Departamento de Engenharia Eletrotécnica
Sistemas de Aquisição de Dados (SAD) - 2º Semestre 2019/2020
Docentes: Filipe Moutinho, José Ferreira e Edgar Silva

Trabalho Prático 3

"Solução de aquisição, análise, monitorização e registo de uma estação meteorológica"

Turno P1

Data de entrega: 30/05/2020

Realizado por: Diogo Freitas nº 46148

Manuel Benzinho nº 45295

Ricardo Araújo nº 47416

Índice

Introdução

O presente relatório explica o terceiro trabalho prático no âmbito da disciplina de Sistemas de Aquisição de Dados.

Tivemos como objetivo simular uma estação meteorológica, isto é, controlar a “temperatura”, a velocidade do “vento” e o valor da “humidade”, desenvolver uma versão do sistema de AQC e de uma aplicação de MR.

Para criar um sistema de aquisição e controlo (AQC) foi nos disponibilizado um software (PICSimLab), para simularmos um PIC16F877A a 4MHz. A aplicação de monitorização e registo (MR) foi implementada, em código C. Por fim, os dados foram armazenados, num Servidor remoto, criado pelo professor.

Foi nos dado a liberdade para decidir como calibrar os sensores, o formato das mensagens JSON e dos ficheiros XML, implementar funcionalidades extras e se pretendêssemos usar peças extras e o osciloscópio disponíveis no PICSimLab.

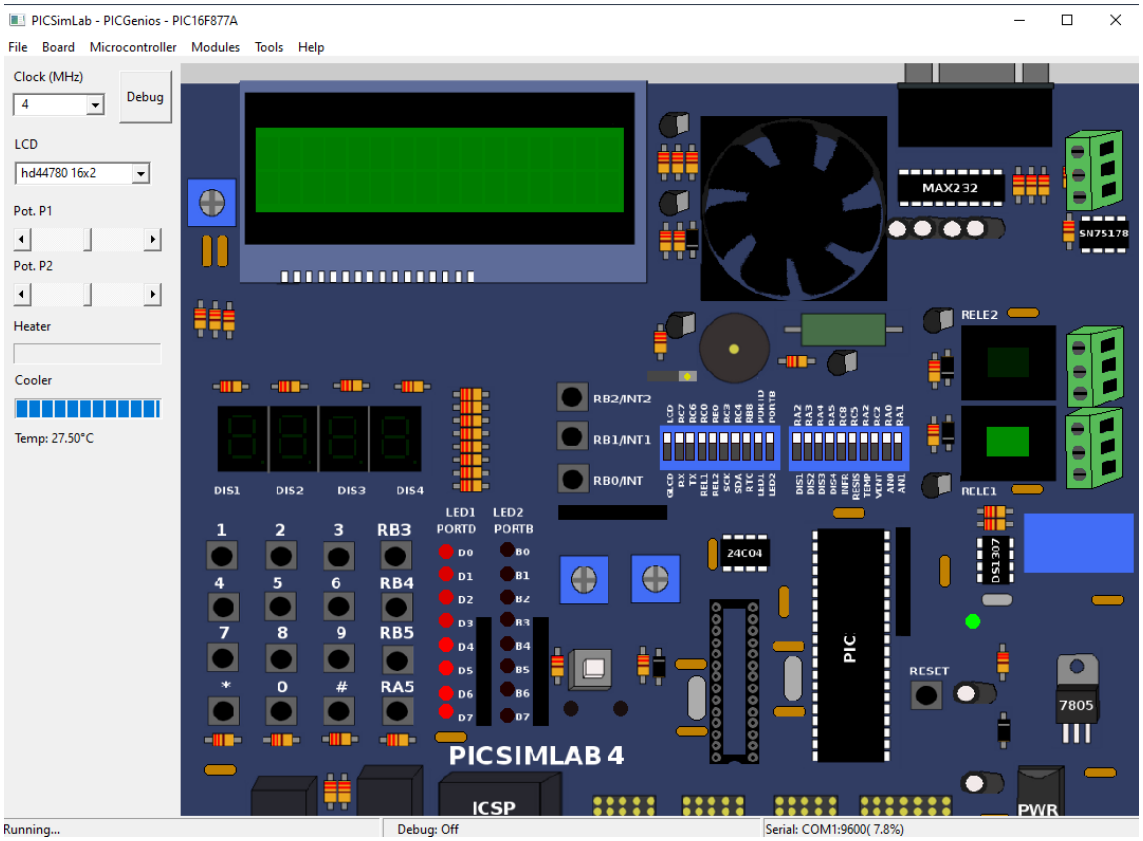

Ilustração 1- PICSIMLAB4

Descrição do Sistema

O sistema de AQC será responsável pela aquisição de valores de temperatura (RA2/AN2), humidade (RA1/AN1) e velocidade do vento (RC0). Este transmite de um em um minuto uma mensagem, para a aplicação de MR, com os valores de temperatura, humidade e velocidade do vento. A partir de um ou mais botões de pressão associados a uma rotina de atendimento de interrupts do PIC, há a alteração do estado do sistema (iniciar, parar, …).

No arranque do sistema de AQC é pedido uma password que terá de ser introduzida através do teclado matricial (saídas: RD3, RD2, RD1, RD0; entradas: RB0, RB1, RB2);

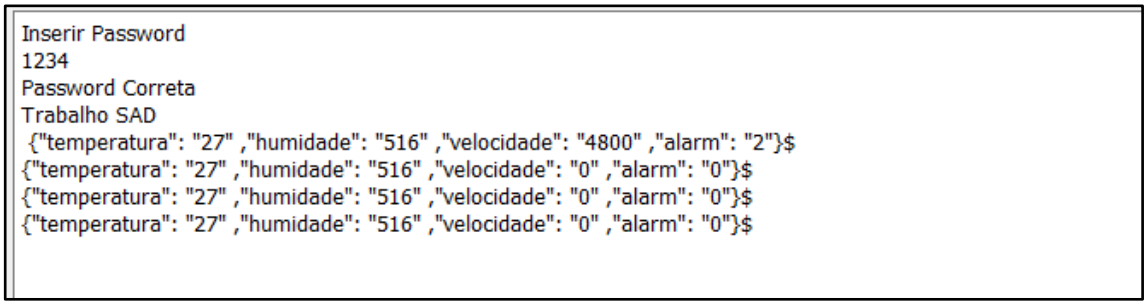

Ilustração 2 - Inserir password 1234 causa um pequeno alerta para iniciar o programa, envia dados a cada 5 segundos.

É possível detetar situações de risco, como por exemplo rajadas de vento, risco elevado de incêndios, entre outras. Quando estamos perante uma situação anômala é ativado um buzzer

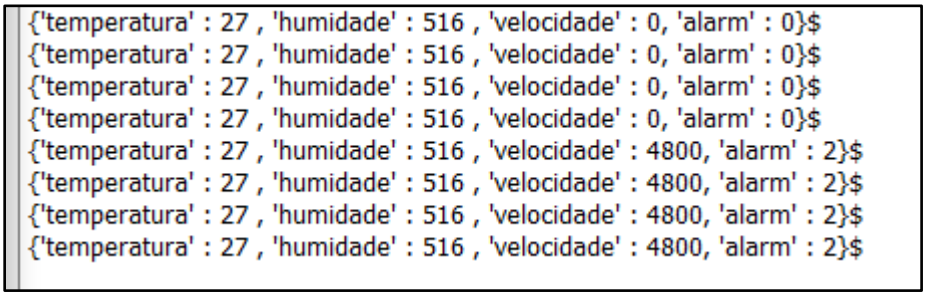

Ilustração 3 - Ativa um alarme quando se mexe no potenciômetro, o buzzer começa a tocar.

(RC1) e é enviado uma mensagem para a aplicação de MR.

A aplicação de MR é responsável por receber as mensagens enviadas pelo sistema de AQC, apresentar os dados recebidos e enviar os dados sobre as situações de risco para o Servidor Remoto (http://193.136.120.133/~sad/), em formato JSON, através de HTTP POST.

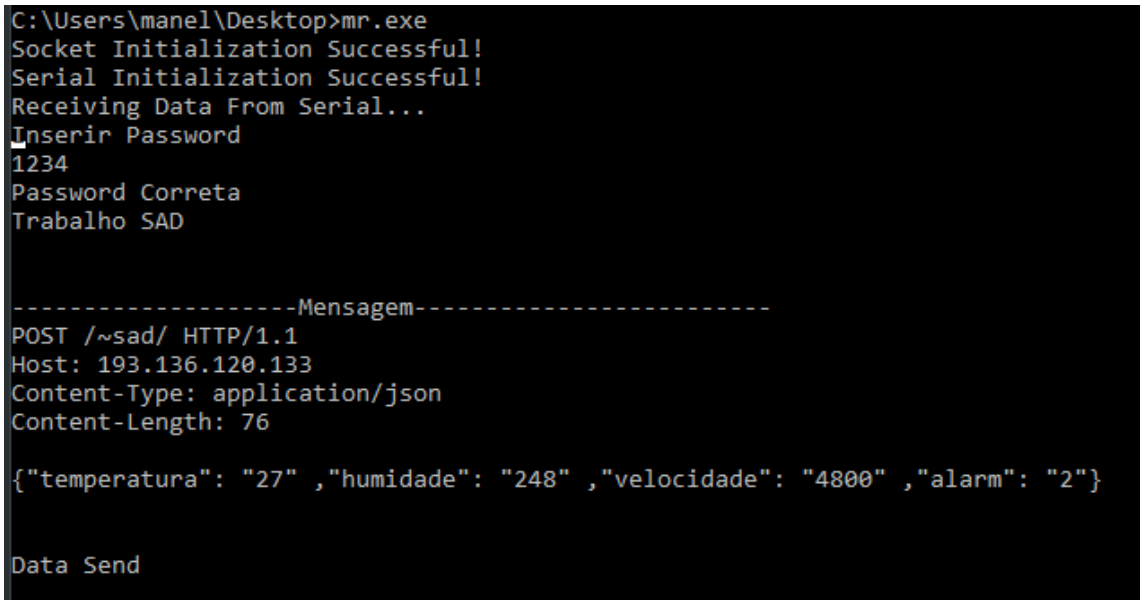

Ilustração 4 - Ficheiro MR permite enviar dados em json para o servidor. Pode consultar password através do cmd.

O sistema AQC e a aplicação de MR estão a interagir através da interface de comunicação série RS232, a uma velocidade de 9600bits/s.

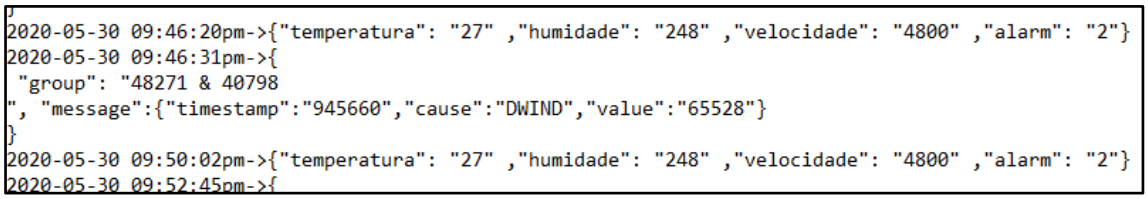

Ilustração 5 - Exemplo de mensagem enviada no log file do dia 30.

Comportamento do Sistema

Sistema AQC:

Inicialização

O programa começa por inicializar os pins do pic, UART e ADC. Grande parte desta configuração é para configurar os 3 timers que usamos e os pins dos sensores. O timer 0 é usado como um clock, lança um interrupt a cada 0.001s, o timer 2 controla o sinal PWM que controla a velocidade da ventoinha e o timer 1 é utilizado para verificar a velocidade da ventoinha, contando as vezes que as pás passam por o sensor infra vermelho num tempo determinado pelo timer 0.

Depois passamos para a configuração das portas referentes ao teclado, dos pins de input analogicos para ler os potenciómetros e finalmente a inicialização de algumas das variáveis utilizadas na main. Aqui só temos que ter em conta algumas variáveis especiais do tipo Data_atual, uma estrutura global que contém os valores dos sensores e Calibrar_struct, que contém a calibração dos sensores que pode ser modificada através de inputs codificados vindos da uart.

Temos sim mais algumas variáveis globais para poder tratar dos interrupts.

Falta ainda as linhas de configuração da eprom mas esses estão explicados no código

```
int main(void)
{
  UART_Init(9600);
  I2C_init(250); //250
  ADC_Init();
  PWM_Initialize();  //This sets the PWM frequency of PWM1


  GIE=1; //Enable Global Interrupt
  PEIE=1; //Enable the Peripheral Interrupt
  INTE = 1; //Enable RB0 as external Interrupt pin

/*****Port Configuration for Timer2 PWM ******/
  CCP1CON = 0X0F;       // Select the PWM Mode
  PR2 = 0x21;           //31Set the Cycle time for varying the duty cycle
  CCPR1L = 50;          // By default set the dutyCycle to 50
  TMR2ON = 1;           //Start the Timer for PWM generatio
/***********_____**********/

/*****Port Configuration for Timer1 Counter******/
  T1CON=0;
  TMR1CS=1;
  T1SYNC=0;
  T1OSCEN=1;
  TMR1ON=1;
/***********_____**********/

/*****Port Configuration for Timer0 Timer******/
  OPTION_REG =0x07;
  TMR0IE=1;
  TMR0=217;
  TMR0IE=1;
/***********_____**********/

  PORTB = 0XFF;                     // PORTB as FF
  TRISB = 0;                        // Configure PORTB as output
  TRISD = 0XFF;                     // TRISD as FF
  PORTD = 0XFF;                     // PORTD as FF
  TRISD = 0XFF;                     // Configure PORTD as input

  TRISA = 0xFF; //Analog pins as Input

  TRISCbits.TRISC0 = 1;//input velocidade fan
  TRISCbits.TRISC1 = 0;//pin buzzer output
  TRISCbits.TRISC2 = 0;//PWM/FAN output
  TRISCbits.TRISC5 = 0;//pin Resistencia output

  TRISBbits.TRISB0 = 1;//pin INT input
  TRISBbits.TRISB7 = 0; //LED ALARM

  char password[4]= {0, 0, 0, 0};
  int password_int = 0;
  int pass_count = 0;

  Data_atual.alarm=0;

  Calibrar_struct calibracao;
  calibracao.rajada_vento = 2400; //calibracao dos sensores
  calibracao.humidade_incendio = 500;  //calibracao dos sensores
  calibracao.temperatuta_incendio = 40;  //calibracao dos sensores
```

Ilustração 6 – Main e inicialização

```
typedef struct
{
    int temperatura;
    int huminade;
    int velocidade;
    int alarm;
}Data_struct;

typedef struct
{
    int rajada_vento;
    int humidade_incendio;
    int temperatuta_incendio;
}Calibrar_struct;
```

Ilustração 7 - estrutura da informação a enviar

Velocidade da ventoinha

A velocidade da ventoinha é calculada de uma maneira peculiar. Temos dois timers que utilizamos para saber as rotações por minuto. Timer1 que está configurado para ser um contador e o Timer0 que já tínhamos referido como um clock normal.

```
void __interrupt() Interrupt_Time()
{
  if (TMR0IF==1) //Check if Timer0 has caused the interrupt
  {
    count_timer_fan_speed++;              //Increasing by 1
    count_minute++;
    TMR0=218; //timer //for 0.001
    T0IF=0;
    if(count_timer_fan_speed==10)
    {  // 1s delay ->100  0.1->10
      count_timer_fan_speed=0;           //when count reaches
      fan_count=TMR1L;
      TMR1L=0;
      TMR0IF=0;
      fan_rpm=(fan_count/7)*600;
    }
```

Ilustração 8 - interrupt da ventoinha

A essência desta maneira de calcular esta no facto que sempre que o Timer0 ativa o interrupt, ele incrementa uma variável e quando esta chega as 10, passaram 0.1s. Depois vamos ao registo onde o Timer1 está a contar as vezes que o sensor RC0 é ativado, iniciamos o counter e como a ventoinha tem 7 pas dividimos por 7 e multiplicamos por 600 para termos as rotações por minuto.

Temos que ter em conta que não podemos deixar o registo TMR1L ficar cheio, daí o tempo ser bastante curto.

O valor 218 veio desta função quando resolvida para Count. O resto das variáveis estão descritas na configuração do Timer0

$$fout = \frac{fclk}{4 * \text{Prescaler} * (256 - \text{TMR0}) * \text{Count}} \quad where \quad Tout = \frac{1}{fout}$$

PIC TIMER0 formula for internal clock

Ilustração 9 - fórmula PIC Timer

Humidade e Temperatura

Foi discutido no trabalho anterior. Utilizamos o ADC previamente inicializado no main e construído para o trabalho anterior mas desta vez escrito no .h e .c e lemos o valor do pin analogico.

Temos que fazer a conversão na temperatura mas decidimos não fazer para a humidade. Bastaria fazer uma regra de 3 simples tal como fizemos com a temperatura.

```
int get_temperatura()
{
  int a;
  int b;

  a = ADC_Read(2);
  b = (a*27.5)/56;
  return b;
}

int get_humidade()
{
  int a;
  a = ADC_Read(1);
  return a;
}
```

Ilustração 10 – função para tirar temperatura

PWM

Tendo em conta a datasheet do pic podemos ver que o timer que temos que usar tem que ser o 2. Também lendo o spreadsheet conseguimos descobrir que o período do PWM é dada por:

7.0 TIMER2 MODULE

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable and is cleared on any device Reset.

$$T_{PWM} = [(PR2) + 1] * 4 * TOSC * (Prescale\ Value)$$

Logo para podermos saber o que pomos no Registo PR2 calculamos

$$PR2 = \left(\frac{Frequência\ do\ Pic}{\frac{1}{T_{PWM}} * 4 * Prescale\ Value}\right) - 1$$

E como podemos ver a azul, o sinal PWM do timer 2, saída RC2 e imagem obtida através do picsimlab

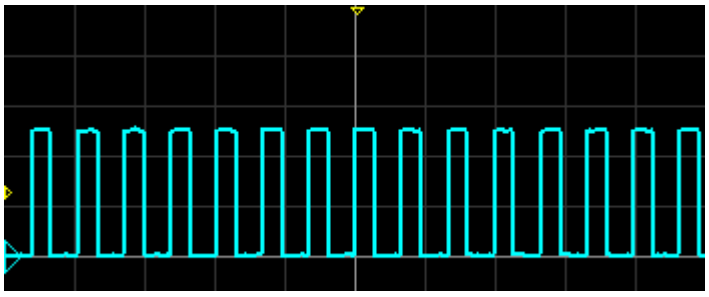

Ilustração 11- osciloscópio do sinal RC2

Aquisição de dados e Envio de json

No nosso programa temos uma função com o nome de Aquisição de dados cujo o objetivo é manter a var global do tipo Data_struct sempre com os valores mais recentes dos sensores. Esta função repete em todos os ciclos do main

```
Data_struct Aquisicao_Dados()
{
  Data_struct aqui;
  aqui.huminade=get_humidade();
  aqui.temperatura=get_temperatura();
  aqui.velocidade=fan_rpm;
  return aqui;
}
```

Ilustração 12- Aquisicao_dados

E com esta variável sempre com os valores mais próximos da realidade podemos então criar um json e enviá lo através da uart para que a aplicação o possa ler e mandar para o servidor.

Esta função é chamada sempre que passa um minuto, mas também quando o programa deteta uma emergência que e definida pelas funções de alarme.

```
void print_json()
{
  sprintf(json,"{'temperatura' : %d , 'humidade' : %d , 'velocidade' : %d, 'alarm' : %d}-> 123$", Data_atual.temperatura , Data_atual.huminade, Data_atu
  UART_Write_Text(json);
  UART_Write_Text("\n");
}
```

Ilustração 13- Escreve para o cuteCom

Funções de Alarme

Temos duas funções de alarme. uma que deteta a existência de uma alarm e outra que processa o alarm. Só vale a pena mencionar que existe uma variável chamada LAST_ALARM que impede o pic de constantemente encher a uart de json sempre que detecta uma situação de alarme. Só quando a situação muda é que o programa envia uma msg. É aqui também que ligamos o LED e o BUZZER para identificar um alarme.

```
void alarm()
{
  if ((Data_atual.alarm==1 || Data_atual.alarm==2 || Data_atual.alarm==3) && LAST
  {
    PORTCbits.RC1=1; //liga buzzer
    PORTBbits.RB7=1; //ligar LED
    print_json(); //imprimir json para uart
    LAST_ALARM=Data_atual.alarm;
  }else if(Data_atual.alarm==0){
    PORTCbits.RC1=0; //desliga buzzer
    PORTBbits.RB7=0; //desliga LED
    LAST_ALARM=0;
  }
}

void alarm_check(Calibrar_struct calibracao)
{
  if(Data_atual.huminade<calibracao.humidade_incendio && Data_atual.temperatura>c
  {
    Data_atual.alarm=1;
    alarm();
  }else if(Data_atual.velocidade>calibracao.rajada_vento){
    Data_atual.alarm=2;
    alarm();
  }else{
    Data_atual.alarm=0;
    alarm();
  }
}
```

Ilustração 14 - função alarme

Ler e escrever na EEPROM

Devidos aos problemas mencionados pelo professor estas funções são chamadas uma depois da outra sem grande propósito. Escreve uma string na memória e le logo de seguida, Estão aqui só para demonstrar que conseguimos utilizar o protocolo IC2 para falar com a eprom e escrever e ler dados. São chamadas antes de entrarmos no loop do main mas depois do teste da palavra pass.

```
I2C_Begin();
I2C_Write(160);
I2C_Write(0);

while(epro_count<count_string)
{
  I2C_Write(text1[epro_count]);
  epro_count++;
}
I2C_End();          // Sends I2C s
```

ter em conta que para escrevermos na eprom basta escrever 160 e depois a localização onde desejamos escrever na memória mas para lermos ja temos que fazer begin mais uma vez para finalizarmos a eeprom que queremos ler e depois também damos a localização onde queremos ler

Em baixo podem ver os valores em asqui da string "Trabalho de SAD" que é o que a função manda para a memória

```
I2C_Begin();
I2C_Write(160);

I2C_Write(1); // word adc

I2C_Begin();
I2C_Write(160);

I2C_Read(data[0]);   // Re
I2C_End();   // Sends I2C
UART_Write_Text(data);
```

Ilustração 15 - funcoes READ WRITE da EPROM

```
0000: 20 54 72 61 62 6C 68 6F 20 53 41 44 20 0A 20 63

0010: 69 70 6C 69 6E 61 20 53 41 44 0A 20 20 33 FF FF
```

Ilustração 16 - excerto da memória EPROM

Mudança de estado do sistema

Para mudarmos de estado utilizamos um interruptor externo e uma variável global. O loop do main está dependente da variável running . Se este for 0 basicamente o programa para de ler os valores e de mecher, ver os alarmes, mexer a ventoinha etc.

```
while(1)
{
  while (running==1)
  {
    move_the_fan();
    Data_atual = Aquisicao_Dados();

    while(UART_Data_Ready())
        calibracao=read_from_MR(calibracao);


    if (!PORTBbits.RB3)
    {
      PORTCbits.RC5=1; //liga resistencia
```

Ilustração 17- enviar informação através de loop

Essa variável é modificada na função que tem o propósito de tratar dos interrupts .

Como podemos observar quando acontece um interrupt se a flag INRF estiver a um mudamos o valor desta variável.

Outra função desta variável é parar o envio dos json que acontece de minuto a minuto visto que este envio também está dependente desta variável

```
if (INTF==1){
  INTF=0; // Reset the external int
  if (running==1)
  {
    running = 0;
    UART_Write_Text("Sleep Mode.");
    UART_Write_Text("\n");
  }else{
    running = 1;
    UART_Write_Text("Waking Up!");
    UART_Write_Text("\n");
  }
 }
}
```

```
if(count_minute==6000) //6000
{
  count_minute=0;
  if (running==1)
  {
    print_json();
  }
 }
}
```

Ilustração 18 - Função interrupt e controlo de transmissao de dados

Ler pedidos da aplicação

O pic está a espera de uma string de 8 valores vinda da uart. No loop da main vemos se UART_Data_Ready está a 1, esta função está a olhar para uma flag que fica a um se existir um valor no registo da uart.

Assim que tivermos lá alguma coisa vamos para a função read_from_MR(9 que vai tratar de decodificar a msg da aplicação.

Se a 1 posição do vetor for 1, quer dizer que a aplicação só quer um reenvio dos dados logo voltamos a enviar o json e saímos da função, se for zero quer dizer que ela quer modificar os valores de alarm. Então temos que olhar para a posição seguinte. Se esta for zero modificamos o vento, se for um modificamos a humidade se for 2 modificamos a temperatura. finalmente as próximas 4 posições é o novo valor a colocar e a última o fim da string.

```
Calibrar_struct read_from_MR(Calibrar_struct calibracao)
{
    char output[8]={0,0,0,0,0,0,0,0};    // recebe uma sring de um numero xxxxxx
    int int_input = 0;                   // se for  1xxxxx é um pedido de informacao
    UART_Read_Text(output,8);            // se for  00yyyy mudar o valor da calibracao do
    UART_Write_Text(output);             // se for  01yyyy mudar o valor da calibracao da
                                         // se for  02yyyy mudar o valor da calibracao da
    if(output[0]==49)
    {
        //UART_Write_Text("1 Pedido de informacao \n");
        print_json();
        return calibracao;
    }
    int_input = (output[2]-48)*1000+(output[3]-48)*100+(output[4]-48)*10+(output[5]-48);
    UART_Write_Decimal(int_input);
    UART_Write_Text("\n");
    switch (output[1])
    {
        case 48:
            //UART_Write_Text("0 0 vento \n");
            calibracao.rajada_vento = int_input;
            break;
        case 49:
            //UART_Write_Text("0 1 humindade \n");
            calibracao.humidade_incendio = int_input;
            break;
        case 50:
            //UART_Write_Text("0 2 temperatura \n");
            calibracao.temperatuta_incendio = int_input;
            break;
    }
    return calibracao;
}
```

Ilustração 19 - Read from MR

Teclado

Com a ajuda do código fonte das instruções do picgenius e o código que o criador do picsimlab disponibiliza no seu site fizemos o código para poder usar o teclado para poder inserir a pass. O valor da pass está guardado uma variável global no início caso seja preciso mudar o valor. o valor é inserido em forma de caracter logo tem q ser passado para inteiro para poder comparar. O código não aceita passes com letras, só números. A função que adquire o valor do teclado tem que ter uma valor de timeout, um delay para que tudo possa correr bem e as linhas não se misturem com as colunas. Pelo menos foi a única maneira de nos conseguirmos pôr a funcionar.

```
while(password_int!=PASS) //fica a esper
{
  pass_count=0;
  while(pass_count<4)
  {
    TRISBbits.TRISB0 = 0;
    TRISD=0x0F;
    unsigned char abc = teclado(1500)+0x
    TRISD=0x00;
    TRISBbits.TRISB0 = 1;
    if (abc!=47)
    {
      UART_Write(abc);
      password[pass_count]=abc;
      pass_count++;
    }
  }
  password_int = (password[0]-48)*1000+(
```

Ilustração 20 - confirmar password

Aplicaçao MR:

Enviar mensagem para o servidor através da aplicação MR

Primeiro passo é criar um socket, temos que ter o pacote *sockaddr_in* com o nome de *server*, com *server* evocamos os métodos address, Port e family para estabelecer ligação, onde o endereço IP vai ser 193.136.120.133, para onde vamos enviar informação, o Porto 80 é normal usar para servidores web e html. AF_INET permite enviar sockets do tipo Internet Protocol v4 addresses (IPv4). Criamos e iniciamos o socket *s* no ***socket_init()***, a ligação é estabelecida com sucesso.

```
WSADATA wsa;
SOCKET s;
struct sockaddr_in server;
HANDLE hSerial;
DCB dcbSerialParams = {0};
COMMTIMEOUTS timeouts = {0};
```

```
server.sin_addr.s_addr = inet_addr("193.136.120.133");
server.sin_family = AF_INET;
server.sin_port = htons(80);
```

Ilustração 21 - No lado esquerdo, temos variáveis globais para o server e handler ,entre outros e no lado direito está IP para onde vamos enviar a mensagem.

```
int Socket_init()
{
    if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
    {
        printf("Failed. Error Code: %d", WSAGetLastError());
        return -1;
    }
    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET)
    {
        printf("Could not create socket : %d", WSAGetLastError());
        return -1;
    }
    if (connect(s, (struct sockaddr *)&server, sizeof(server)) < 0)
    {
        puts("Connect Error");
        return -1;
    }
    return 0;
}
```

Ilustração 22 - função socket_init() responsável por estabelecer ligação com o endereço IP

Próximo passo é criar um handle chamado ***hSerial***, este handler é responsável para podermos estabelecer ligação com o CuteCom através das portas COM1/ COM2, no modo READ/WRITE. Para ler informação vinda do CuteCom usamos a função **ReadFile()** e para escrever usamos **WriteFile()**. hSerial está configurado como COM1 ou seja o COM2 terá que ser feita no CuteCom.

```
int Serial_init()
{
    hSerial = CreateFile("\\\\.\\COM1", GENERIC_READ | GENERIC_WRITE, 0, NULL,
                         OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hSerial == INVALID_HANDLE_VALUE)
    {
        return -1;
    }
    else
    {
        dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
        if (GetCommState(hSerial, &dcbSerialParams) == 0)
        {
            CloseHandle(hSerial);
            return -1;
        }
        dcbSerialParams.BaudRate = CBR_9600;
        dcbSerialParams.ByteSize = 8;
        dcbSerialParams.StopBits = ONESTOPBIT;
        dcbSerialParams.Parity = NOPARITY;
        if (SetCommState(hSerial, &dcbSerialParams) == 0)
        {
            CloseHandle(hSerial);
            return -1;
        }
        timeouts.ReadIntervalTimeout = 50;
        timeouts.ReadTotalTimeoutConstant = 50;
        timeouts.ReadTotalTimeoutMultiplier = 10;
        timeouts.WriteTotalTimeoutConstant = 50;
        timeouts.WriteTotalTimeoutMultiplier = 10;

        if (SetCommTimeouts(hSerial, &timeouts) == 0)
        {
            CloseHandle(hSerial);
            return -1;
        }
        return 0;
    }
}
```

Ilustração 23 - Serial_init() função responsável pelo handler e por estabelecer ligação com CuteCom através de das portas COM 1 e 2.

A função **Serial_receive()** trata da leitura de dados. O ReadFile tem 5 parâmetros, o primeiro é hSerial para saber o ficheiro ou buffer donde vai buscar a informação, de seguida temos um char **c** que basicamente vai buscar um caracter no buffer, a ideia é ler por caracteres até encontrar o $, o ReadFile retorna sempre true sempre que receber um caracter e se o **c** for igual a $ o while fica false e vai sair do loop.Como o PIC24 está sempre enviar informação para o COM2, o ReadFile vai estar sempre a ler e daí retornar sempre true, o PIC também já envia em formato json não precisamos de adaptar o nosso código, cada linha termina com $.

durante a condição do while enchemos o vector **read_file** com o caracteres vindos do buffer, como $ quebra o while e não vai entrar no read_file.

```
void Serial_receive(char read_file[76])
{
    char c;
    int bytes_read = 0, n = 0, keep_alive = 1;
    printf("Receiving Data From Serial...\n");
    int i=0;
    while (ReadFile(hSerial, &c, 1, &bytes_read, NULL) && c!='$')
    {
        if (bytes_read==1)
        {
            read_file[i] = c;
            printf("%c", c);
            i++;
        }
    }
    read_file[i]='$';

}
```

Ilustração 24 - Serial_receive() , responsável por receber informação do CuteCom e guardar mensagem na variável read_file, que é a mesma que a variável json e Station_data, visto no main.

Após a criação dos socket, do handler e receber a mensagem que está guardada no read_file. No main(), tratamos primeiro, da autenticação do socket, **socket_init()** e depois handler, através **serial_init()**, que estabelece a ligação entre o COM 1 e 2 , como já mencionado atrás.

Depois vamos receber informação do buffer com **serial_receive()** que tem como parâmetro o vector **Station_data**, que é mesmo o vetor **read_file**, onde vai buscar os caracteres.

se o Station_data não estiver vazio, ele está pronto para ser enviado.

usamos o REQBIN para testar as mensagens em json e para obtermos o RAW file que basicamente é o header necessário para o seu envio através do MR.c.

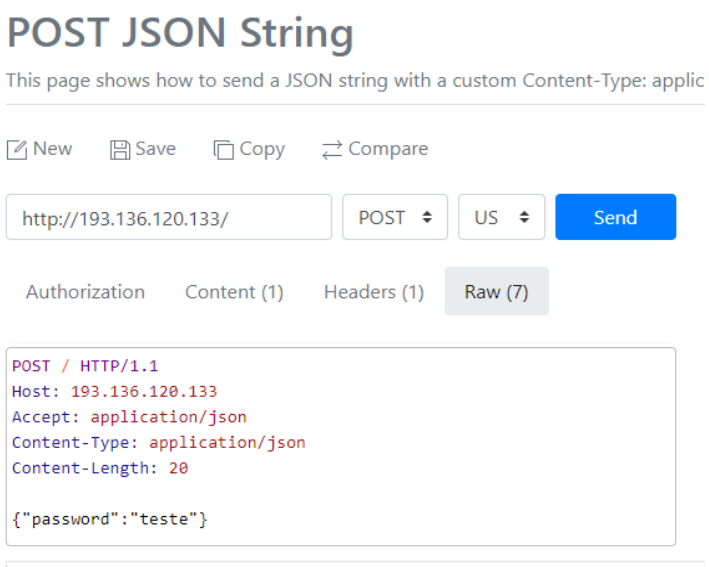

Ilustração 25 - formato RAW de json usado para aplicação MR

Para corrigir uns bugs, associamos a mensagem Station_data como sendo **json**, sem $, e precisamos do tamanho da mensagem. Com isto, criamos a mensagem final **msg**, através do raw File vindo do reqbin, para criamos a estrutura final do Json. Com isto, bastou enviar msg através da função send(), que é fornecida através do pacote **server** , o mesmo usado para criação de sockets.

```
json = strtok(Station_data, "$");
length = strlen(json);

sprintf(msg,"POST /~sad/ HTTP/1.1\nHost: 193.136.120.133\nContent-Type: application/json\nContent-Length: %d\n\n%s",length,json);


if( send(s , msg , strlen(msg) , 0) < 0)
{
    puts("\n\nSend failed");
    return 1;
}
puts("\n\nData Send\n");
```

Ilustração 26 - passamos o header do RAW file para msg e incluímos a mensagem json e o seu tamanho. a string msg é que será enviado para o servidor.

Conclusão

Com este trabalho, desenvolvemos competências no que diz respeito ao uso do PIC16F877A, bem como, o software PicsimLab que simula este PIC e interagir entre uma aplicação em C e um servidor remoto.

Devido a todos os obstáculos que tivemos de enfrentar devido a situação atual do país, este trabalho foi realizado em casa. O grupo juntou-se usando uma aplicação que nos permitia fazer partilha de ecrã.

No decorrer do trabalho, houveram membros do grupo que tiveram alguns bugs com a instalação dos softwares recomendados pelo professor. Também existiram problemas na parte do MR pelo simples facto de ter sido a primeira vez que interligamos uma aplicação C feita por nós, com um servidor na internet.

No fim, apesar dos problemas descritos, anteriormente, o grupo conseguiu contornar-los e apresentar um trabalho bastante positivo aos nossos olhos.