLatitude);
}

float Longitude() {
    return parse_rawDegree(rawLongitude);
}

float Altitude() {
    return atof(rawAltitude);
}

uint8_t Satellites() {
    return atoi(rawSatellites);
}

float Speed() {
    return (atof(rawSpeed) * 1.852);
}

float Course() {
    return atof(rawCourse);
}
```



### 3. Llibreria DS1337

```

//Les dades obtingudes del rellotge són en format BCD. D' aquesta manera les convertim a binàries.
int BCDBIN(byte bcd)
{
    int varia;
    varia=bcd;
    varia>>=1;
    varia &= 0x78;
    return(varia + (varia>>2) + (bcd & 0x0f));
}

BYTE BINABCD(BYTE bin)
{
    BYTE temp;
    BYTE retval;

    temp = bin;
    retval=0;

    while(true)
    {
        if(temp>=10)
        {
            temp-=10;
            retval+=0x10;
        }
        else
        {
            retval+=temp;
            break;
        }
    }
    return(retval);
}

void rtc_get_time(byte& yr, byte& mth, byte& dt, byte& dy, byte& hr, byte& min, byte& sec){
    i2c_start(); //Inici comunicació escriptura
    i2c_write(0xd0); //Paraula de control d' escriptura
    i2c_write(0x00); //Puntero a la primera direcció
    i2c_start(); //Inici de comunicació lectura
    i2c_write(0xd1); //Paraula de control de lectura
    sec=BCDBIN(i2c_read()&0x7f); //Lectura dels 7 bits dels segons
    min=BCDBIN(i2c_read()&0x7f);
    hr=BCDBIN(i2c_read()&0x3f); //Lectura dels 6 bits de les hores
    dt=BCDBIN(i2c_read()&0x07);
    dy=BCDBIN(i2c_read()&0x3f);
    mth=BCDBIN(i2c_read()&0x1f);
    yr=BCDBIN(i2c_read()&0xff);
    i2c_stop(); //Final de la lectura
}

void rtc_set_time(BYTE day, BYTE mth, BYTE year, BYTE dow, BYTE hr, BYTE min, BYTE sec)
{
    sec &=0x7F;
    hr &=0x3F;

    i2c_start();
    i2c_write(0xd0);
    i2c_write(0x00);
    i2c_write(BINABCD(sec));
    i2c_write(BINABCD(min));
    i2c_write(BINABCD(hr));
    i2c_write(BINABCD(dow));
    i2c_write(BINABCD(day));
    i2c_write(BINABCD(mth));
    i2c_write(BINABCD(year));
}

```



## 4. Llibreria 24LC256

```
void write_ext_eeprom(long int address, byte data){
    short int status;
    i2c_start(); //Inici comunicació escriptura
    i2c_write(0xa0); //Paraula de control escriptura
    i2c_write(address>>8); //Part alta direcció
    i2c_write(address); //Part baixa direcció
    i2c_write(data); //Dada a escriure
    i2c_stop(); //Final comunicació escriptura
    i2c_start(); //Reiniciem
    status=i2c_write(0xa0); //Lectura del bit ACK

    while (status == 1) //Si és 1 es segueix esperant
    {
        i2c_start();
        status=i2c_write(0xa0);
    }
    i2c_stop(); //Si és 0, finalitza la comunicació
}

byte read_ext_eeprom(long int address){
    byte data;
    i2c_start(); //Inici comunicació escriptura
    i2c_write(0xa0); //Paraula de control escriptura

    i2c_write(address>>8); //Part alta direcció
    i2c_write(address); //Part baixa direcció

    i2c_start(); //Inici comunicació lectura
    i2c_write(0xa1); //Paraula de control lectura

    data=i2c_read(0); //Lectura de la dada
    i2c_stop(); //Final comunicació
    return(data);
}
```

## 5. Llibreria BME280

```

#include <stdint.h>

#ifndef BME280_I2C_ADDRESS
#define BME280_I2C_ADDRESS 0xEE
#endif
float altitudebme;
#define BME280_CHIP_ID 0x60

#define BME280_REG_DIG_T1 0x88
#define BME280_REG_DIG_T2 0x8A
#define BME280_REG_DIG_T3 0x8C

#define BME280_REG_DIG_P1 0x8E
#define BME280_REG_DIG_P2 0x90
#define BME280_REG_DIG_P3 0x92
#define BME280_REG_DIG_P4 0x94
#define BME280_REG_DIG_P5 0x96
#define BME280_REG_DIG_P6 0x98
#define BME280_REG_DIG_P7 0x9A
#define BME280_REG_DIG_P8 0x9C
#define BME280_REG_DIG_P9 0x9E

#define BME280_REG_DIG_H1 0xA1
#define BME280_REG_DIG_H2 0xE1
#define BME280_REG_DIG_H3 0xE3
#define BME280_REG_DIG_H4 0xE4
#define BME280_REG_DIG_H5 0xE5
#define BME280_REG_DIG_H6 0xE7

#define BME280_REG_CHIPID 0xD0
#define BME280_REG_SOFTRESET 0xE0

#define BME280_REG_CTRLHUM 0xF2
#define BME280_REG_STATUS 0xF3
#define BME280_REG_CONTROL 0xF4
#define BME280_REG_CONFIG 0xF5
#define BME280_REG_PRESS_MSB 0xF7

int32_t adc_T, adc_P, adc_H, t_fine;

// BME280 sensor modes, register ctrl_meas mode[1:0]
enum bme280_mode
{
    MODE_SLEEP = 0x00, // sleep mode
    MODE_FORCED = 0x01, // forced mode
    MODE_NORMAL = 0x03 // normal mode
};

// oversampling setting. osrs_h[2:0], osrs_t[2:0], osrs_p[2:0]
enum bme280_sampling
{
    SAMPLING_SKIPPED = 0x00, //skipped, output set to 0x80000 (0x8000 for humidity)
    SAMPLING_X1 = 0x01, // oversampling x1
    SAMPLING_X2 = 0x02, // oversampling x2
    SAMPLING_X4 = 0x03, // oversampling x4
    SAMPLING_X8 = 0x04, // oversampling x8
    SAMPLING_X16 = 0x05 // oversampling x16
};

// filter setting filter[2:0]
enum bme280_filter
{
    FILTER_OFF = 0x00, // filter off
    FILTER_2 = 0x01, // filter coefficient = 2
    FILTER_4 = 0x02, // filter coefficient = 4
    FILTER_8 = 0x03, // filter coefficient = 8

```

```

    FILTER_16 = 0x04 // filter coefficient = 16
};

// standby (inactive) time in ms (used in normal mode), t_sb[2:0]
enum standby_time
{
    STANDBY_0_5 = 0x00, // standby time = 0.5 ms
    STANDBY_62_5 = 0x01, // standby time = 62.5 ms
    STANDBY_125 = 0x02, // standby time = 125 ms
    STANDBY_250 = 0x03, // standby time = 250 ms
    STANDBY_500 = 0x04, // standby time = 500 ms
    STANDBY_1000 = 0x05, // standby time = 1000 ms
    STANDBY_10 = 0x06, // standby time = 10 ms
    STANDBY_20 = 0x07 // standby time = 20 ms
};

struct
{
    uint16_t dig_T1;
    int16_t dig_T2;
    int16_t dig_T3;
    uint16_t dig_P1;
    int16_t dig_P2;
    int16_t dig_P3;
    int16_t dig_P4;
    int16_t dig_P5;
    int16_t dig_P6;
    int16_t dig_P7;
    int16_t dig_P8;
    int16_t dig_P9;

    uint8_t dig_H1;
    uint8_t dig_H3;
    int16_t dig_H4;
    int16_t dig_H5;
    int8_t dig_H6;
} BME280_calib;

// writes 1 byte '_data' to register 'reg_addr'
void BME280_Write(uint8_t reg_addr, uint8_t _data)
{
    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS);
    I2C_Write(reg_addr);
    I2C_Write(_data);
    I2C_Stop();
}

// reads 8 bits from register 'reg_addr'
uint8_t BME280_Read8(uint8_t reg_addr)
{
    uint8_t ret;

    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS);
    I2C_Write(reg_addr);
    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS | 1);
    ret = I2C_Read(0);
    I2C_Stop();

    return ret;
}

// reads 16 bits from register 'reg_addr'

```

```

uint16_t BME280_Read16(uint8_t reg_addr)
{
    union
    {
        uint8_t b[2];
        uint16_t w;
    } ret;

    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS);
    I2C_Write(reg_addr);
    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS | 1);
    ret.b[0] = I2C_Read(1);
    ret.b[1] = I2C_Read(0);
    I2C_Stop();

    return(ret.w);
}

// BME280 sensor configuration function
void BME280_Configure(bme280_mode mode, bme280_sampling T_sampling, bme280_sampling H_sampling,
                    bme280_sampling P_sampling, bme280_filter filter, standby_time standby)
{
    uint8_t ctrl_hum, ctrl_meas, _config;

    _ctrl_hum = H_sampling;
    _config = ((standby << 5) | (filter << 2)) & 0xFC;
    _ctrl_meas = (T_sampling << 5) | (P_sampling << 2) | mode;

    BME280_Write(BME280_REG_CTRLHUM, _ctrl_hum);
    BME280_Write(BME280_REG_CONFIG, _config);
    BME280_Write(BME280_REG_CONTROL, _ctrl_meas);

    // initializes the BME280 sensor, returns 1 if OK and 0 if error
    int1 BME280_begin(bme280_mode mode,
                    bme280_sampling T_sampling = SAMPLING_X1,
                    bme280_sampling H_sampling = SAMPLING_X1,
                    bme280_sampling P_sampling = SAMPLING_X1,
                    bme280_filter filter = FILTER_OFF,
                    standby_time standby = STANDBY_0_5)
    {
        if(BME280_Read8(BME280_REG_CHIPID) != BME280_CHIP_ID)
            return 0;

        // reset the BME280 with soft reset
        BME280_Write(BME280_REG_SOFTRESET, 0xB6);
        delay_ms(100);

        // if NVM data are being copied to image registers, wait 100 ms
        while( (BME280_Read8(BME280_REG_STATUS) & 0x01) == 0x01 )
            delay_ms(100);

        BME280_calib.dig_T1 = BME280_Read16(BME280_REG_DIG_T1);
        BME280_calib.dig_T2 = BME280_Read16(BME280_REG_DIG_T2);
        BME280_calib.dig_T3 = BME280_Read16(BME280_REG_DIG_T3);

        BME280_calib.dig_P1 = BME280_Read16(BME280_REG_DIG_P1);
        BME280_calib.dig_P2 = BME280_Read16(BME280_REG_DIG_P2);
        BME280_calib.dig_P3 = BME280_Read16(BME280_REG_DIG_P3);
        BME280_calib.dig_P4 = BME280_Read16(BME280_REG_DIG_P4);
        BME280_calib.dig_P5 = BME280_Read16(BME280_REG_DIG_P5);
        BME280_calib.dig_P6 = BME280_Read16(BME280_REG_DIG_P6);
        BME280_calib.dig_P7 = BME280_Read16(BME280_REG_DIG_P7);
        BME280_calib.dig_P8 = BME280_Read16(BME280_REG_DIG_P8);
        BME280_calib.dig_P9 = BME280_Read16(BME280_REG_DIG_P9);
    }
}

```

```

BME280_calib.dig_H1 = BME280_Read8(BME280_REG_DIG_H1);
BME280_calib.dig_H2 = BME280_Read16(BME280_REG_DIG_H2);
BME280_calib.dig_H3 = BME280_Read8(BME280_REG_DIG_H3);
BME280_calib.dig_H4 = ((uint16_t)BME280_Read8(BME280_REG_DIG_H4) << 4) | (BME280_Read8(BME280_REG_DIG_H4 + 1) & 0x0F);
if (BME280_calib.dig_H4 & 0x0800) // if BME280_calib.dig_H4 < 0
    BME280_calib.dig_H4 |= 0xF000;
BME280_calib.dig_H5 = ((uint16_t)BME280_Read8(BME280_REG_DIG_H5 + 1) << 4) | (BME280_Read8(BME280_REG_DIG_H5) >> 4);
if (BME280_calib.dig_H5 & 0x0800) // if BME280_calib.dig_H5 < 0
    BME280_calib.dig_H5 |= 0xF000;
BME280_calib.dig_H6 = BME280_Read8(BME280_REG_DIG_H6);

BME280_Configure(mode, T_sampling, H_sampling, P_sampling, filter, standby);

return 1;
}

// takes a new measurement, for forced mode only!
// Returns 1 if ok and 0 if error (sensor is not in sleep mode)
int1 BME280_ForcedMeasurement()
{
    uint8_t ctrl_meas_reg = BME280_Read8(BME280_REG_CONTROL);

    if ( (ctrl_meas_reg & 0x03) != 0x00 )
        return 0; // sensor is not in sleep mode

    // set sensor to forced mode
    BME280_Write(BME280_REG_CONTROL, ctrl_meas_reg | 1);
    // wait for conversion complete
    while (BME280_Read8(BME280_REG_STATUS) & 0x08)
        delay_ms(1);

    return 1;
}

// read (updates) adc_P, adc_T and adc_H from BME280 sensor
void BME280_Update()
{
    union
    {
        uint8_t b[4];
        uint32_t dw;
    } ret;
    ret.b[3] = 0x00;

    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS);
    I2C_Write(BME280_REG_PRESS_MSB);
    I2C_Start();
    I2C_Write(BME280_I2C_ADDRESS | 1);
    ret.b[2] = I2C_Read(1);
    ret.b[1] = I2C_Read(1);
    ret.b[0] = I2C_Read(1);

    adc_P = (ret.dw >> 4) & 0xFFFFF;

    ret.b[2] = I2C_Read(1);
    ret.b[1] = I2C_Read(1);
    ret.b[0] = I2C_Read(1);

    adc_T = (ret.dw >> 4) & 0xFFFFF;

    ret.b[2] = 0x00;
    ret.b[1] = I2C_Read(1);
    ret.b[0] = I2C_Read(0);
    I2C_Stop();

    adc_H = ret.dw & 0xFFFF;
}

```

```

}

// Reads temperature from BME280 sensor.
// Temperature is stored in hundredths C (output value of "5123" equals 51.23 DegC).
// Temperature value is saved to *temp, returns 1 if OK and 0 if error.
int1 BME280_readTemperature(int32_t *temp)
{
    int32_t var1, var2;

    BME280_Update();

    // calculate temperature
    var1 = (((((adc_T / 8) - ((int32_t)BME280_calib.dig_T1 * 2))) *
            ((int32_t)BME280_calib.dig_T2)) / 2048;

    var2 = (((((adc_T / 16) - ((int32_t)BME280_calib.dig_T1)) *
            ((adc_T / 16) - ((int32_t)BME280_calib.dig_T1))) / 4096) *
            ((int32_t)BME280_calib.dig_T3)) / 16384;

    t_fine = var1 + var2;

    *temp = (t_fine * 5 + 128) / 256;

    return 1;
}

// Reads humidity from BME280 sensor.
// Humidity is stored in relative humidity percent in 1024 steps
// (output value of "47445" represents 47445/1024 = 46.333 %RH).
// Humidity value is saved to *humi, returns 1 if OK and 0 if error.
int1 BME280_readHumidity(uint32_t *humi)
{
    int32_t v_x1_u32r;
    uint32_t H;

    v_x1_u32r = (t_fine - ((int32_t)76800));

    v_x1_u32r = (((((adc_H * 16384) - (((int32_t)BME280_calib.dig_H4) * 1048576) - (((int32_t)BME280_calib.dig_H5) * v_x1_u32r)) +
            ((int32_t)16384)) / 32768) * ((((((v_x1_u32r * ((int32_t)BME280_calib.dig_H6)) / 1024) * ((v_x1_u32r *
            ((int32_t)BME280_calib.dig_H3)) / 2048) + ((int32_t)32768))) / 1024) + ((int32_t)2097152)) *
            ((int32_t)BME280_calib.dig_H2) + 8192) / 16384);

    v_x1_u32r = (v_x1_u32r - (((((v_x1_u32r / 32768) * (v_x1_u32r / 32768)) / 128) * ((int32_t)BME280_calib.dig_H1)) / 16));
    v_x1_u32r = (v_x1_u32r < 0 ? 0 : v_x1_u32r);
    v_x1_u32r = (v_x1_u32r > 419430400 ? 419430400 : v_x1_u32r);

    H = (uint32_t)(v_x1_u32r / 4096);
    *humi = H;

    return 1;
}

// Reads pressure from BME280 sensor.
// Pressure is stored in Pa (output value of "96386" equals 96386 Pa = 963.86 hPa).
// Pressure value is saved to *pres, returns 1 if OK and 0 if error.
int1 BME280_readPressure(uint32_t *pres)
{
    int32_t var1, var2;
    uint32_t p;

    // calculate pressure
    var1 = (((int32_t)t_fine) / 2) - (int32_t)64000;
    var2 = (((var1/4) * (var1/4)) / 2048) * ((int32_t)BME280_calib.dig_P6);

    var2 = var2 + ((var1 * ((int32_t)BME280_calib.dig_P5)) * 2);
    var2 = (var2/4) + (((int32_t)BME280_calib.dig_P4) * 65536);

    var1 = (((((int32_t)BME280_calib.dig_P3 * (((var1/4) * (var1/4)) / 8192)) / 8) +
            (((int32_t)BME280_calib.dig_P2) * var1/2)) / 262144;

    var1 = (((32768 + var1)) * ((int32_t)BME280_calib.dig_P1)) / 32768;

    if (var1 == 0)
        return 0; // avoid exception caused by division by zero

    p = (((int32_t)((int32_t)1048576 - adc_P) - (var2 / 4096)) * 3125;

    if (p < 0x80000000)
        p = (p * 2) / ((uint32_t)var1);
    else
        p = (p / (uint32_t)var1) * 2;

    var1 = (((int32_t)BME280_calib.dig_P9) * ((int32_t)((p/8) * (p/8) / 8192))) / 4096;
    var2 = (((int32_t)(p/4)) * ((int32_t)BME280_calib.dig_P8)) / 8192;

    p = (uint32_t)((int32_t)p + ((var1 + var2 + (int32_t)BME280_calib.dig_P7) / 16));

    *pres = p;

    return 1;
}

float BMP280Altitude(float p) //Calculem la altura mitjançant fórmula del Datasheet. 1012.7 Pressio superficial a l' altura de polinyà
{
    altitudbme = 44330*(1-pow(p/1012.7,1/5.255));
    return(altitudbme);
}

```

## 6. Codi de l' estació receptora de dades (Arduino)

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

//Crear els objectes LCD 16 columnas x 2 filas
LiquidCrystal_I2C lcd1(0x27, 16, 2);
LiquidCrystal_I2C lcd2(0x22, 16, 2);
LiquidCrystal_I2C lcd3(0x23, 16, 2);
LiquidCrystal_I2C lcd4(0x26, 16, 2);
LiquidCrystal_I2C lcd5(0x25, 16, 2);

String data;
bool stringComplete = false;
String dia, mes, any, hora, minut, segon, tempBMP, presBMP, altBMP, humBMP, latGPS, lonGPS, altGPS, satGPS, temp1, temp2, temp3, temp4;
double altura;

byte paquet = 1;

void setup()
{
  Serial.begin(9600);
  // Inicialitzar els LCD
  lcd1.init();
  lcd2.init();
  lcd3.init();
  lcd4.init();
  lcd5.init();
  data.reserve(70); //Reservem espai del string per evitar fragmentacions
  //Engegar la llum de fons i mostrar missatge
  lcd1.backlight();
  lcd2.backlight();
  lcd3.backlight();
  lcd4.backlight();
  lcd5.backlight();
  lcd1.print("Inicialitzant...");
  lcd2.print("Inicialitzant...");

  lcd3.print("Inicialitzant...");
  lcd4.print("Inicialitzant...");
  lcd5.print("Inicialitzant...");
  //Al cap de dos segons netegem pantalla i apaguem
  delay(2000);
  lcd1.clear();
  lcd1.noBacklight();
  lcd2.clear();
  lcd2.noBacklight();
  lcd3.clear();
  lcd3.noBacklight();
  lcd4.clear();
  lcd4.noBacklight();
  lcd5.clear();
  lcd5.noBacklight();
}

void loop()
{
  //El següent condicional s' executa quan s' ha acabat la recepció
  if (stringComplete)
  {
    lcd1.clear();
    lcd2.clear();
    lcd3.clear();
    lcd4.clear();
    lcd5.clear();
    lcd1.backlight();
    lcd2.backlight();
    lcd3.backlight();
    lcd4.backlight();
    lcd5.backlight();

    //Primera pantalla
    lcd1.setCursor(4, 0);
```

```
lcd1.print(dia + "/" + mes + "/" + any);
lcd1.setCursor(4, 1);
lcd1.print(hora + ":" + minut + ":" + segon);

//Segona pantalla
lcd2.setCursor(0, 0);
lcd2.print("TA: " + tempBMP + " H: " + humBMP);
lcd2.setCursor(0, 1);
lcd2.print("Pres:" + presBMP);

//Tercera pantalla
lcd3.setCursor(0, 0);
lcd3.print("Alt: " + String(altura));
lcd3.setCursor(0, 1);
lcd3.print("Sat: " + satGPS);

//Quarta pantalla
lcd4.setCursor(0, 0);
lcd4.print("Lat: " + latGPS);
lcd4.setCursor(0, 1);
lcd4.print("Lon: " + lonGPS);

//Cinquena pantalla
lcd5.setCursor(0, 0);
lcd5.print("T1: " + temp1);
lcd5.setCursor(8, 0);
lcd5.print("T2: " + temp2);
lcd5.setCursor(0, 1);
lcd5.print("T3: " + temp3);
lcd5.setCursor(8, 1);
lcd5.print("T4: " + temp4);

stringComplete = false;
}
}

void serialEvent()
{
  while (Serial.available())
  {
    if (paquet != 18)
    {
      data = Serial.readStringUntil(',');
      Serial.read();
    }
    else
    {
      data = Serial.readStringUntil('F');
      Serial.read();
    }
    switch (paquet)
    {
      case 1:
        dia = data;
        paquet += 1;
        break;
      case 2:
        mes = data;
        paquet += 1;
        break;
      case 3:
        any = data;
        paquet += 1;
        break;
      case 4:
        hora = data;
        paquet += 1;
        break;
      case 5:
        minut = data;
        paquet += 1;
```

```
        break;
    case 6:
        segon = data;
        paquet += 1;
        break;
    case 7:
        tempBMP = data;
        paquet += 1;
        break;
    case 8:
        presBMP = data;
        paquet += 1;
        break;
    case 9:
        altBMP = data;
        paquet += 1;
        break;
    case 10:
        humBMP = data;
        paquet += 1;
        break;
    case 11:
        latGPS = data;
        paquet += 1;
        break;
    case 12:
        lonGPS = data;
        paquet += 1;
        break;
    case 13:
        altGPS = data;
        paquet += 1;
        break;
    case 14:
        satGPS = data;

        paquet += 1;
        break;
    case 15:
        temp1 = data;
        paquet += 1;
        break;
    case 16:
        temp2 = data;
        paquet += 1;
        break;
    case 17:
        temp3 = data;
        paquet += 1;
        break;
    case 18:
        temp4 = data;
        altura = (altBMP.toDouble() + altGPS.toDouble()) / 2;
        stringComplete = true;
        paquet = 1;
        break;
    }
}
```