UNIVERSIDADE NOVA DE LISBOA

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Electrotécnica

Sistema e Aquisição de Dados

2º Semestre 2019 / 2020

Estação Meteorológica

05/06/2020

Número de Aluno	Nome	Turno Prático
49547	Gonçalo Sousa Santos	P5
50351	Ricardo Filipe Lagoa Walker	P5
43853	Rafael Oliveira	P5

Índice

1- Introdução

Vai ser criada uma estação meteorológica através de uma plataforma microcontrolada. Neste caso, foi utilizada a placa PICGenios com um PIC16F877A a 4MHz. Para ser possível a obtenção de dados, foi desenvolvida uma aplicação dividida em duas partes com a função de obter, analisar, monitorizar e registar os dados obtidos pelos sensores incorporados na placa PICGenios.

1.1 AQC (Acquisition and Control System)

O sistema de aquisição e controlo será utilizado para obter os valores de temperatura, humidade e velocidade do vento, onde irá enviar estes dados a uma aplicação MR, onde irá ser processado os dados.

A inicialização do sistema será feita através da introdução de uma password por um teclado matricial incorporado e de seguida, estará a obter os valores dos sensores e a transmitir para o ecrã.

Os valores que estão a ser obtidos vão ser enviados para o MR, onde caso, os valores obtidos estejam acima dos limites estipulados, simula um alerta através da ativação de um buzzer, onde em caso real, significaria rajadas de vento ou elevado risco de incêndio.

O MR também pode enviar mensagens para o AQC, onde é pedido as informações que estão a ser obtidas. Como, por exemplo, a informação dos valores atuais, a possível configuração e calibração dos valores. Há a ocorrência da leitura e escrita dos valores da memória EEPROM externa 24C04 (I2C - RC3 e RC4), como irá ser apresentado futuramente.

1.2 MR (Monitorization and Register Application)

A aplicação de monitorização e registo irá ter a função de receber as mensagens enviadas do sistema AQC e apresentá-las ao utilizador. Enquanto mostra os dados ao utilizador, este irá também registá-los em ficheiros XML, para depois serem usados como base de dados.

Como já foi dito no 1.1, o MR tem a função de enviar para o AQC, os valores de risco que o utilizador deseja alterar, como também um pedido para receber os dados atuais dos sensores do AQC.

Por fim, o MR também irá enviar as situações de risco num ficheiro com a data do dia do envio, determinadas anteriormente, para o servidor remoto dado, sendo estes dados em formato JSON, através de HTTP POST.

2- Material

Para a produção deste sistema, foi utilizado vários programas no seu desenvolvimento, sendo estes o MPLAB, especificamente o MPLAB X IDE e MPLAB XC8 C Compiler, o PICSimLab e, por fim, o CuteCom.

Para a composição e desenvolvimento do código para o sistema, foi utilizado o MPLAB XC8 C Compiler, onde depois é testado através do PicSimLab e o CuteCom, sendo estes uma simulação onde tem os sensores que são utilizados na estação meteorológica escolhida.

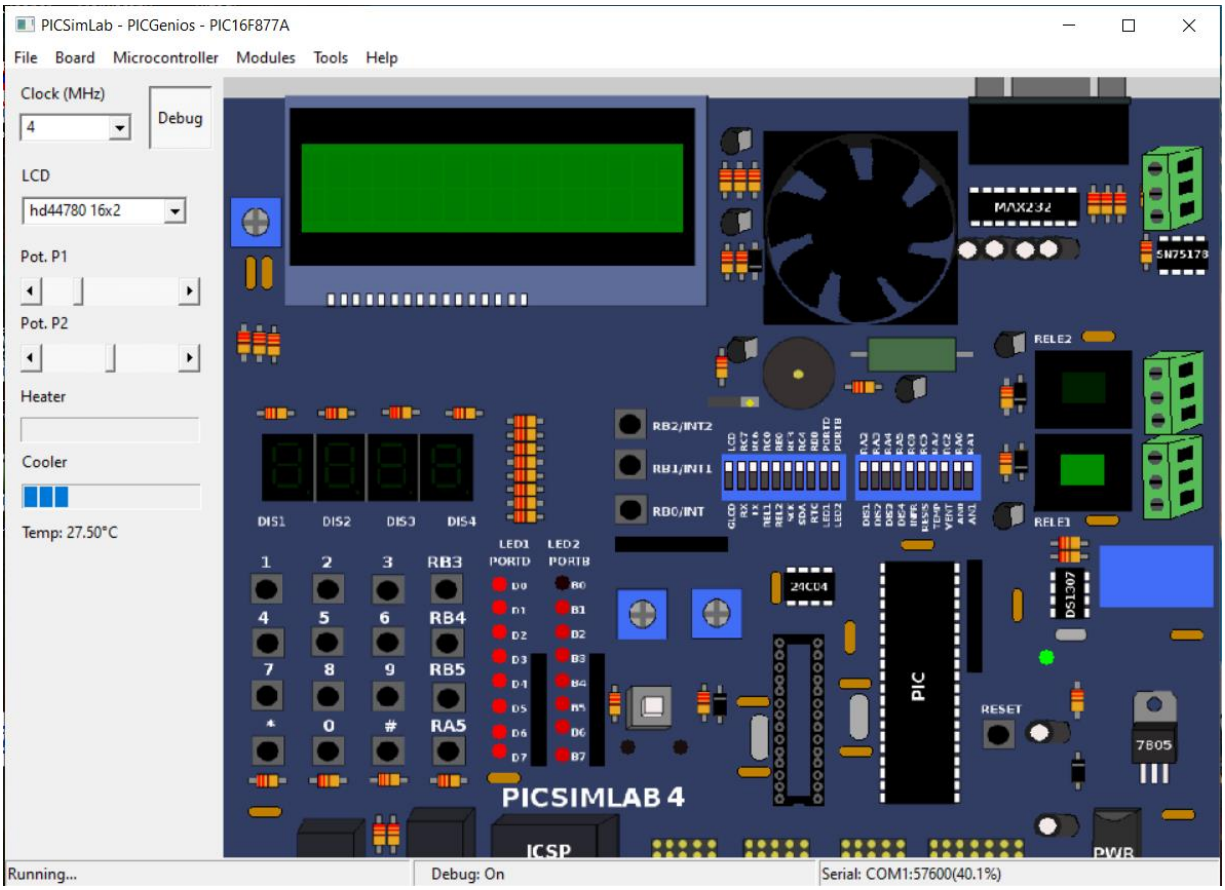

Figura 1 - Imagem do PICSimLab

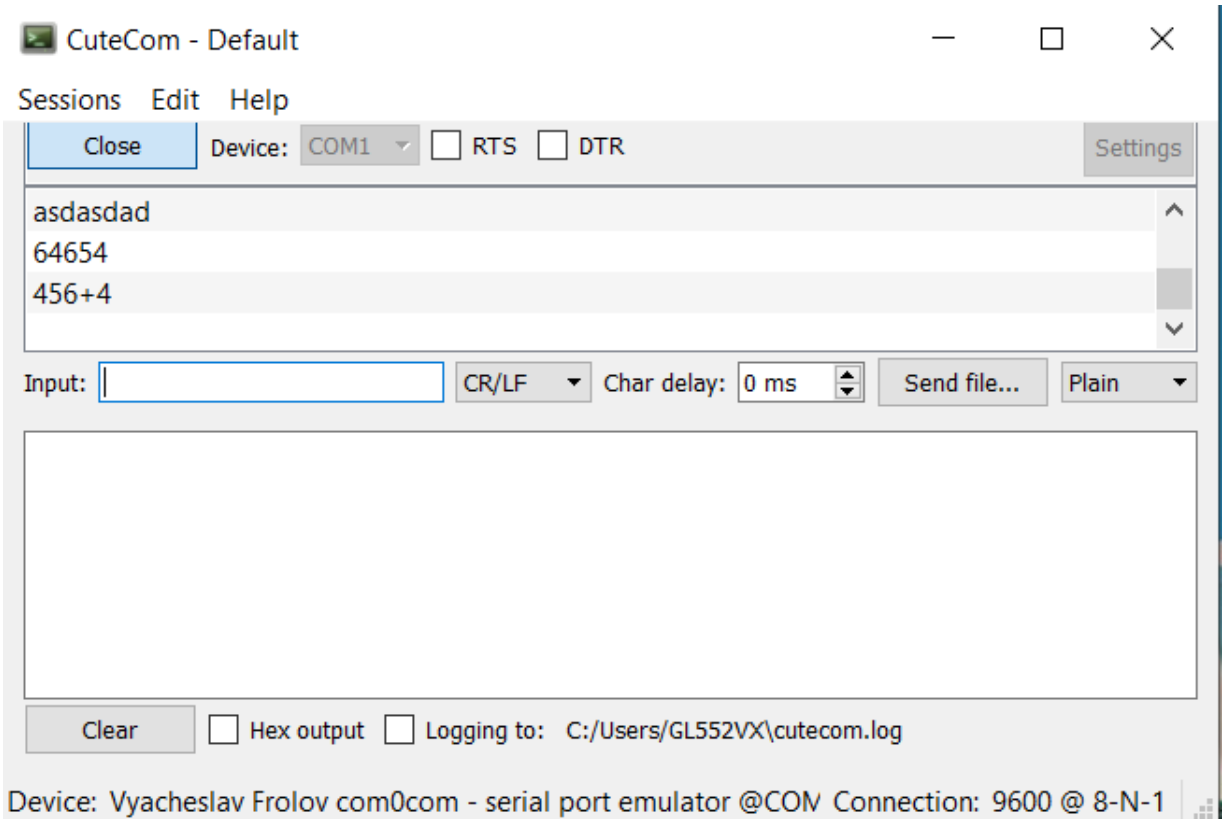

Figura 2 -Display do CuteCom

3 - Código e funcionalidades

3.1- Sensores

O PIC é composto por vários tipos de sensores, onde vai ser possível simular os dados a obter, neste caso, existe uma resistência de calor (RC5) que sempre que é ativado, através da pressão do botão RB3, a temperatura começa a aumentar. Ao carregar novamente, desliga a resistência e a temperatura começa a baixar. Existe também um sensor de vento, onde é simulado através de uma ventoinha integrada no PIC. A velocidade da ventoinha é controlada por PWM (*Pulse - Width Modulation*), onde é regulado através de um potenciómetro (P1). A ventoinha é ativada através do registo CCP1CON, onde é seleccionado o modo PWM. Depois da seleção do modo, é necessário especificar o período através do PR2. O PR2 está diretamente envolvido com o Timer 2 que assim que os valores forem iguais, o Timer 2 vai ser resetado e o CCP1 é posto a 1. Para além disto, é necessário especificar o duty cycle que também está dependente do Timer 2 e vai ser escrito no registo CCPR1L. A ventoinha é posta a girar no sistema através da porta RC2. E por último, existe uma simulação de um sensor de humidade onde é regulado através de outro potenciómetro (P2).

```
void temp_Control(){

    if(PORTBbits.RB3 == 0){
        PORTCbits.RC5 =!PORTCbits.RC5;
        delay(50);
    }

}
```

Figura 3 - Controlo da temperatura

```
void read_ht(){
    //char tmp;
    //tmp=TRISA;
    //TRISA=0x07;
    t = (ADC_read(TEMP)*10)/2;
    //TRISA=tmp;

    h=((float)ADC_read(HUM)/1023)*100;
}
```

Figura 4 - Leitura de valores de temperatura

```
void init_fan(){
    CCP1CON = 0x3f; //bit 3-0: 11xx - PWM mode bit 5-4: PWM LSBs;
    PR2 = 0xff; //Set PWM period
    CCPR1L = 0xff; //Set PWM duty cycle
    T2CKPS1=0; //
    T2CKPS0=0; //prescaler=1
    TMR2ON = 1; //ativa Timer 2
    PORTCbits.RC2=1; // ativa a ventoinha
}
```

Figura 5 - Inicialização da ventoinha

```
void stop_fan(){
    CCP1CON = 0;
    PR2 = 0; //Set PWM period
    CCPR1L = 0; //Set PWM duty cycle
    TMR2ON = 0; //desativa Timer2
    PORTCbits.RC2=0; // desativa a ventoinha
}
```

Figura 6 - Paragem da ventoinha

```
void fan_value(){
    int wind = ADC_read(WIND);
    CCPR1L = wind >> 2; //shift 2 p afetar os 2LSB
    delay(200);
}
```

Figura 7 - Obtenção dos valores da ventoinha

```
void read_fan(){
    int val;
    TMR1 = 0; //TMR1H + TMR1L - Contador
    TMR1ON = 1;
    delay(100);
    TMR1ON = 0;
    v=(unsigned)TMR1*1.5;
}
```

Figura 8 - Contador a considerar a velocidade do vento com correção dos valores

```
void read_ht(){
    //char tmp;
    //tmp=TRISA;
    //TRISA=0x07;
    t = (ADC_read(TEMP)*10)/2;
    //TRISA=tmp;

    h=((float)ADC_read(HUM)/1023)*100;
}
```

Figura 9 - Leitura dos valores de temperatura e humidade

3.2 - USART

A USART vai permitir para existir comunicação série entre as plataformas que estão a ser usadas e que pode ser efetuada de uma maneira síncrona ou assíncrona. Neste caso, foi utilizado o modo assíncrono para não estar dependente do clock. Para tal, foi necessário ligar o porto de série, através do registo do receptor (RCSTAbits.SPEN = 1), e no registo de transmissão, ativar o modo de transmissão (TXSTAbits.TXEN = 1) e seleccionar o modo assíncrono (TXSTAbits.SYNC = 0) e de uma forma contínua (RCSTAbits.CREN = 1), sem ser interrompido o processo de transmissão. Após a inicialização da USART, foi construído algumas funções para a transmissão e leitura de strings.

```
void initUSART(){
     SPBRG=25; // Baud Rate para 4 MHz
     TXSTAbits.BRGH=1; //High Speed Communication

     TXSTAbits.SYNC=0; //Asynchronous data transfer
     RCSTAbits.SPEN=1; //Serial Port enabled

     TXSTAbits.TXEN=1; //Transmit Enabled, also sets bit TXIF
     RCSTAbits.CREN=1; //Enables Continuous receive in Asynchronous mode
}

void transmitChar(char c){

        while(!TXIF);
        TXREG=c;
}

void transmitString(char s[]){

    int i=0;

    while(s[i]!='\0') transmitChar(s[i++]);
}

char receiveChar(){

    if(OERR){ // check for error
        CREN=0;
        CREN=1;
    }

    while(!RCIF){};
    return RCREG;
}

char *receiveString(char *s){


   char ch;
   do{
       ch = receiveChar();
       *s = ch;
       s++;
   }while(ch!='\r' && ch!='\n');
   s--;
   *s = '\0';
   return s;
   //while((ch=receiveChar())!='\0')
}
```

Figura 10 - Funções do USART

3.3 ADC

O ADC vai permitir realizar a conversão dos dados de sinais analógicos para sinais digitais para ser lido pelo sistema e interpretar quais são os valores que estão a ser obtidos. Para tal, é necessário inicializar o módulo ADC através dos seus registos ADCON0 e ADCON1. O ADCON0 = 0x01 vai ligar o módulo. O ADCON1 = 0xc9 vai fazer com que as portas de AN0-5 estejam em modo analógico e que a conversão de clock seja Fosc/4 (isto também é seleccionado devido aos bits do ADCON0<ADCS1:ADCS0> = 00), e o formato de seleção é justificado à direita, em que os bits do ADRESH sejam lidos a 0. Isto faz com que consigamos obter valores até 1023 no PIC.
De seguida à inicialização, é necessário realizar a leitura das portas analógias que se pretende utilizar. Neste trabalho, temos os valores de temperatura a serem obtidos de AN2, os valores de vento a serem obtidos de AN1 e os valores de humidade a serem obtidos do AN0. A função ADC_read vai necessitar da seleção do porto que se pretende ler através ADCON0bits.CHS = canal analógico desejado. O bit GO_nDONE é ativo a 1 para estar a realizar a conversão enquanto existir valores e vão ser colocados numa variável, realizando um shift no ADRESH para o lado esquerdo de 8 e realizando um OR com o ADRESL colocando tudo a 1, retornando depois o valor guardado na variável.

```
void ADC_init(){
    ADCON1=0xf9;
    ADCON0=0x01;
}

int ADC_read(int ch){

    unsigned int i;
    ADCON0bits.CHS = ch;
    ADCON0bits.GO_nDONE = 1;
    delay(10);
    while(ADCON0bits.GO_nDONE == 1){}
    i = ((ADRESH<<8) | (ADRESL & 0xff));
    return i;
}
```

Figura 11 - Funções do ADC

3.4 - Password

A password irá impedir o utilizador de entrar dentro do programa até introduzir uma sequência que é feita ao carregar nos botões, que é anteriormente definida pelo programador, sendo neste caso a password “2020”. Quando é carregado um botão irá acender no Display de sete segmentos o número que está associado a esse botão, isto sendo sequencialmente.

Para carregar nos botões, tem se de ativar a consola dos botões, onde neste caso só se pode ativar a consola por colunas. Com isso, tem-se de dividir o código para as três colunas, o que leva a haver um ciclo onde irá habilitar uma coluna de cada vez (PORTBbits.RB1=0 ou RB2 e RB3, dependendo da coluna) o que depois irá criar uma condição onde verifica se cada botão de cada coluna está a ser carregado(if (!PORTDbits.RD3, RD2, RD1 e RD0), o que depois irá retornar o valor a que está associado cada botão. Após carregar em 4 botões, o controlador irá esperar que o utilizador carregue no “#” ou no “*”, sendo o “#” a introdução da password para a base de dados, e o “*” o reset da password para voltar a introduzi-la.

Após premir os 4 botões necessários para a password, clica-se no “#” para verificar se a password está correta. Caso esteja, os 4 displays irão piscar com a password e entrar no programa. Caso esteja incorreto, o programa envia uma string a confirmar que a password estava incorreta.

```
char keyboardInput(){
    //TECLADO MATRICIAL

    PORTB=0xFF;
    TRISB=0; //PORTB a output
    PORTD=0xFF;
    TRISD=0xFF;//PORTD a input

    while(1){
        PORTBbits.RB1=0; //Habilita coluna 1
        if (!PORTDbits.RD3) {delay(25); return '1';}
        else if (!PORTDbits.RD2) {delay(25); return '4';}
        else if (!PORTDbits.RD1) {delay(25); return '7';}
        else if (!PORTDbits.RD0) {delay(25); return '*';}
        delay(25);
        PORTBbits.RB1=1; //DESHabilita coluna 1

        PORTBbits.RB2=0; //Habilita coluna 2
        if (!PORTDbits.RD3) {delay(25); return '2';}
        else if (!PORTDbits.RD2) {delay(25); return '5';}
        else if (!PORTDbits.RD1) {delay(25); return '8';}
        else if (!PORTDbits.RD0) {delay(25); return '0';}
        delay(10);
        PORTBbits.RB2=1; //DESHabilita coluna 2

        PORTBbits.RB0=0; //Habilita coluna 3
        if (!PORTDbits.RD3) {delay(25); return '3';}
        else if (!PORTDbits.RD2) {delay(25); return '6';}
        else if (!PORTDbits.RD1) {delay(25); return '9';}
        else if (!PORTDbits.RD0) {delay(25); return '#';}
        delay(25);
        PORTBbits.RB0=1; //DESHabilita coluna 3
    }
}
```

Figura 12 - Função keyboardInput

```
unsigned char code7s(char v)
{
  switch(v)
  {
    case '0':
      return 0x3F;
    case '1':
      return 0x06;
    case '2':
      return 0x5B;
    case '3':
      return 0x4F;
    case '4':
      return 0x66;
    case '5':
      return 0x6D;
    case '6':
      return 0x7D;
    case '7':
      return 0x07;
    case '8':
      return 0x7F;
    case '9':
      return 0x6F;
    default:
      return 0;
  }
}
```

Figura 13 - Função code7s

```
void display7s (char code, char j){
    char tmpad, tmpporta, tmptrisa, tmpportd, tmptrisd;
    tmpad=ADCON1;
    tmpporta=PORTA;
    tmptrisa=TRISA;
    tmpportd=PORTD;
    tmptrisd=TRISD;

    ADCON1 =0x06; //configura todos os pinos AD como I/O digital
    PORTA = 0; //resseta todos os pinos do porta
    TRISA = 0; //define porta como saida
    TRISD = 0; //define portd como saida
    PORTD = 255; //seta todos os pinos do portd


    delay(30);
    switch(j)
      {
         case 3:
           PORTA=0x20;
           break;
         case 2:
           PORTA=0x10;
           break;
         case 1:
           PORTA=0x08;
           break;
         case 0:
           PORTA=0x04;
           break;
       }
    PORTD=code7s(code);
    delay(70);

    PORTA=tmpporta;
    TRISA=tmptrisa;
    PORTD=tmpportd;
    TRISD=tmptrisd;
    ADCON1=tmpad;
}
```

Figura 14 - Função display7s

```
int password(){

    char code[5]="----\0";
    char i = 0;
    char erase;
    char tmpporta, tmptrisa,tmpportb, tmptrisb, tmpportd, tmptrisd;
    tmpporta=PORTA;
    tmptrisa=TRISA;
    tmpportb=PORTB;
    tmptrisb=TRISB;
    tmpportd=PORTD;
    tmptrisd=TRISD;

    PORTA=0;
    TRISA=0;

    transmitString("Introduza a password e confirme com a tecla #\r\n");
    for(i=0;i<4;i++){
            code[i]= keyboardInput();
            if (code[i]=='*')return 0;
            if (code[i]=='#'){i-=1; continue;}
            display7s(code[i],i);
        }
    while(keyboardInput()!='#');
    if(!strcmp(code,PASS)){
        transmitString("Password correta. Bem vindo\r\n\n");
        for(i=0;i<4;i++){
        PORTA=0x3C;
        PORTD=0xFF;
        delay(10);
        PORTA=0;
        PORTD=0;
        delay(20);
        }
        PORTA=tmpporta;
        TRISA=tmptrisa;
        PORTB=tmpportb;
        TRISB=tmptrisb;
        PORTD=tmpportd;
        TRISD=tmptrisd;
        pass=1;
        return pass;
    }
```

```
    else{
        transmitString("Password incorreta! Tente outra vez.\r\n\n");
        PORTA=tmpporta;
        TRISA=tmptrisa;
        PORTB=tmpportb;
        TRISB=tmptrisb;
        PORTD=tmpportd;
        TRISD=tmptrisd;
        pass=0;
        return pass;

    }
}
```

Figura 15 - Função Password

3.5 - Main

O main irá inicializar a todos os componentes necessários para a estação iniciar, através da inicialização do interrupt, USART, ADC, a ventoinha, o sensor de temperatura, um contador, entre outros.

Com isto, define-se as portas de entrada e saída da máquina, onde o TRISA0, TRISA1, TRISA2, TRISB3, TRISB0 e TRISC0 são entradas, e o resto do TRISA, TRISB, TRISC e a trotalidade do TRISD e TRISE como portas de entrada.

Após definir as portas de entrada e saída, o programa é iniciado onde fica parado até que a função password acabe. Depois de introduzir a password, o main irá iniciar as várias funções dos sensores da estação, que, por consequente, irá receber os valores de cada sensor. Através destes, o main irá verificar se está acima ou abaixo de um *threshold* e, caso esteja, manda uma mensagem a avisar que ultrapassou esse limite.

```
int main(void) {

   init_Interrupt();
   initUSART();
   //Init_I2C();
   ADC_init();
   init_fan();
   init_t0();
   T1CON=0x02; //inicializa timer1 como contador
   TRISA=0b00000111; //A0, A1, A2 como entrada, os outros como saída
   TRISD=0; //define porta como saida
   TRISE=0;
   TRISBbits.TRISB3 = 1;
   TRISBbits.TRISB0 = 1;
   TRISCbits.TRISC0 = 1;
   TRISCbits.TRISC1 = 0;
   TRISCbits.TRISC2 = 0;
   TRISCbits.TRISC5 = 0;
   TRISDbits.TRISD0 = 0;
   PORTDbits.RD0 = 0;
//char tmp=0;
int i=0;
int oldvento=0;
int oldval=0;
int newval=0;
int risco=0;
//int ventoinha;
limt=40;
limh=15;
limv=50;
init_fan();
delay(250);
```

```
do{
    if (stop == 0){
        if(pass==0) password();

        else{
            temp_Control();
            fan_value();
            read_ht();
            oldvento=v;
            read_fan();
            if (minuto==1){ //MSG1
                send_values();
                minuto=0;
            }
            //Verificar ocorrência de situação de risco
            //t/10 para ajustar
            oldval=newval;
            if((t/10)>=limt || h<=limh || v>=limv ) newval=1;
            else                                    newval=0;
            if(oldval==0 && newval==1){
                transmitString("{\"Evento\":\"Risco Elevado de Incendio\"}\r\n"); //MSG2
                send_values();
                risco=1;
            }
            if(oldval==1 && newval==0)
                risco=0;
            if(risco){//ligar o buzzer e os leds
                PORTD= 0xFF;
                PORTCbits.RC1 = 1;
                for(i = 0 ; i < 20000 ; i++){};
                PORTD= 0;
                PORTCbits.RC1 = 0;
                for(i = 0 ; i < 20000 ; i++){};

            }
              }
              if(RCIF){
                  received=receiveChar();
                  switch(received){
                      case 'v': transmitString("Valores atuais\n");
                                send_values();
                                break;
                      case 'c': received=receiveChar();
                                limt=(int)received;
                                received=receiveChar();
                                limh=(int)received;
                                received=receiveChar();
                                limv=(int)received;
                                transmitString("Calibracao concluida\n");
                                break;
                      case 'x': transmitString("A Encerrar Programa\n");
                                break;
                      case '\n': break;
                      default:  transmitString("Opcao invalida\n");
                  }
              }
          }
      }
    }while(1);
}
```

Figura 16 - Main do AQC

3.6 - MR

A aplicação de MR, irá ser programado em C, e vai ter como função principal a passagem de informação entre o AQC e o servidor remoto, onde vai receber as mensagens enviadas e os dados que sejam pedidos do sistema de AQC e apresentar os dados recebidos. Estes dados terão de ser registados em ficheiro XML, sendo usados como base de dados.

Para isso, irá ser apresentado ao detalhe cada uma das partes constituintes da aplicação de MR.

Para haver a comunicação entre o servidor e o AQC, criou-se uma função onde estabelece esta ligação em série, sendo esta o start_serial. Para tal, é criado um ficheiro que irá ser usado para verificar se a comunicação entre os dois foi sucedida. A verificação é feita através de vários passos, desde verificar se o ficheiro foi criado de forma correta, a verificar o estado da comunicação através do GetCommState, ao introduzir parâmetros da comunicação, através do SetCommState, ou, por fim, verificar se as definições de timeout foram bem definidas, através do SetCommTimeouts.

Caso tenha sucedido, este irá iniciar o close_serial e ter um sai com o valor 1, para começar a transferência de dados.

```
void start_serial(){
        // Iniciar a comunicação série na porta COM_X
    fprintf(stderr, "Opening serial port...");
    hSerial = CreateFile(
                "\\\\.\\COM2", GENERIC_READ|GENERIC_WRITE, 0, NULL,
                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL );
    if (hSerial == INVALID_HANDLE_VALUE)
    {
            fprintf(stderr, "Error\n");
            exit(1);
    }
    else fprintf(stderr, "OK\n");

    // Set device parameters (9600 baud, 1 start bit,
    // 1 stop bit, no parity)
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    if (GetCommState(hSerial, &dcbSerialParams) == 0)
    {
        fprintf(stderr, "Error getting device state\n");
        CloseHandle(hSerial);
       exit(1);
    }

    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity = NOPARITY;
    if(SetCommState(hSerial, &dcbSerialParams) == 0)
    {
        fprintf(stderr, "Error setting device parameters\n");
        CloseHandle(hSerial);
        exit(1);
    }

    // Set COM port timeout settings
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;
    if(SetCommTimeouts(hSerial, &timeouts) == 0)
    {
        fprintf(stderr, "Error setting timeouts\n");
        CloseHandle(hSerial);
        exit(1);
    }
}
```

Figura 17 - Inicialização da comunicação em série

O close_serial vai ter o efeito contrário da função start_serial, ou seja, assim que a transmissão de dados esteja finalizada, é necessário fechar a porta série da aplicação e para tal vai ser efeito através do fecho da handle definida, que caso esteja a zero, vai dar erro e não vai fechar a porta série.

```
void close_serial(){
    // Close serial port
    fprintf(stderr, "Closing serial port...");
    if (CloseHandle(hSerial) == 0)
    {
        fprintf(stderr, "Error\n");
        exit(1);
    }
    fprintf(stderr, "OK\n");
}
```

Figura 18 - Encerramento da comunicação em série

Com a função write_byte, vai ser possível escrever para a aplicação de MR através da porta série, utilizado a função WriteFile, que necessita de um handle para o ficheiro a ser utilizado, um apontador para o buffer que contém os bytes que irão ser escritos, a quantidade de bytes que serão escritos, e um apontador para a variável que recebe o número de caracteres escritos ao utilizar um handle síncrono, por último como não é utilizado a estrutura de OVERLAPPED é colocado a NULL. Em caso de erro, emite uma mensagem de erro, fecha a porta série e termina o programa.

```
void write_byte(char *byte2send){
    DWORD bytes_written, total_bytes_written = 0;
    if(!WriteFile(hSerial, byte2send, 1, &bytes_written, NULL))
    {
        fprintf(stderr, "Error\n");
        CloseHandle(hSerial);
        exit(1);
    }
    //fprintf(stdout, "%c\n", byte2send);
}
```

Figura 19 - Escrita do Byte

Ao inicializar o MR, irá ser apresentado um menu com opções de buscar os valores atuais do AQC ou alterar os limites a que os valores recebidos possam chegar.

```
void menu(){
    printf("\n\n                MENU\n\n");
    printf("v - Verificar valores atuais\n");
    printf("c - Alterar valores limite de situacao de risco\n");
    printf("x - Encerrar programa\n\n");
    printf("Escolha uma opcao:");
}
```

Figura 20 - Função Menu do MR

O main da aplicação de MR vai ser composto pela interação do menu desenvolvido, dependendo do que o utilizador pressione. Neste caso, teremos a letra ‘v’ para a verificação de valores, em que vai imprimir no ecrã os valores que estão a ser obtidos no PICSimLab; a letra ‘c’ vai alterar os valores de situação de risco, de acordo com intervalos pré-definidos, fazendo com que os limites de risco sejam alterados, recebendo a mensagem de alerta depois de ultrapassar estes novos valores; a letra ‘x’ vai apenas encerrar o programa. Qualquer outra letra também termina o programa.

```
strcpy(message_buffer, "");
printf("Para aceder ao menu, pressione uma tecla\n");
do{

    if(kbhit()){
        getch();
        menu();
        bytes_to_send[0]=getche();
        printf("\n\n");
        switch (bytes_to_send[0])
        {
        case 'v':
            write_byte(bytes_to_send);
            break;

        case 'c':
            write_byte(bytes_to_send);
            printf("Introduza o novo limite de temperatura (0-100C):");
            scanf("%d", &t);
            if(t<0 || t>100){
                printf("Valor invalido!!!\n");
                break;
            }
            printf("Introduza o novo limite de humidade(0-100%%):");
            scanf("%d", &h);
            if(h<0 || h>100){
                printf("Valor invalido!!!\n");
                break;
            }
            printf("Introduza o novo limite de velocidade do vento (0-105 Km/h):");
            scanf("%d", &v);
            if(v<0 || v>105){
                printf("Valor invalido!!!\n");
                break;
            }
            bytes_to_send[0]=(char)t;
            write_byte(bytes_to_send);
             bytes_to_send[0]=(char)h;
            write_byte(bytes_to_send);
            bytes_to_send[0]=(char)v;
            write_byte(bytes_to_send);
            break;
        case 'x':
            exit=1;
            break;
        default:
            break;
        }
    }
```

Figura 21 - Início do Main

Caso não seja detectado um toque no teclado, ele vai começar a ler os valores do PIC, dependendo do valor de uma variável chamada read. Caso read seja igual a 0, vai ler o ficheiro dos valores através da comunicação da porta série, enquanto o número de caracteres lido for diferente de 1. Estes valores vão ter de ser guardados em formato XML, portanto cada caracter vai ser passado para um message_buffer enquanto não chega a um \n ou \r. Se chegar a um \n ou \r, vai imprimir para o ecrã o que está no buffer. Porém é preciso analisar os caracteres para identificar certos pontos, para poder guardar os dados no formato desejado. Os caracteres utilizados para identificar o tipo de mensagem a receber será o 'E' que corresponde à mensagem de alerta de valores acima dos estipulados e o 'T' para começar a guardar os valores de cada um dos sensores na seguinte ordem: temperatura, humidade e velocidade do vento. Isto vai estar sempre a ser colocado no xml_message, vindo do message_buffer. No final da obtenção de valores, o message_buffer vai ser limpo e o read é colocado a 1, indicando que a leitura foi bem-sucedida.

```
        else{
            read=0;
            if(read==0){
                ReadFile(hSerial, &c, 1, &bytes_read, NULL);
                if (bytes_read != 1) continue;

                if (c != '\n' && c != '\r')
                    strncat(message_buffer, &c, 1);
                else{
                    fprintf(stdout, "%s\r\n", message_buffer);
                    if(message_buffer[0]=='{'){
                        //Envio para o Server
                        init_wsock();
                        create_socket();
                        connect2server();
                        send2server();
                        close_wsock();
                        //XML
                        tempo=time(NULL);
                        if(message_buffer[2]=='E'){
                        strcpy(xml_message,"<Msg2>\n\t<Hora>\n\t\t");
                        strcat(xml_message,ctime(&tempo));
                        strcat(xml_message,"\t<\\Hora>\n\t<Evento>\n\t\tRisco Elevado de Incendio\n\t<\\Evento>\n<\\Msg2>\n\n");
                        fputs(xml_message, fp);
                        strcpy(xml_message, "");
                        }
                        if(message_buffer[2]=='T'){
                            tempo=time(NULL);
                            strcpy(xml_message,"<Msg1>\n\t<Hora>\n\t\t");
                            strcat(xml_message,ctime(&tempo));
                            strcat(xml_message,"\t<\\Hora>\n\t<Valores>\n\t\t<Temperatura>");
                            i=3;
                            while(message_buffer[i++]!=':'){}
                            for(j=0;message_buffer[++i]!='"';j++){
                                buff[j]=message_buffer[i];
                            }
                            buff[j]='\0';
                            strcat(xml_message,buff);
                            strcat(xml_message,"<\\Temperatura>\n\t\t<Humidade>");
                            while(message_buffer[i++]!=':'){}
                            for(j=0;message_buffer[++i]!='"';j++){
                                buff[j]=message_buffer[i];
                            }
                            buff[j]='\0';
                            strcat(xml_message,buff);
                            strcat(xml_message,"<\\Humidade>\n\t\t<Velocidade Vento>");
                            while(message_buffer[i++]!=':'){}
                            for(j=0;message_buffer[++i]!='"';j++){
                                buff[j]=message_buffer[i];
                            }
                            buff[j]='\0';
                            strcat(xml_message,buff);
                            strcat(xml_message,"<\\Velocidade Vento>\n\t<\\Valores>\n<\\Msg1>\n\n");
                            fputs(xml_message, fp);
                            strcpy(xml_message, "");
                            strcpy(buff, "");
                            i=0;
                        }
                    }
                    strcpy(message_buffer, ""); //limpar o message_buffer
                    read=1;
                }
            }
        }
    }while(exit==0);

    close_serial();
    // exit normally
    return 0;
}
```

Figura 22 - Parte do Main: Envio de Mensagens para o Servidor

4 - Conclusões

Para a criação do sistema com o objetivo de adquirir, analisar, monitorizar e registar dados de uma estação meteorológica, foi necessário recorrer a várias fontes para aprender a programar certas partes do sistema, principalmente através dos datasheets dados. Quanto às suas funções, foi possível suceder todos os requerimentos dados.

5 – Anexo

5 .1–PIC (main.c)

```
#include <xc.h>
#include <htc.h>
#include <stdio.h>
#include <string.h>

#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit
#pragma config PWRTE = OFF // Power-up Timer Enable bit
#pragma config BOREN = OFF // Brown-out Reset Enable bit
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming
Enable bit
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit
#pragma config WRT = OFF // Flash Program Memory Write Enable bits
#pragma config CP = OFF // Flash Program Memory Code Protection bit

#define SDA      RC4       // Data pin for i2c
#define SCK      RC3       // Clock pin for i2c
#define SDA_DIR    TRISC4      // Data pin direction
#define SCK_DIR    TRISC3      // Clock pin direction
#define I2C_SPEED  100       // kbps

#define EEPROM_Address_R 0xA1
#define EEPROM_Address_W 0xA0

#define TEMP 0x02
#define HUM 0x01
```

```
#define WIND 0x00
#define PASS "2020\0"

char string[20];
char received;
int stop;
int pass;
int countt0=0;
int sec=0;
int minuto=0;
int t, h, v=0; //armazenar os valores lidos pelo ADC
int limt, limh, limv, rajada; //limites para ocorrência de situação de risco

void delay(int d){
        int i = 0;
        int j = 0;
        for (i = 0; i<d ; i++){
                for (j = 0; j < 300 ; j ++){}
        }
}

void inttoStr(int x, char *string){

        sprintf(string, "%d", x);
}

/************************************************************Funções
USART*****************************************************/
void initUSART(){
    SPBRG=25; // Baud Rate para 4 MHz
    TXSTAbits.BRGH=1; //High Speed Communication

    TXSTAbits.SYNC=0; //Asynchronous data transfer
    RCSTAbits.SPEN=1; //Serial Port enabled

    TXSTAbits.TXEN=1; //Transmit Enabled, also sets bit TXIF
    RCSTAbits.CREN=1; //Enables Continuous receive in Asynchronous mode
}

void transmitChar(char c){

                while(!TXIF);
                TXREG=c;
}

void transmitString(char s[]){

        int i=0;

        while(s[i]!='\0') transmitChar(s[i++]);
```

```
}

char receiveChar(){

   if(OERR){ // check for error
      CREN=0;
      CREN=1;
   }

         while(!RCIF){};
   return RCREG;
}

/*char *receiveString(char *s){


         char ch;
         do{
      ch = receiveChar();
      *s = ch;
      s++;
   }while(ch!='\r' && ch!='\n');
         s--;
         *s = '\0';
   return s;
         //while((ch=receiveChar())!='\0')
}*/
/*****************************************************************************
*********************************************/

/*************************************************************Funções
ADC*****************************************************/
void ADC_init(){
   ADCON1=0xc9;  //11001001
   ADCON0=0x01;  //ADON=1: A/D powered up
}

int ADC_read(int ch){

   unsigned int i;
   ADCON0bits.CHS = ch;
   ADCON0bits.GO_nDONE = 1;
   delay(10);
   while(ADCON0bits.GO_nDONE == 1){}
   i = ((ADRESH<<8) | (ADRESL & 0xff)); // Shift left ADRESH.
   return i;
}

/*****************************************************************************
*********************************************/
```

```
/**********************************************************Funções PASS*****************************************************/

char keyboardInput(){
   //TECLADO MATRICIAL

   PORTB=0xFF;
   TRISB=0; //PORTB a output
   PORTD=0xFF;
   TRISD=0xFF;//PORTD a input

   while(1){
      PORTBbits.RB1=0; //Habilita coluna 1
      if (!PORTDbits.RD3) {delay(25); return '1';}
      else if (!PORTDbits.RD2) {delay(25); return '4';}
      else if (!PORTDbits.RD1) {delay(25); return '7';}
      else if (!PORTDbits.RD0) {delay(25); return '*';}
      delay(10);
      PORTBbits.RB1=1; //DESHabilita coluna 1

      PORTBbits.RB2=0; //Habilita coluna 2
      if (!PORTDbits.RD3) {delay(25); return '2';}
      else if (!PORTDbits.RD2) {delay(25); return '5';}
      else if (!PORTDbits.RD1) {delay(25); return '8';}
      else if (!PORTDbits.RD0) {delay(25); return '0';}
      delay(10);
      PORTBbits.RB2=1; //DESHabilita coluna 2

      PORTBbits.RB0=0; //Habilita coluna 3
      if (!PORTDbits.RD3) {delay(25); return '3';}
      else if (!PORTDbits.RD2) {delay(25); return '6';}
      else if (!PORTDbits.RD1) {delay(25); return '9';}
      else if (!PORTDbits.RD0) {delay(25); return '#';}
      delay(10);
      PORTBbits.RB0=1; //DESHabilita coluna 3
   }
}

unsigned char code7s(char v)
{
 switch(v)
 {
   case '0':
     return 0x3F;
   case '1':
     return 0x06;
   case '2':
     return 0x5B;
```

```
    case '3':
      return 0x4F;
    case '4':
      return 0x66;
    case '5':
      return 0x6D;
    case '6':
      return 0x7D;
    case '7':
      return 0x07;
    case '8':
      return 0x7F;
    case '9':
      return 0x6F;
    default:
      return 0;
  }
}

void display7s (char code, char j){
    char tmpad, tmpporta, tmptrisa, tmpportd, tmptrisd;
    tmpad=ADCON1;
    tmpporta=PORTA;
    tmptrisa=TRISA;
    tmpportd=PORTD;
    tmptrisd=TRISD;

    ADCON1 =0x06; //configura todos os pinos AD como I/O digital
    PORTA = 0; //resseta todos os pinos do porta
    TRISA = 0; //define porta como saida
    TRISD = 0; //define portd como saida
    PORTD = 255; //seta todos os pinos do portd


    delay(30);
    switch(j)
      {
        case 3:
          PORTA=0x20;
          break;
        case 2:
          PORTA=0x10;
          break;
        case 1:
          PORTA=0x08;
          break;
        case 0:
          PORTA=0x04;
          break;
       }
```

```
    PORTD=code7s(code);
    delay(70);

    PORTA=tmpporta;
    TRISA=tmptrisa;
    PORTD=tmpportd;
    TRISD=tmptrisd;
    ADCON1=tmpad;
}

int password(){

    char code[5]="----\0";
    char i = 0;
    char erase;
    char tmpporta, tmptrisa,tmpportb, tmptrisb, tmpportd, tmptrisd;
    tmpporta=PORTA;
    tmptrisa=TRISA;
    tmpportb=PORTB;
    tmptrisb=TRISB;
    tmpportd=PORTD;
    tmptrisd=TRISD;

    PORTA=0;
    TRISA=0;

    transmitString("Introduza a password e confirme com a tecla #\r\n");
    for(i=0;i<4;i++){
          code[i]= keyboardInput();
          if (code[i]=='*')return 0;
          if (code[i]=='#'){i-=1; continue;}
          display7s(code[i],i);
       }
    while(keyboardInput()!='#');
    if(!strcmp(code,PASS)){
       transmitString("Password correta. Bem vindo\r\n\n");
       for(i=0;i<4;i++){
       PORTA=0x3C;
       PORTD=0xFF;
       delay(10);
       PORTA=0;
       PORTD=0;
       delay(20);
       }
       PORTA=tmpporta;
       TRISA=tmptrisa;
       PORTB=tmpportb;
       TRISB=tmptrisb;
       PORTD=tmpportd;
       TRISD=tmptrisd;
```

```
      pass=1;
      return pass;
   }
   else{
      transmitString("Password incorreta! Tente outra vez.\r\n\n");
      PORTA=tmpporta;
      TRISA=tmptrisa;
      PORTB=tmpportb;
      TRISB=tmptrisb;
      PORTD=tmpportd;
      TRISD=tmptrisd;
      pass=0;
      return pass;

   }
}

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

/*************************************************************I2C***************
*******************************************/
/*
void Init_I2C(){ //Setup I2C in Master Mode

   SDA_DIR = 1; //input
   SCK_DIR = 1; //input
   SSPADD = 4;  //When SSP configured in Master mode, lower seven bits act as the baud
rate generator reload value.
   SSPSTAT = 0x80; //Slew Rate control is disabled
   SSPCON = 0x28; //select and enable I2C in master mode
   SSPCON2 = 0x00;
}

void I2C_wait(){ //Garantir que o estado anterior foi concluido
   while ((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}

void I2C_start(){ //Start Condition
   I2C_wait();
   SEN = 1; //send start bit

}

void I2C_stop(){ //Stop Condition
   I2C_wait();
   PEN = 1; //send stop bit

}
```

```
void I2C_restart(){ //Restart Condition
   I2C_wait();
   RSEN = 1; //send start bit

}

void I2C_ACK(){
   I2C_wait();
   ACKDT = 0; // 0-> ACK, 1->NACK
   ACKEN = 1; // Envia o sinal

}

void I2C_NACK(){
   I2C_wait();
   ACKDT = 1; // 0-> ACK, 1->NACK
   ACKEN = 1; // Envia o sinal
}

char I2C_sendByte(unsigned char byte){
   I2C_wait();
   SSPBUF = byte;
}

char I2C_readByte(){
   int temp;
   I2C_wait();
   RCEN = 1;
   I2C_wait();
   temp=SSPBUF;
   I2C_wait();
   SSPIF = 0;
   return SSPBUF;
}

void EEPROM_writeByte(unsigned int x, unsigned char byte){
   I2C_start();

   while (I2C_sendByte(EEPROM_Address_W))
      I2C_restart();

   I2C_sendByte(x>>8);
   I2C_sendByte((unsigned char)x);
   I2C_sendByte(byte);
   I2C_stop();

}

void EEPROM_writeString(unsigned int x, char *byte, unsigned int len){
   I2C_start();
```

```
    while (I2C_sendByte(EEPROM_Address_W))
      I2C_restart();

    I2C_sendByte(x>>8);
    I2C_sendByte((unsigned char)x);
    for (int i = 0; i < len; i++)
      I2C_sendByte(byte[i]);
    I2C_stop();
}

unsigned char EEPROM_readByte(unsigned int x){
  unsigned char byte;
  I2C_start();

  // Wait Until EEPROM Is IDLE
  while(I2C_sendByte(EEPROM_Address_W))
    I2C_restart();

  I2C_sendByte(x>>8);
  I2C_sendByte((unsigned char)x);
  I2C_restart();

  I2C_sendByte(EEPROM_Address_R);
  byte = I2C_readByte();
  I2C_NACK();
  I2C_stop();

  return byte;

}

void EEPROM_readString(unsigned int x, char *byte, unsigned int len){
    I2C_start();

  // Wait Until EEPROM Is IDLE
  while(I2C_sendByte(EEPROM_Address_W))
    I2C_restart();

  I2C_sendByte(x>>8);
  I2C_sendByte((unsigned char)x);
  I2C_restart();
  I2C_sendByte(EEPROM_Address_R);
  for(unsigned int i=0; i<len; i++)
  {
    byte[i] = I2C_readByte();
    I2C_ACK();
  }
  I2C_stop();
}*/
```

```
/*************************************************************************************
*********************************************/

/*************************************************************Funções
Timer0**************************************************/
void init_t0(){
   TMR0=0x06; //para contar até 250;
   OPTION_REG= 0x01;// Set Timer 0 as timer (TOCS=0); Prescale 4;
}

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


/*************************************************************Funções
valores*************************************************************/

void temp_Control(){

   if(PORTBbits.RB3 == 0){
      PORTCbits.RC5 =!PORTCbits.RC5;
      delay(20);
   }

}

void read_ht(){
   //char tmp;
   //tmp=TRISA;
   //TRISA=0x07;
   t = (ADC_read(TEMP)*10)/2;
   //TRISA=tmp;

   h=((float)ADC_read(HUM)/1023)*100;
}

void temp_Values(){

   inttoStr(t, string);
   transmitString("\"Temperatura\":\"");
   transmitChar(string[0]);
   transmitChar(string[1]);
   transmitChar(',');
   transmitChar(string[2]);
   transmitString("\"");

}

void hum_Values(){
```

```
   inttoStr(h, string);
   transmitString("\"Humidade\":\"");
   transmitString(string);
   transmitString("%\"");
}

void wind_Values(){

   inttoStr(v, string);
   transmitString("\"Velocidade do Vento\":\"");
   transmitString(string);
   transmitString("\"");
}

/*void P1_Values(){
   int val;
   val=ADC_read(WIND);
   inttoStr(val, string);
   transmitString("P1: ");
   transmitString(string);
   transmitString("\r\n");
}*/

void send_values(){
   transmitString("{");
   temp_Values();
   transmitString(", ");
   hum_Values();
   transmitString(", ");
   wind_Values();
   transmitString("}");
   transmitString("\r\n");
}
/*****************************************************************************
*********************************************/


/*************************************************************Funções Ventoinha &
PWM*****************************************************/

/*8.3.3 SETUP FOR PWM OPERATION
The following steps should be taken when configuring
the CCP module for PWM operation:
1. Set the PWM period by writing to the PR2 register.
2. Set the PWM duty cycle by writing to the
CCPR1L register and CCP1CON<5:4> bits.
3. Make the CCP1 pin an output by clearing the
TRISC<2> bit.
4. Set the TMR2 prescale value and enable Timer2
```

```
by writing to T2CON.
5. Configure the CCP1 module for PWM operation.*/

void init_fan(){
   CCP1CON = 0x3f; //bit 3-0: 11xx - PWM mode bit 5-4: PWM LSBs;
   PR2 = 0xff; //Set PWM period
   CCPR1L = 0xff; //Set PWM duty cycle
   T2CKPS1=0; //
   T2CKPS0=0; //prescaler=1
   TMR2ON = 1; //ativa Timer 2
   PORTCbits.RC2=1; // ativa a ventoinha
}

void stop_fan(){
   CCP1CON = 0;
   PR2 = 0; //Set PWM period
   CCPR1L = 0; //Set PWM duty cycle
   TMR2ON = 0; //desativa Timer2
   PORTCbits.RC2=0; // desativa a ventoinha
}

void fan_value(){
   int wind = ADC_read(WIND);
   CCPR1L = wind >> 2; //shift 2 p afetar os 2LSB
   delay(50);
}

void read_fan(){
   TMR1 = 0; //TMR1H + TMR1L - Contador
   TMR1ON = 1;
   delay(50); //periodo de contagem
   TMR1ON = 0;
   v=(unsigned)TMR1/2; // /2 para ajustar a gama de valores 0-105
}

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

/*************************************************************Interrupts**********
*****************************************/

void init_Interrupt(){

   GIE = 1;
   TMR0IE = 1;
   INTCONbits.PEIE = 0;
   INTE = 1;
   RCIE=1;

}
```

```
void __interrupt() ISR(){

   if(INTF == 1){

      stop=~stop;
      if(stop) stop_fan();
      else     init_fan();
      INTF = 0;
      delay(50);
   }

   if(T0IF ==1){ //Timer 0 Overflow

      countt0++;

      if(countt0==1000){ //1us*Prescale*timerInterval=1,000e-3 *1000=1s
         sec+=1;
         countt0=0;
         TMR0=0x06;
      }
      if(sec==60){
         minuto=1;
         sec=0;
         //TMR0=0x06
      }
      T0IF=0;
   }

}

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


int main(void) {

  init_Interrupt();
  initUSART();
  //Init_I2C();
  ADC_init();
  init_fan();
  init_t0();
  T1CON=0x02; //inicializa timer1 como contador
  TRISA=0b00000111; //A0, A1, A2 como entrada, os outros como saída
  TRISD=0; //define todos os portd como saida
  TRISE=0; // PORTDs como saída
  TRISBbits.TRISB3 = 1; //B3 como entrada
  TRISBbits.TRISB0 = 1; //B0 como entrada
```

```
TRISCbits.TRISC0 = 1; //C0 como entrada
TRISCbits.TRISC1 = 0; //C1 como saída
TRISCbits.TRISC2 = 0; //C2 como saída
TRISCbits.TRISC5 = 0; //C5 como saída
TRISDbits.TRISD0 = 0; //D0 como saída
PORTDbits.RD0 = 0; //D0 inicicializado a 0


//char tmp=0;
int i=0;
int oldvento=0;
int oldval=0;
int newval=0;
int risco=0;
//int ventoinha;
limt=40;
limh=15;
limv=50;
init_fan();
delay(250);

/*I2C_start();
I2C_sendByte(0xA0);
for(i=0;i<10;i++){

    I2C_sendByte(i);
    I2C_sendByte(0x02);
}
I2C_stop();*/
do{
    if (stop == 0){
        if(pass==0) password();

        else{
            temp_Control();
            fan_value();
            read_ht();
            oldvento=v;
            read_fan();
            if (minuto==1){ //MSG1
                send_values();
                minuto=0;
            }
            //Verificar ocorrência de situação de risco
            //t/10 para ajustar
            oldval=newval;
            if((t/10)>=limt || h<=limh || v>=limv ) newval=1;
            else                                    newval=0;
            if(oldval==0 && newval==1){
                transmitString("{\"Evento\":\"Risco Elevado de Incendio\"}\r\n"); //MSG2
```

```
            risco=1;
        }
        if(oldval==1 && newval==0)
            risco=0;
        if(risco){//ligar o buzzer e os leds
            PORTD= 0xFF;
            PORTCbits.RC1 = 1;
            for(i = 0 ; i < 20000 ; i++){};
            PORTD= 0;
            PORTCbits.RC1 = 0;
            for(i = 0 ; i < 20000 ; i++){};

        }
        if(RCIF){
            received=receiveChar();
            switch(received){
                case 'v': transmitString("Valores atuais\n");
                        send_values();
                        break;
                case 'c': received=receiveChar();
                        limt=(int)received;
                        received=receiveChar();
                        limh=(int)received;
                        received=receiveChar();
                        limv=(int)received;
                        transmitString("Calibracao concluida\n");
                        break;
                case 'x': transmitString("A Encerrar Programa\n");
                        break;
                case '\n': break;
                default:  transmitString("Opcao invalida\n");
            }
        }
      }
    }
  }while(1);
}
```

5.2– Aplicação (MR.c)

```
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>

// Variaveis e estruturas
   HANDLE hSerial;
   DCB dcbSerialParams = {0};
   COMMTIMEOUTS timeouts = {0};

void start_serial(){
      // Iniciar a comunicação série na porta COM_X
   fprintf(stderr, "Opening serial port...");
   hSerial = CreateFile(
            "\\\\.\\COM2", GENERIC_READ|GENERIC_WRITE, 0, NULL,
            OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL );
   if (hSerial == INVALID_HANDLE_VALUE)
   {
         fprintf(stderr, "Error\n");
         exit(1);
   }
   else fprintf(stderr, "OK\n");

   // Set device parameters (9600 baud, 1 start bit,
   // 1 stop bit, no parity)
   dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
   if (GetCommState(hSerial, &dcbSerialParams) == 0)
   {
      fprintf(stderr, "Error getting device state\n");
      CloseHandle(hSerial);
     exit(1);
   }

   dcbSerialParams.BaudRate = CBR_9600;
   dcbSerialParams.ByteSize = 8;
   dcbSerialParams.StopBits = ONESTOPBIT;
   dcbSerialParams.Parity = NOPARITY;
   if(SetCommState(hSerial, &dcbSerialParams) == 0)
   {
      fprintf(stderr, "Error setting device parameters\n");
      CloseHandle(hSerial);
      exit(1);
   }
```

```
    // Set COM port timeout settings
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;
    if(SetCommTimeouts(hSerial, &timeouts) == 0)
    {
        fprintf(stderr, "Error setting timeouts\n");
        CloseHandle(hSerial);
        exit(1);
    }
}

void close_serial(){
    // Close serial port
    fprintf(stderr, "Closing serial port...");
    if (CloseHandle(hSerial) == 0)
    {
        fprintf(stderr, "Error\n");
        exit(1);
    }
    fprintf(stderr, "OK\n");
}

void write_byte(char *byte2send){
    DWORD bytes_written, total_bytes_written = 0;
    if(!WriteFile(hSerial, byte2send, 1, &bytes_written, NULL))
    {
        fprintf(stderr, "Error\n");
        CloseHandle(hSerial);
        exit(1);
    }
    //fprintf(stdout, "%c\n", byte2send);
}

void menu(){
    printf("\n\n              MENU\n\n");
    printf("v - Verificar valores atuais\n");
    printf("c - Alterar valores limite de situacao de risco\n");
    printf("x - Encerrar programa\n\n");
    printf("Escolha uma opcao:");
}

int main(){

    int read=0; //para ver se acabou de ler
    int exit=0;
    int i=0;
```

```
int j=0;
int t,h,v;
char bytes_to_send[1];
char c,opt;
char message_buffer[100];
char xml_message[200];
char buff[10];
FILE *fp; //fp- file pointer
time_t tempo;

printf("%s\n", ctime(&tempo));

DWORD bytes_read;

start_serial();

fp = fopen("AQC.xml", "a"); //abrir o ficheiro XML em modo append
/* fopen() return NULL if last operation was unsuccessful */
if(fp == NULL)
{
   /* File not created hence exit */
   printf("Unable to create file.\n");
   return 1;
}

strcpy(message_buffer, "");
printf("Para aceder ao menu, pressione uma tecla\n");
do{


   if(kbhit()){
      getch();
      menu();
      bytes_to_send[0]=getche();
      printf("\n\n");
      switch (bytes_to_send[0])
      {
      case 'v':
         write_byte(bytes_to_send);
         break;

      case 'c':
         write_byte(bytes_to_send);
         printf("Introduza o novo limite de temperatura (0-100C):");
         scanf("%d", &t);
         if(t<0 || t>100){
            printf("Valor invalido!!!\n");
            break;
         }
         printf("Introduza o novo limite de humidade(0-100%%):");
```

```
            scanf("%d", &h);
            if(h<0 || h>100){
               printf("Valor invalido!!!\n");
               break;
            }
            printf("Introduza o novo limite de velocidade do vento (0-105 Km/h):");
            scanf("%d", &v);
            if(v<0 || v>105){
               printf("Valor invalido!!!\n");
               break;
            }
            bytes_to_send[0]=(char)t;
            write_byte(bytes_to_send);
             bytes_to_send[0]=(char)h;
            write_byte(bytes_to_send);
            bytes_to_send[0]=(char)v;
            write_byte(bytes_to_send);
            break;
         case 'x':
            exit=1;
            break;
         default:
            break;
         }
      }
      else{

         read=0;

         if(read==0){

            ReadFile(hSerial, &c, 1, &bytes_read, NULL);
            if (bytes_read != 1) continue;


            if (c != '\n' && c != '\r')
               strncat(message_buffer, &c, 1);
            else{
               fprintf(stdout, "%s\r\n", message_buffer);

               if(message_buffer[0]=='{'){
                  tempo=time(NULL);
                  if(message_buffer[2]=='E'){
                  strcpy(xml_message,"<Msg2>\n\t<Hora>\n\t\t");
                  strcat(xml_message,ctime(&tempo));
                  strcat(xml_message,"\t<\\Hora>\n\t<Evento>\n\t\tRisco Elevado de
Incendio\n\t<\\Evento>\n<\\Msg2>\n\n");
                  printf(xml_message); //substituir pela cena de escrever no ficheiro
                  fputs(xml_message, fp);
                  strcpy(xml_message, "");
```

```
                }
                if(message_buffer[2]=='T'){
                   tempo=time(NULL);
                   strcpy(xml_message,"<Msg1>\n\t<Hora>\n\t\t");
                   strcat(xml_message,ctime(&tempo));
                   strcat(xml_message,"\t<\\Hora>\n\t<Valores>\n\t\t<Temperatura>");
                   i=3;
                   while(message_buffer[i++]!=':'){}
                   for(j=0;message_buffer[++i]!="";j++){
                      buff[j]=message_buffer[i];
                   }
                   buff[j]='\0';
                   strcat(xml_message,buff);
                   strcat(xml_message,"<\\Temperatura>\n\t\t<Humidade>");
                   while(message_buffer[i++]!=':'){}
                   for(j=0;message_buffer[++i]!="";j++){
                      buff[j]=message_buffer[i];
                   }
                   buff[j]='\0';
                   strcat(xml_message,buff);
                   strcat(xml_message,"<\\Humidade>\n\t\t<Velocidade Vento>");
                   while(message_buffer[i++]!=':'){}
                   for(j=0;message_buffer[++i]!="";j++){
                      buff[j]=message_buffer[i];
                   }
                   buff[j]='\0';
                   strcat(xml_message,buff);
                   strcat(xml_message,"<\\Velocidade Vento>\n\t<\\Valores>\n<\\Msg1>\n\n");
                   fputs(xml_message, fp);
                   printf(xml_message);
                   strcpy(xml_message, "");
                   strcpy(buff, "");
                   i=0;
                }

             }

             strcpy(message_buffer, ""); //limpar o message_buffer

             read=1;
          }
       }

    }
  }while(exit==0);

  close_serial();
  // exit normally
  return 0;
}
```