Universidade de Aveiro

Departamentos de Física e Mecânica 2007

Provisório

Development of integrated electroestimulation equipment for medical rehabilitation of spinal cord injuries

Referência

ANEXO

Eduardo Durana

Aveiro, 20 de Novembro de 2007

Índice

Introdução

Este relatório descreve o conjunto de técnicas e competências desenvolvidas no decorrer dos trabalhos realizados no âmbito deste projecto. A programação em C de micro controladores PIC da Microchip (família 18F), com recurso ao ambiente de desenvolvimento MPLAB IDE, compiladores C MPLAB C18 e assembly MPASM, e ao programador ICD2 também da Microchip.

A primeira fase de trabalhos consistiu na criação de circuitos electrónicos em placa branca para a ambientação ao laboratório, acompanhado do desenvolvimento de programas em C para programar os microcontroladores. Nestes programas foram exploradas as entradas analógicas e digitais do PIC, assim como dos seus módulos de conversão analógica/digital, temporização, geração de PWMs (Pulse Width Modulation, com base na qual se pode criar sinais analógicos a partir de pinos digitais. Muito útil para a criação de sinais para aplicar em electroestimulação) e comunicação UART (para a comunicação por porta de série RS232) e SPI (Serial Peripheral Interface).

Implementação de comunicação ethernet através do controlador ENC28J60 e do microcontrolador PIC18F2550 ambos da microchip. Com base na comunicação ethernet foram utilizados os protocolos de transporte TCP/IP e UDP. Comunicação por ethernet é muito vantajosa em relação à comunicação por porta de série rs232, uma vez que permite taxas de envio muito mais elevadas, com maior segurança permite comunicar sem fios, comunicar com um PDA que torna o dispositivo mais portátil e utilizar a Internet como meio de comunicação. No entanto para ser implementada com absoluta robustez exige maior gasto de tempo.

Objectivos

O objectivo deste projecto passa por integrar um conjunto de competências no domínio da instrumentação para desenvolver equipamento a utilizar na reabilitação de lesões medulares. Com características principais a capacidade de aplicar electroestimulação e ser sensível a força de pressão aplicada pelo utilizador.

Resultados

Protótipos dos circuitos:

Controlo e comunicação;

Electroestimuação;

Sensores de força capazes de se ajustar às diferentes capacidades motoras dos pacientes.

Aplicação para o PDA para monitorização dos sensores de força.

Aplicação de electroestimulação realizada.

Perspectivas

Melhorar e calibrar os sensores de força;
Sinal de electroestimulação com ondas bifásicas;
Medir a corrente e controlara sua intensidade com o PIC. Mostrar as características da onda aplicada na electroestimulação e impedância da região electroestimulada na aplicação PDA.
Utilizar o controlo sobre a sensibilidade dos sensores e dos parâmetros da onda de electroestimulação como mais valia em exercícios de fisioterapia e em estudos.
Utilizar conversores de tensão para corrente para obter obter uma fonte de corrente controlável (envolve trabalhar num sistema elevador de tensão-provávemente com condensadores)
Garantir o cumprimento dos parâmetros de segurança na aplicação de electroestimulação.

Conclusão

Anexo Técnico

MPLAB/COMPILADOR C

O ambiente de desenvolvimento integrado MPAB, permite o desenvolvimento de aplicações para microcontroladores da microchip®. A linguagem de desenvolvimento pode ser em assemby cujo compilador é incluído por defeito no MPLAB, ou pode ser utilizada a linguagem de programação C, recorrendo a um compilador apropriado, que terá de ser adicionado ao ambiente de desenvolvimento. No caso foi utilizado um compilador também da microchip, designado MCC18, de Microchip C Compiler para controladores da família 18F.

O MPLAB IDE para além de servir para escrever código de programas a serem carregados para o micro controlador, compila, simula, faz detecção de erros (‘debugging’) e faz a linkagem para código de máquina apropriado para ser transferido para o respectivo micro controlador.

O MPLAB pode ser obtido gratuitamente através do site da Microchip, a sua instalação é iniciada correndo um executável de instalação.

Para criar código, o ficheiro onde este é escrito tem de estar inserido num projecto.
Antes de criar o projecto é necessário seleccionar o ‘device’ (microcontrolador a ser utilizado), que pode ser feito directamente usando o ‘project wizard’ ou através do ‘select device’ do menu ‘configure’.
Para criar o projecto pode ser através de ‘new project’ ou ‘project wizard’ ambos do menu ‘project’.
As ‘language tools’ são definidas no segundo passo do ‘Project wizard’ ou podem ser definidas directamente em ‘set language tools’ do menu ‘project’. É importante verificar e/ou definir os executáveis e os caminhos e directorias de procura, para o assembler no

‘microchip MPASM Toolsuite’ e para o compilador C no microchip C18 toolsuite. Isto em termos de assembler, compilador C, livrarias e linker.

Nos projectos realizados foi utilizado o compilador mcc18. No ‘set language tools’ os executaveis para:

- ‘MPASM assembler’ são incluidos no próprio MPLAB a e sua localização é ‘C:\Program Files\Microchip\MPASM Suite\MPAsmWin.exe’;
- MPLAB C18 C Compiler tem de se ir buscar o executável à pasta de instalação do mcc18, previamente instalada no C: na localização ‘C:\mcc18\bin\mcc18.exe’;
- a livraria MPLIB Librarian o executável a ser usado está na localização ‘C:\mcc18\bin\mplib.exe’; o executável para o MPLINK Object linker vai ser localizado em ‘C:\mcc18\bin\mplink.exe’.

Definiu-se também:

- o ‘include path’ como ‘C:\mcc18\h\’, (onde o compilador armazena os system header files);
- o Library Path, como ‘C:\mcc18\lib\’ (onde residem as livrarias e ficheiros de objectos pré-compilados);
- e o Linker-Script Path, como ‘C:\mcc18\lkr\’ (onde se encontram os ficheiros linker scrip).

Esta definição pode ser feita também no default search path & directories do microchip C18 toolsuit no ‘set language tools’, ou nas ‘build options’ do projecto no menu ‘project’, janela ‘general’.
Nas ‘build options…’, na janela ‘MPLINK linker’ é conveniente activar a opção ‘Supress COD-file generation’ caso se pretenda que a dimensão da localização do projecto não constitua um problema (caso contrario existe um ‘file/path format’ com dimensão máxima de 62 caracteres).

Na continuação da construção do projecto tem de ser atribuído um nome e uma directoria, onde serão incluídos os ficheiros constituintes.

Existe ainda a necessidade de adicionar alguns ficheiros ao projecto. Como se utiliza o compilador C18 é necessário a adição dos ficheiros de livraria e de linker script. Desta forma adiciona-se:

- p18F258.lib existente em ‘C:\mcc18\lib\’ a ‘Library Files’;
- o ficheiro 18F258.lkr’ existente em ‘C:\mcc18\lkr\’ a ‘linker scripts’, para informar o linker sobre a organização da memoria do microcontrolador seleccionado;
- o ficheiro do código a ‘Source files’;

Isto pode ser feito directamente no ‘project wizard’ ou então adicionar na janela de projecto (‘nome do projecto.mcw’).

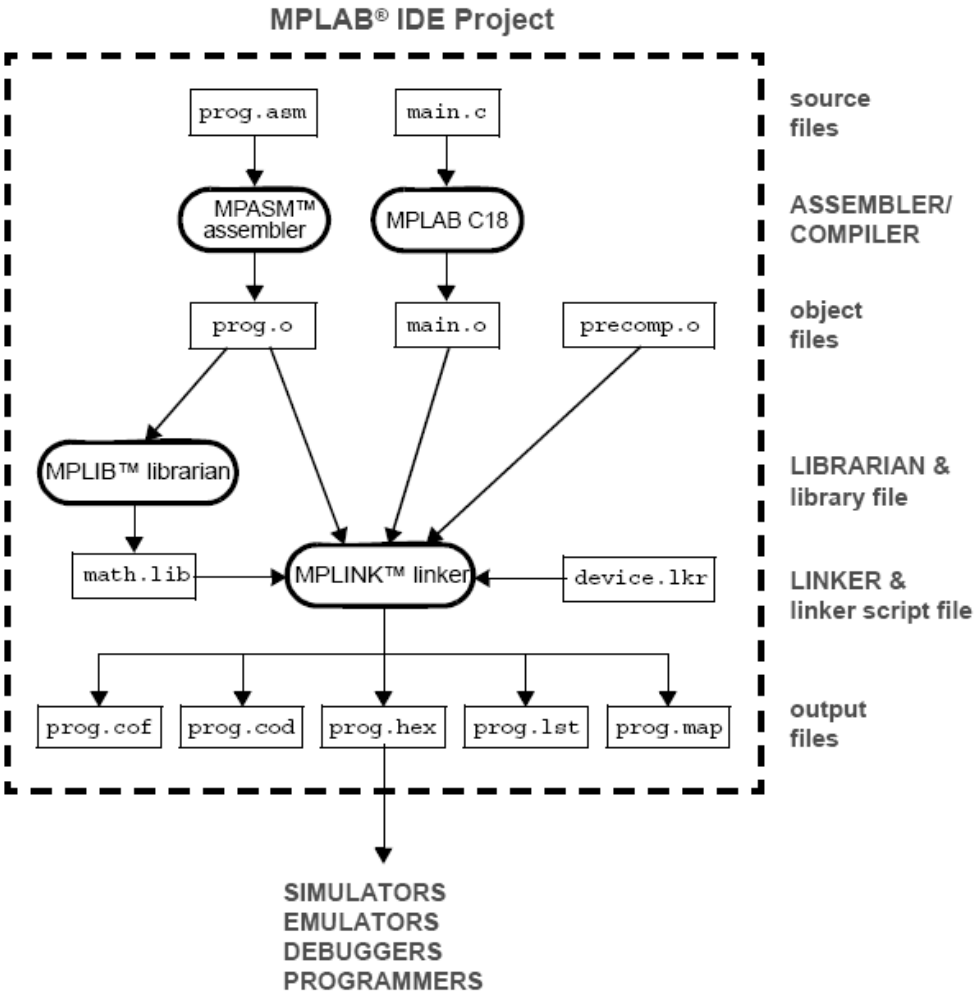

Fig. 1: Esquema retirado do MPASM/MPLINK/MPLIB User's Guide. Este esquema descreve o processo desenvolvido pelo MPLAB IDE projet, mostrando a função do MPASM, MPLAB C18, MPLIB e MPLINK, donde se pode concluir que é fundamental a correcta definição dos executáveis para estas ferramentas para que a programação do PIC esteja em conformidade.

No item 'configuration bits' do menu 'configure' é possível configurar algumas características especiais do CPU. Podem também ser configuradas no código, mas para não haver confusões convém fazer apenas numa das duas formas.

No sentido de se conseguir uma boa familiarização com a programação de micro controladores e da aplicação das potencialidades de alguns dos seus módulos constituintes úteis para o desenvolvimento do trabalho, foram realizados alguns exercícios cujos programas iram ser de seguida brevemente descritos. Nesta descrição vão sendo introduzidos a forma da construção dos programas e alguns conceitos relevantes.

Programa 1 (Criação de uma onda quadrada e activação de uma saída por acção de um interruptor de botão)

A criação de ondas quadradas assume especial importância na programação de micro controladores na medida em que estes apenas possuem saídas digitais, i.e. apenas se

podem encontrar em um de dois estados, ‘1’ (high) ou ‘0’ (low). Podem servir para criar PWM (Pulse With Modulation), que será descrito mais à frente assim como as suas aplicações. A acção de um botão também tem grande importância, pois permite um controlo físico externo por parte de um utilizador.

```
/*
Inclusão da ‘header file’ p18f128.h que contém protótipos de funções e
definições para o microcontrolador em função do compilador. Também se
poderia por um ‘header file’ para processador genérico para todos os
PIC18XXXX chamado p18CXXX.
*/
#include <p18f258.h>

void main(void)   //é a função principal.
{
//inicialização das variáveis. Onde ‘i’ é uma variável de incrmento e
‘T’ uma variável temporal em função do tempo de ciclo do
microcontrolador
int i;
const unsigned int T=50000;

//configuração do PORTA em termos de input/output.
TRISA = 0b00000100;          //RA2 – entrada os outros pinos são saída
/*
A configuração do conversor analógico digital coloca todos os pinos
como digitais, para definir deve-se consultar o ‘datasheet’ do micro
controlador para ver qual o valor dos bits PCFG correspondentes à
situação pretendida.
*/
ADCON1bits.PCFG3 = 0;
ADCON1bits.PCFG2 = 1;
ADCON1bits.PCFG1 = 1;

/*
ciclo com o objective de manter o programa a correr ininterruptamente
*/
      while(1)
      {
/*
criação de uma onda quadrada à saída do pino RA0 com período
equivalente a 2T iterações (funciona como delay). Ao pino RA0 foi
ligado um LED para visualizar a oscilação.
*/
            for (i=0;i<T;i++)
            {
                  i=i;
            }

            PORTAbits.RA0 =1;

            for (i=0;i<T;i++)
            {
                  i=i;
            }

            PORTAbits.RA0 =0;

/*
Utilização do pino RA2 como entrada, com tensão proveniente de um
interruptor de ‘press button’ que está a ‘0’ (ou baixa voltagem) com o
```

```
interruptor aberto e a '1' (ou alta voltagem) quando o interruptor é
pressionado.
o pino RA1 é uma saída com o mesmo valor da entrada em RA2, onde se
ligou um LED e vermelho por forma a estar ligado com interruptor
liberto e desligado com o interruptor pressionado, e um LED verde e um
motor eléctrico por forma a terem comportamento contrário ao LED
vermelho. Correspondente ao esquema mostrado na Fig. 2.
*/
            if (PORTAbits.RA2 == 1)
            {
                  PORTAbits.RA1 =1;
            } else {
                  PORTAbits.RA1 =0;
            }

      }

}
```

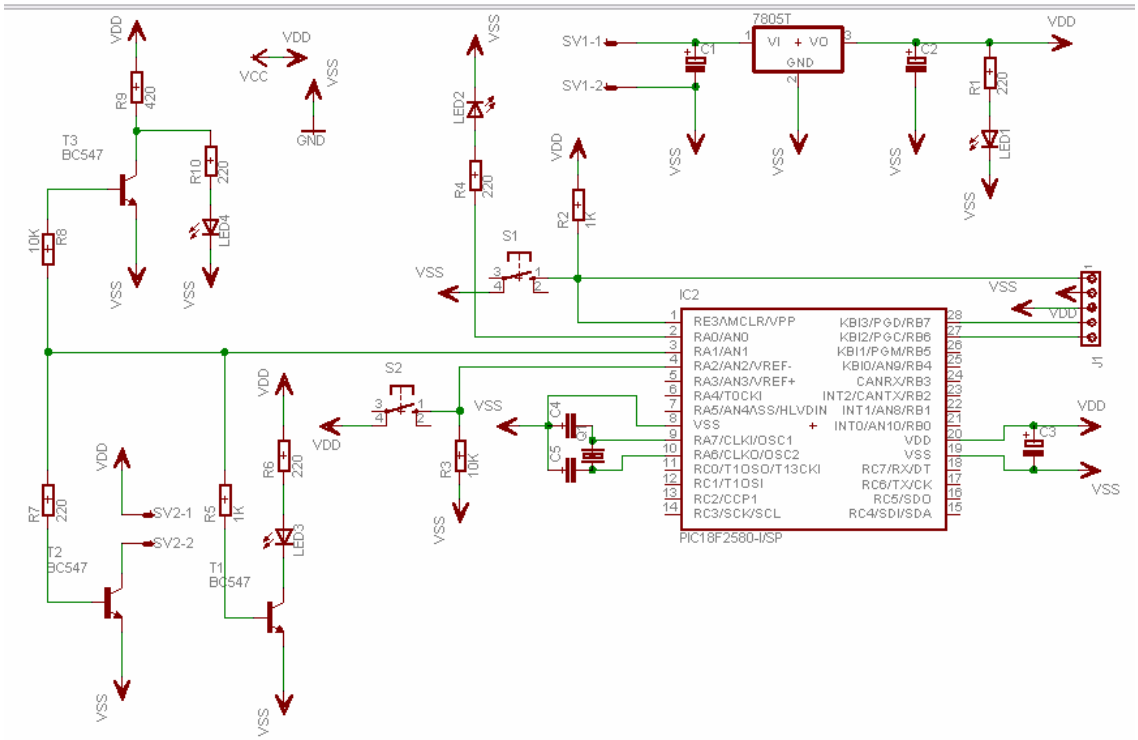

Fig. 2: Esquema de LEDs e motor ligados ao PIC através de transístores por forma a diminuir a corrente solicitada ao microcontrolador. Outra das atenções importantes a ter em conta é a necessidade de condensadores para a alimentação do PIC e para o cristal de modo a estabilizar ao máximo a tensão aplicada, fundamental para o correcto funcionamento do PIC.

Programa 2 (construção de uma PWM – Pulse Width Modulation)

Outro exemplo consistiu na construção de uma PWM (Pulse Width Modulation). Uma PWM é uma onda quadrada, onde a largura do pulso (quando o sinal esta a '1') pode ser modulado em relação ao período total da onda. É caracterizada pelo período da onda e pelo 'duty cycle', que é a percentagem do período que a largura do pulso representa.

O programa conta com a seguinte inicialização:

```
…
const unsigned int T=50000, duty_cycle=20;
unsigned int  t_on, t_off;
t_on=duty_cycle*(T/100);
t_off=T-t_on;
…
```

Com atrasos obtidos através de ciclos for, com a razão entre o tempo on e tempo off definidos pelo 'duty cycle' definido pelo programador.

```
…
      while(1)
      {
            for (i=0;i<t_on;i++)
            {

                  for(j=0;j<5;j++)
                        i=i;

                  PORTAbits.RA1 =1;
            }

            for (i=0;i<t_off;i++)
            {

                  for(j=0;j<5;j++)
                        i=i;

                  PORTAbits.RA1 =0;
            }

      }
…
```

Programa 3 (modulação da PWM por conversão analogica/digital de um sinal proveniente de um potenciometro)

No exemplo seguinte a diferença em relação ao anterior é que o duty cycle passou a ser definido pela conversão analógico/digital do valor de tensão à saída de um potenciómetro.

Para tal foi necessário configurar uma entrada analógica.
Os registos de configuração do conversor foram inicialmente definidos de seguinte forma:

```
…
ADCON0=0b10000001;
ADCON1=0b01001110;
…
```

Resumindo, estes registos configuram o conversor para funcionar com as seguintes propriedades:

- Frequência de conversão é igual à frequência do oscilador sobre 64 (Fconv = Fosc/64).
- Esta frequência é para dar tempo de segurança ao conversor para funcionar correctamente e respeita a especificação do datasheet do microcontrolador, que está a funcionar com uma Fosc = 40Mhz.

- O pino que está configurado como analógico é o AN0.
- A conversão não está em progresso.
- O conversor é ligado.
- O resultado da conversão é justificado à esquerda. O que significa que dos 10 bits de conversão, os 8 mais significativos vão ser colocados no registo ADRESH. Regendo apenas por registo está-se a desconsiderar apenas os dois bits menos significativos o que contribui apenas para uma pequena perda de resolução.
- Apenas o pino AN0 é analógico enquanto os restantes pinos do PORTA são configurados como digitais.

Notar que a configuração destes parâmetros pode ser alterada em função das características do módulo de conversão analógico digital. Para tal é fundamental ver o datasheet do micro controlador utilizado para o programar de forma a responder da melhor maneira possível ás necessidades.

Dentro do código, para cada conversão é necessário activar o bit go/done do registo ADCON0 (activando o conversor) e esperar que este fique a '0' para garantir que o conversor vai fazer uma leitura correcta.

```
…
      while(1)
      {
      ADCON0bits.GO=1;
//    //esperar que o conversor fique pronto
      while (ADCON0bits.GO == 1);
…
```

Só depois se atribui ao duty cycle o valor lido pelo conversor.

```
…
      duty_cycle=(((unsigned int)ADRESH)*100/255);
…
```

E prosseguir com o código da mesma forma que no exemplo anterior.

Programa 4 (recurso a um temporizador para modular a PWM)

Neste novo exemplo é utilizado um timer do microcontrolador para estipular o período em que a saída está a '1' e a '0' em vez de usar ciclos 'for' para fazer o 'delay'.

A grande vantagem em relação ao método escolhido anteriormente (utilizando ciclos 'for' para provocar o atraso) para a criação da PWM é que ao se utilizar um temporizador para definir os tempos em que a saída se encontra num determinado estado é mediada pela activação de uma 'bandeira' que assinala que o temporizador concluiu a contagem do tempo programado. Esta bandeira é um bit de interrupção do temporizador, que é avaliado ciclicamente o que liberta o micro controlador para executar outras tarefas no entretanto.

O timer escolhido foi o TMR0 pois permite um prescaler maior que os restantes. O prescaler é o valor pelo qual é dividida a frequência interna do microcontrolador. Isto associado ao facto de dar para ser configurado com 16 bits. Permite temporizar períodos de poucos segundos, sem que se tenha de contabilizar 'overflows'. Esta temporização máxima é mais que suficiente para o efeito pretendido. Uma vez que a PWM a ser gerada vai estar ligada a um motor eléctrico e o que se pretende é com o potenciómetro alterar a sua velocidade de rotação. Para este fim foi escolhido um período de 10ms.

A configuração do timer consistiu em definir alguns bits do registo T0CON da seguinte forma:

```
…
T0CONbits.TMR0ON = 1; (tem a função de ligar o timer0).
T0CONbits.T08BIT = 0; (configura o timer com 16 bits).
T0CONbits.T0CS = 0; (define TMR0 como timer interno, ou seja funciona
como temporizador e não como contador).
T0CONbits.PSA = 0; (prescaler assigned - a clock do TMR0 deriva da
saída do prescaler).
INTCONbits.TMR0IF = 0; (baixa a bandeira de interrupção do TMR0).
…
```

Utilizou-se o seguinte código para encontrar e definir um prescaler suficiente para os valores do período T e frequência interna de oscilação inicializados em milisegundos e kHz respectivamente.

```
…
while (pre_scl_aprox<T*(Freq_int/1000)/(0xffff/1000))
      {
            pre_scl_aprox=pre_scl_aprox++;
      }

      if (pre_scl_aprox<2)
            {

            pre_scl_val=2;
            T0CONbits.T0PS0=0;
            T0CONbits.T0PS1=0;
            T0CONbits.T0PS2=0;
            }
      else if (pre_scl_aprox<4)
            {
            pre_scl_val=4;
            T0CONbits.T0PS0=1;
            T0CONbits.T0PS1=0;
            T0CONbits.T0PS2=0;
            }
      else if (pre_scl_aprox<8)
            {
            pre_scl_val=8;
            T0CONbits.T0PS0=0;
            T0CONbits.T0PS1=1;
            T0CONbits.T0PS2=0;
            }
…
}
```

E assim sucessivamente até um valor máximo de prescaler de 256.

A seguinte parte do código destina-se a encontrar a posição de inicialização do timer e introduzi-la nos registos TMR0H e TMR0L, para o espaço de tempo em que a PWM está a ‘1’ e a ‘0’ respectivamente.

```
…
            t_on=(unsigned int)((float)duty_cycle*((float)T/100.0));
            incr_ini_on=0xffff-Freq_int/pre_scl_val*t_on;
```

```
TMR0H=incr_ini_on>>8;
TMR0L=(incr_ini_on<<8)>>8;
PORTAbits.RA1 =1;
INTCONbits.TMR0IF=0;


if (INTCONbits.TMR0IF);
{
      t_off=T-t_on;
      incr_ini_off=0xffff-Freq_int/pre_scl_val*t_off;

      TMR0H=incr_ini_off>>8;
      TMR0L=(incr_ini_off<<8)>>8;
      PORTAbits.RA1 =0;
      INTCONbits.TMR0IF=0;
      INTCONbits.TMR0IF=0;
}
```

…

O PIC sabe que o tempo definido chegou ao fim quando o bit do interrupt flag do TMR0 fica a ‘1’ após o overflow do mesmo. É fundamental não esquecer de voltar a ‘0’ esta flag antes de se voltar a testar o seu estado, caso contrário o PIC vai interpretar de imediato que voltou a acontecer overflow. A onda gerada foi confirmada através do osciloscópio.

Programa 5 (Comunicação RS232 entre o PIC e um PC)

Outro exemplo ainda com a PWM foi a criação de um programa para que o PIC transmitisse o valor lido pelo conversor quando solicitado pela recepção de um caractere especifico, neste caso ‘T’. A comunicação foi efectuada por porta de série.

Para configurar a transmissão e recepção utilizou-se a USART do PIC em modo assíncrono. Que usa o formato non-return-to-zero com os bits a comunicar enquadrados entre um start bit e um stop bit.

A programação de ‘interrupts’ vai ser utilizada para comunicação por porta de série. O recurso à programação de ‘interrupts’ tem a vantagem de permitir que um determinado evento, imediatamente interrompa a linha de execução do programa, sendo a execução transferida para outro espaço da memória onde se encontra código escrito especificamente para dar resposta ao evento que desencadeia a interrupção. O compilador utilizado faz automaticamente o que se chama de salvar o contexto para que assim que a rotina de atendimento ao ‘interrupt’ termina, a execução volte exactamente ao ponto onde estava anteriormente mantendo as mesmas condições.

```
…
//configuração da transmissão e recepção
RCSTAbits.SPEN = 1;          //porta de série - enable
TXSTAbits.TX9 = 0;           //transmissão de 8bits
TXSTAbits.TXEN = 1;          //transmit enable
TXSTAbits.SYNC = 0;          //modo assíncrono, único válido para
                             //comunicação //com o computador.

TXSTAbits.BRGH = 0;          //low speed
```

```
/*
o valor do baude rate é determinado em função da frequência de
oscilação e da taxa a que os bytes são enviados.
*/
SPBRG = 64;
/*
activa o interrup enable do receptor.
o interrupt acontece quando RCIF estiver a '1', o que acontece quando
a recepção está completa.
*/
PIE1bits.RCIE = 1;

RCSTAbits.CREN = 1;           // activa recepção continuamente.

RCONbits.IPEN = 1;            //set priority levels on interrupts
/*
define o interrupt de recepção como de alta prioridade
*/
IPR1bits.RCIP = 1;

INTCONbits.GIEH = 1;    //enable all high priority interrupts
...
```

Para que o PIC execute uma acção específica ao receber informação de fora é necessário criar um 'interrupt'.
O protótipo da rotina de serviço ao 'interrupt' (RSI) é definido da seguinte forma.

```
...
void high_priority_interrupt (void);
void HighVector(void);
...
```

O código da RSI deve ser atribuído ao espaço da memória destinado a esse fim. Como este espaço é limitado atribui-se o código da RSI à localização correcta, mas para prevenir situações em que o código é superior ao espaço destinado, escreve-se o código numa outra localização da memória, mas atribui-se à posição correcta. Da forma a seguir descrita.

```
...
//define-se a posição onde o código vai ser atribuído
#pragma code HighVector=0x8
void HighVector (void)
{
/*
Código em assembler para ir para onde o código está realmente a ser
escrito (fora do espaço originalmente destinado ao interrupt).
*/
 _asm goto high_priority_interrupt _endasm
}
#pragma code // return to default code section
#pragma interrupt high_priority_interrupt
void high_priority_interrupt (void)
{
/*
Desactivação do interrupt de recepção da USART, para evitar
sobreposição de interrupts
*/
      PIE1bits.RCIE=0;
//Lê e atribui a uma variável o valor recebido
      c = RCREG;
//Atribui a variáveis os valores a transmitir
```

```
      RA0 = val_potenc;
      RA2 = PORTAbits.RA2;
//Apenas vai transmitir se receber 'T'
      if (c == 'T')
      {
/*
Espera que TXIF fique a '1' que acontece assim que o TXREG acaba de
transmitir a informação precedente e é esvaziado.
*/
            while (!PIR1bits.TXIF);
/*
Só então o TXREG volta a ser carregado com nova informação e o TXIF
volta a ficar a '0'.
*/
            TXREG = (1<<5)|(RA0*31/255);

            while (!PIR1bits.TXIF);
            TXREG = (10<<5)|(RA2*31/255);
/*
O protocolo para a comunicação utilizado consiste nos valores a serem
transmitidos estarem codificados nos 5 bits menos significativos e a
identificação está nos 3 bits mais significativos.
      }

//É necessário que TXREG seja lido para que RCIF seja posto a '0'.

//Volta a activar o interrupt de recepção da USART
      PIE1bits.RCIE=1;
}
…
```

As directivas #pragma code definem a posição da memória a ser utilizada

Para que a transmissão e recepção sejam realizadas é necessário que a informação chegue ao PIC através do pino RX e sai através do pino TX. Como esquematizado na figura seguinte.

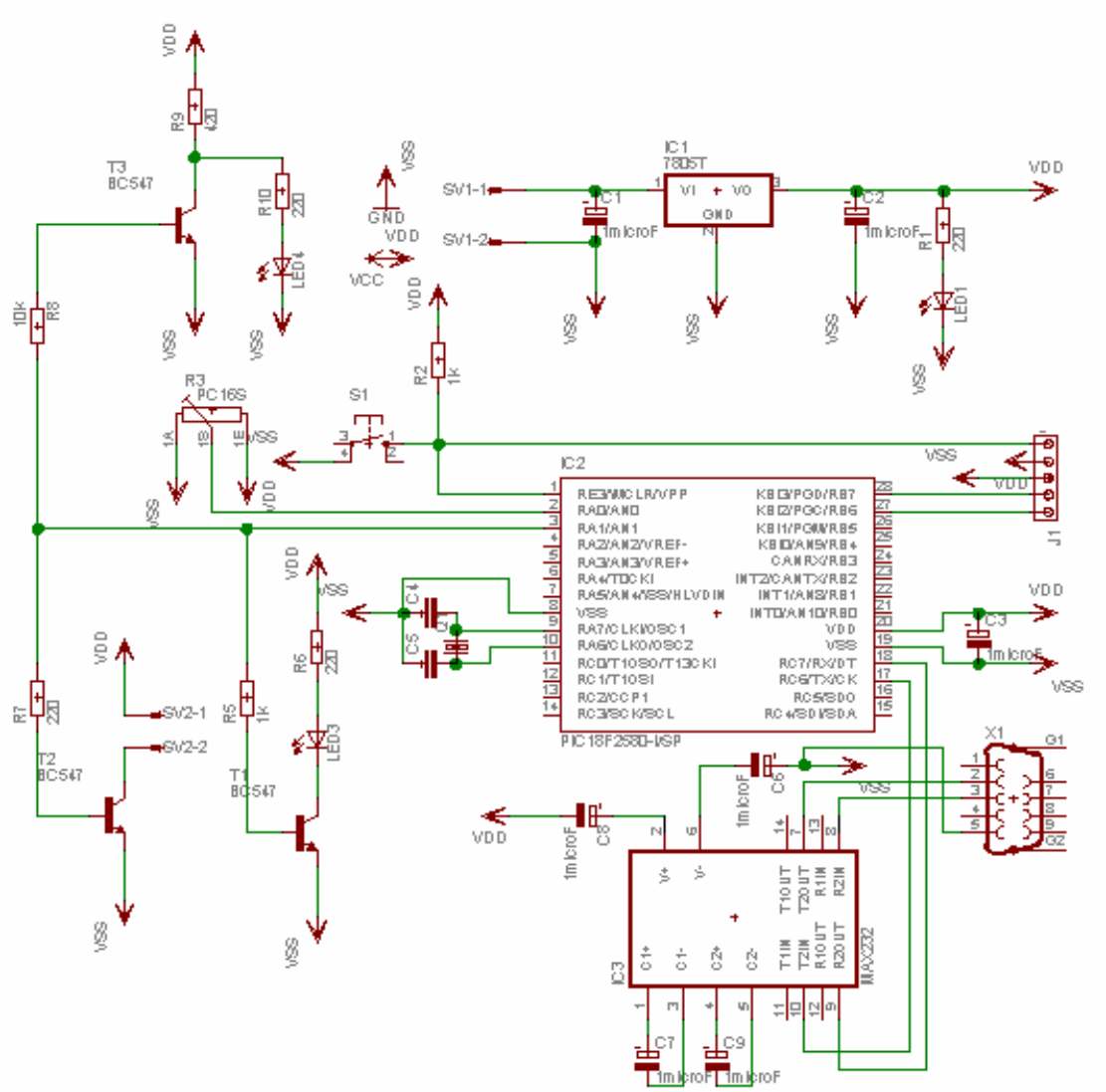

Fig. 3: Esquema de ligação de uma porta RS232 ao PIC18F258.

Programa 5.1 (criação de um temporizador em Matlab)

Depois foi criado um programa em MATLAB para fazer o gráfico dos valores recebidos.

_____

```
clc
delete(timerfindall)
clear all
close all

periodo=0.01;

RA0=zeros(1,20,'uint8');
RA2=zeros(1,20,'uint8');
val_RA0=0;
val_RA2=0;
```

Onde a função principal cria um timer que em intervalos de tempo bem definidos chama a função TXRC definida da seguinte forma

```
t = timer( 'Period', periodo,'ExecutionMode','fixedrate');
set(t,'timerFcn','TXRC')

start (t)
```

programa 5.2 (função série para comunicar com o PIC e mostrar graficamente a informação recebida)

```
tempo_0=0;
periodo=0.1;
dim_max=20;

s=serial('com4','BaudRate',9600,'DataBits',8,'flowcontrol','none','par
ity','none','stopbits',1);
%(cria uma ligação de série com as mesmas características definidas
para a transmissão no PIC)
fopen(s)

fprintf(s,'T')
%onde envia a informação (‘T’) para que o PIC transmita os
%valores pretendidos, para depois serem lidos,
%identificados e utilizados para criar um gráfico.

[R] = fread(s,2,'uint8');

if bitshift(R(1),-5)==1
    val_RA0=bitand(R(1),31);
end
if bitshift(R(2),-5)==1
    val_RA0=bitand(R(2),31);
end

if bitshift(R(1),-5)==2
    val_RA2=bitand(R(1),31);
end
if bitshift(R(2),-5)==2
    val_RA2=bitand(R(2),31);
end

fclose(s)
RA0=[RA0(1,2:end),val_RA0];
RA2=[RA2(1,2:end),val_RA2];

tempo=linspace(-dim_max+1,0,dim_max);
plot(tempo,RA0,tempo,RA2)
drawnow
```

Programa 6 (criação de um contador de impulsos com visualização através de LEDs)

Foi realizado ainda um outro exemplo para um contador de impulsos com visualização através de LEDs. No caso apenas foram utilizados dois LEDs para visualizar os dois bits menos significativos.

```
…
//RA2 – entrada os outros pins são saida
      TRISA = 0b00000100;
//configura o PORTB como saida
      TRISB=0b0;

      //na entrada RA2 foi introduzida uma onda quadrada criada por um
gerador de ondas.

      counter=0;  O contador é iniciado em ‘0’
      prev_input=0;

            PORTBbits.RB0=counter & 1;
            PORTBbits.RB1=(counter>>1)&1;
/*
Os LEDs vão estar ligados aos pinos RB0 e RB1 aos quais vão ser
associados o primeiro e segundo bit menos significativo
respectivamente.
*/
      while(1)
      {
            input=PORTAbits.RA2;
/*
O contador conta ciclos e portanto vai fazer o incremento quando
detecta uma passagem do valor de entrada de ‘0’ para ‘1’.
*/
            if (prev_input == 0 && input==1)
            {
                  counter++;
                  PORTBbits.RB0=counter&1;
                  PORTBbits.RB1=(counter>>1)&1;

                  prev_input=1;
            }
      else if (input == 0 )
            prev_input=0;
      }
}
```

Comunicação ethernet

Breve introdução ao controlador ENC28J60

O integrado ENC28J60 é um controlador capaz de enviar e receber pacotes ethernet, cumprindo a especificação IEEE 802.3. É composto por um módulo, Controlador de Acesso ao Meio (MAC) e integra também um módulo da camada física (PHY). Suporta funcionamento em full- e half- duplex, detecta e corrige a polaridade do porto 10 base T de forma automática. Permite ainda a programação de alguns automatismos, como a retransmissão de pacotes em colisões, geração de 'padding' (preenchimento do pacote até uma dimensão mínima para cumprir as especificações IEEE 802.3) e CRC (campo do pacote que serve para quem o recebe fazer verificação de erros) e rejeição de pacotes com erros. Tem também uma interface de comunicação série SPI (Serial Peripheral Interface) capaz de suportar frequências até 20MHz.

O PIC comunica com o controlador ethernet (ENC28J60) por SPI (Serial Peripheral Interface), recorrendo a comandos definidos no datasheet do ENC28J60.

A memória do ENC28J60 encontra-se dividida em três tipos:

- Registos de controlo (constituindo por 4 bancos de 32 registos cada);
    - Ethernet (ETH);
    - Médium Access control (MAC);
    - Media independent interface management (MIIM);
- Ethernet buffer
    - 8kbytes de memória;
    - Necessita de ser definido espaço para transmissão e para recepção;
- Registos PHY (constituindo 32 endereços de registos de 2bytes);
    - Loopback;
    - Half/full duplex;
    - Codificação/ descodificação da informação na interface de pares entrelaçados;
    - LED.

Constituição do pacote ethernet;

Campo	Tamanho	Função
Preambulo	7bytes	Sincronização
Inicio de frame	1byte	Delimitação
MAC address de destino	6bytes	Identificação destinatário
Mac address de origem	6bytes	Identificação remetente
Tipo/tamanho	2bytes	Informação protocolar
Dados	46 a 1500bytes	Informação dados
FCS	4bytes	Verificação de erros

Esquema

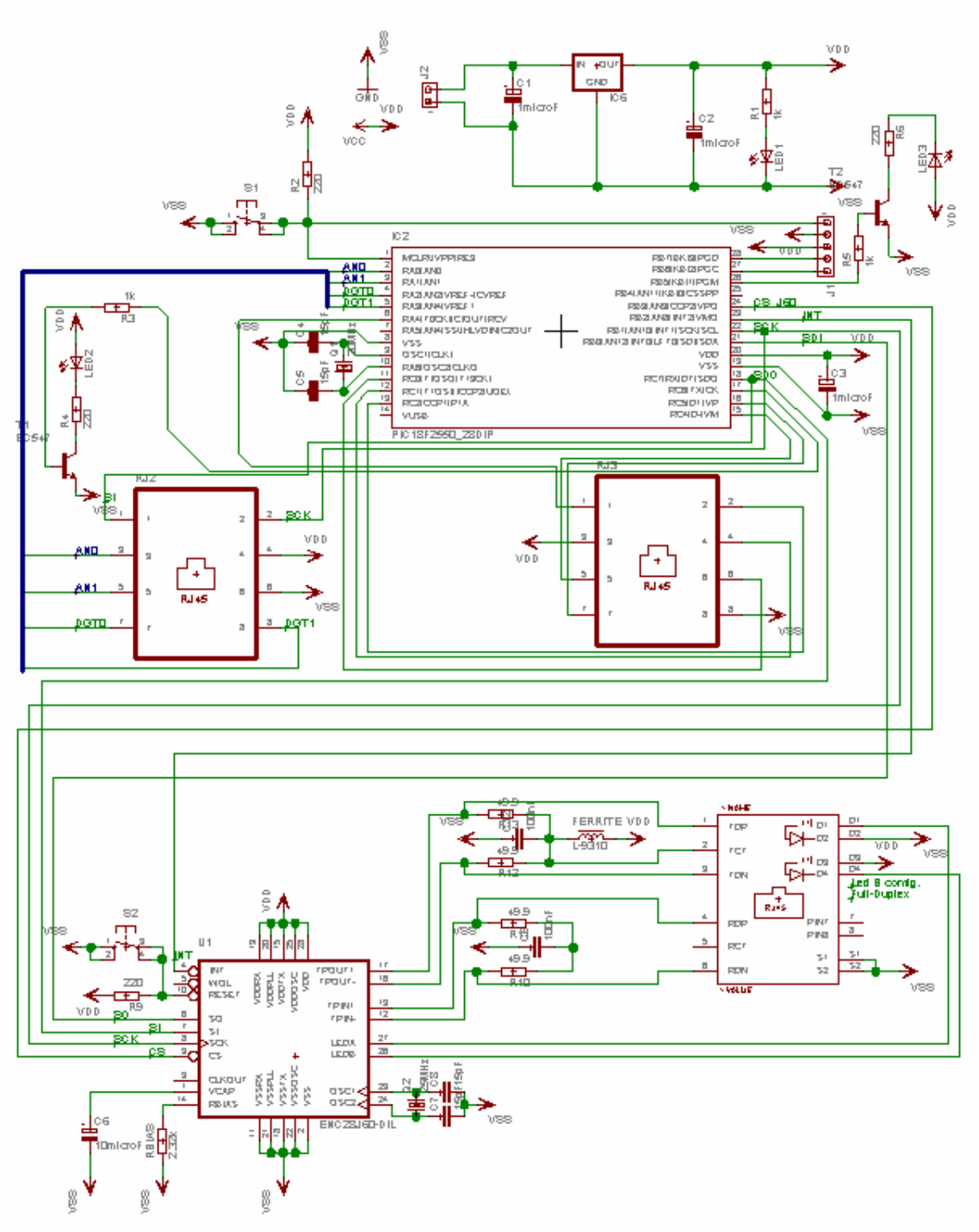

Fig. 4. esquema da ligação do controlador ethernet ENC28J60 ao PIC.

Rotina

No ficheiro da função principal (Main) do projecto criado para por em pratica a comunicação por ethernet, são definidos os bits de configuração importantes. São importantes, porque determinam o modo de do funcionamento do PIC.
O benefício de fazer esta configuração directamente neste ficheiro, em relação a realiza-la na interface gráfica, é que desta forma é mais difícil que aconteça uma alteração acidental destes parâmetros.

Cada PIC tem uma forma característica de definir os bits de configuração. É importante ver o respectivo capítulo do datasheet.

```
…
////////////////////////////////////////////////////////////////////
////////////CONFIGURATION BITS//////////////////////////////////////
////////////////////////////////////////////////////////////////////
/*
Existem registos próprios para os bits de configuração, mas o
compilador mcc18 permite usar o comando 'pragma config' para os
programar.
Para se utilizar o Pragma config convém primeiro conhecer a lista de
configuration bits que o compilador dispõe para o pic em causa.para
isso deve-se escrever no command prompt "mcc18 -p18f2550 --help-
config"
*/

/*
O multiplicador da frequência PLL (Phase Locked Loop circuit),
necessita de uma frequência de entrada de 4MHz para gerar uma
frequência fixa de 96MHz. Como foi usado um cristal de 20MHz terá de
ser dividido por 5 pela PLL.
*/
#pragma config PLLDIV = 5
//é utilizada a frequência usada pelo 'usb' de 48MHz
#pragma config USBDIV = 2
//indica de se utiliza a frequência da PLL (96MHz) dividida por 2
#pragma config CPUDIV = OSC1_PLL2
//usa-se um cristal de alta velocidade
#pragma config FOSC = HSPLL_HS
//activação do monitor 'Fail-Safe Clock'
#pragma config FCMEM = ON
//activação do reset de voltagem por 'brown-out'
#pragma config BOR = ON
//quando o potencial VDD for inferior a 2V há reset por 'brown-out'
#pragma config BORV = 21
//desliga o temporizador cão de guarda (WDT).
//ou seja o wdt é controlado por software
#pragma config WDT = OFF
//o post scaler temporiza o WDT para 4ms
#pragma config WDTPS = 1
//activa o pino de 'Master Clear' (reset)
#pragma config MCLRE = ON
//activa o reset por 'Stack full/underflow'
#pragma config STVREN = ON
//desactiva programação em baixa voltagem
#pragma config LVP = OFF
```

Esta é a rotina que é executada ciclicamente:

```
...
while(1)
{
/*
É feito pooling da recepção, que consiste na avaliação do estado de um
bit que funciona como bandeira que assinala a chegada de um pacote.
Quando esse bit estiver a '1' activa-se a variável de estado global
STAT.receber, para que a recepção seja efectuada. De seguida é
```

```
importante baixar a respectiva bandeira para que possa assinalar a
chegada de novos pacotes.
*/
      if (LerRegETH(EIR)&0b01000000)
      {
            STAT.receber = 1;

            BFCReg((byte)EIR,0b01000000);
      }
```

Faz-se também pooling de erros na recepção e transmissão, de forma análoga à anterior mas agora a bandeira a avaliar é relativa à chegada de pacotes com erros, e as variável globais de estado a serem activadas são STAT.RXERROR e STAT.TXERROR. para facilitar a análise visual de cada vez que acontece um erro, realiza-se toggle a um LED especifico.

```
      if (LerRegETH(EIR)&0b00000001)
      {
            STAT.RXERROR = 1;
            BFCReg((byte)EIR,0b00000001);
            PORTBbits.RB5 = !PORTBbits.RB5
      }

      if (LerRegETH(EIR)&0b00000010)
      {
            STAT.TXERROR = 1;
            BFCReg((byte)EIR,0b00000010);
            PORTCbits.RC6 = !PORTCbits.RC6;
      }

/*
Avaliação da conexão do cabo Ethernet. Faz-se a avaliação da bandeira
de alteração do estado do link como nos casos anteriores mas neste
caso especifico para se baixar a bandeira (automáticamente) é
necessário ler um registo do módulo PHY para verificar qual o estado
de link actual.
*/
      if (LerRegETH(EIR)&0b00010000)
      {
            STAT.linkChange = 1;

      }

/*
Agora é a fase de dar resposta ao estado do sistema.
Caso a variável de estado STAT.receber esteja activa (a '1') é chamada
a função de recepção. Esta função é responsável por ir buscar a
informação recebida ao buffer de recepção, ir desencapsulando e
enquadrando nos respectivos campos e por fim agir em conformidade.
Seguidamente é importante voltar a colocar STAT.receber a 'o'.
*/
      if (STAT.receber == 1)
      {
            receber();
            STAT.receber = 0;
      }
/*
Caso a variável ReqToSend esteja active, significa que em alguma
      instância do programa foi solicitado o envio de um pacote. Nesta
      fase não interessa onde, porque ou qual a natureza do pacote a
      ser enviado. À medida que as variáveis de estado relevantes
      forem sendo verificadas pela função transmissão, o pacote vai
      sendo construído. Uma vez que se pretende a transmissão de
      pacotes TCP, deve-se colocar a variável STAT.ReqToSend a '1'.
*/
      if( (STAT.ReqToSend == 1) )
      {
            STAT.TCP=1;
```

```
            transmitir();

            STAT.TCP=0;
            STAT.lastAck=0;
            STAT.ReqToSend = 0;

      }

      if (STAT.linkChange == 1)
      {

/*
A leitura do registo PHIR automaticamente apaga a bandeira de
alteração de estado de link.
*/

/*
Para aceder a um registo a registos PHY (da camada física) não pode
ser feito de forma directa. É necessário programar o endereço do
registo pretendido no registo MIREGADR comandar o início da leitura
activando o bit MICMC.MIIRD, esperar que o processo de leitura fique
concluído (até o bit MISTAT.BUSY passar a '0')
*/
//Mudar para o banco2, onde se encontra o registo MIREGADR
            BankSel(2);
            EscreverReg((byte)MIREGADR, PHIR);
//Colocar a '1' o bit MICMD.MIIRD para iniciar a operação de leitura
            BFSReg((byte)MICMD, 0b00000001);
//mudar para o banco3, onde se encontra o registo MISTAT
            BankSel(3);
//esperar que a operação de leitura fique coompleta
            while (LerRegMAC((byte)MISTAT) & 0b00000001);

            BankSel(2);

            BFCReg((byte)MICMD, 0b00000001);
/*
Colocar a '0' o bit MICMD.MIIRD para terminar a operação de leitura.
Nesta altura a bandeira de alteração de link é baixada
automaticamente.
*/
            PHIRL = LerRegMAC(MIRDL);

//para verificar se o link está ou não ligado
            BankSel(2);
            EscreverReg((byte)MIREGADR, PHSTAT2);
            BFSReg((byte)MICMD, 0b00000001);
            BankSel(3);
            while (LerRegMAC((byte)MISTAT) & 0b00000001);
            BankSel(2);
            BFCReg((byte)MICMD, 0b00000001);
//ler o byte mais significativo do registo PHYSTAT2
            PHSTAT2H = LerRegMAC(MIRDH);
/*
Verificar o estado do bit correspondente ao LSTAT. Caso esteja a '1' o
link está estabelecido. Caso esteja a '0', significa que o cabo está
desconectado. Importa voltar a colocar a '0' a variável
STAT.linkChange.
*/
            if(!(PHSTAT2H & 0b00000100))
            {
            //LINK 'off'
            }
            else if ( (PHSTAT2H & 0b00000100))
            {
            //LINK 'on'
            }
            STAT.linkChange = 0;
      }
}
```

Configuração do micro controlador

A configuração dos portos e dos pinos do PIC é realizada numa função de inicialização.

```
//Inicialização dos portos
      PORTA = 0;
      LATA = 0;
      PORTB = 0;
      LATB = 0;
      PORTC = 0;
      LATC = 0;

/*
A configuração dos pinos é feita da seguinte forma: RA0 e RA1 -
entrada os outros pins são saida (como o PIC é sempre Master o RA5/!SS
não irá funcionar como 'slave select' podendo portanto ser saída); SDI
e INT2 são entradas, SCK (RB1) e RB3, RB4 [escolhido para Chip Select
(CS)] são saídas; o TRISC<7> tem de ser posto a '0' porque o SDO é uma
saída.
Nos registos TRISX para os respectivos portos, as entradas assumem
valor '1' e as saídas assumem valor '0'.
*/
      TRISA = 0b00000011;
      TRISB = 0b00000101;
      TRISC = 0b00000000;

//Configuração dos interrupts
//Activa niveis de prioridade nos interrupts
      RCONbits.IPEN = 1;
//Permite todos os interrupts de alta prioridade
      INTCONbits.GIEH = 1;
//Faz com que a interrupção externa 2 aconteça em 'falling edge'
      INTCON2bits.INTEDG2=0;
//INT2 como interrupt de alta prioridade
      INTCON3bits.INT2IP=1;
//INT2 como interrupt activado
      INTCON3bits.INT2IE=1;

//Configuração do conversor ADC
/*
AN0 (RA0) e AN1 (RA1) é analógico todos os outros pinos digitais.
Configura as voltagens de referência max e min como Vdd e Vss (Gnd)
respectivamente.
Valor da conversão justificado à esquerda (ficando os 8MSb no registo
ADRESH os restantes dois bits nos 2 MSb do registo ADRESL)
Para o relógio de conversão é escolhida uma frequência = F_osc/64
(devido à elevada frequência do PIC).
O tempo de aquisição é definido como 12 períodos do relógio de
conversão (para poder executar os 10bits de resolução da conversão).
*/
      ADCON1=0b00001101;
      ADCON2=0b00101110;
```

Configuração do SPI para a comunicação entre o PIC, o controlador ethernet e os potenciómetros digitais:

A comunicação por SPI (Serial Peripheral Interface), permite a comunicação série síncrona entre vários dispositivos. Tem a característica de possuir um 'Master', que é responsável por definir o relógio da comunicação, de iniciar a comunicação e escolher os 'slaves' que iram participar na respectiva comunicação.
Este tipo de comunicação exige portanto a existência das seguintes linhas:
A linha de relógio é fundamental porque a comunicação é síncrona;
A linha de selecção de 'slave', inha que o 'Master' coloca a '0' com os 'slaves' que pretende comunicar;
A linha de 'output' de dados e nos casos de comunicação bidireccional torna-se também necessária uma linha de 'input' de dados.
A configuração dos registos associados à comunicação por SPI no PIC é feita da seguinte forma.

```
...
/*
Vem especificado que o ENC28J60 suporta apenas o SPI modo 0,0. Os
potenciómetros digitais MCP42010 funcionam em modo SPI 0,0 ou 1,1.
portanto a configuração SPI será feita para funcionar no modo 0,0 que
implica: CKE=1;CKP=0.

//Reset
SSPCON1bits.SSPEN = 0;
SSPSTATbits.SMP = 0;
SSPSTATbits.CKE = 1;
SSPCON1bits.CKP = 0;
//Configura o PIC como 'master' com 'clock=Fosc/16'
SSPCON1bits.SSPM3 = 0;
SSPCON1bits.SSPM2 = 0;
SSPCON1bits.SSPM1 = 0;
SSPCON1bits.SSPM0 = 1;
//Em modo 'master' o SSPOV não é activado.
SSPCON1bits.SSPOV = 0;
//Para entretanto não haver comunicação por SPI com o J60.
ChipSelect=1;
//Activa o serial port e configura os pinos SCK, SDO, SDI e SS
SSPCON1bits.SSPEN = 1;
...
```

Configuração do controlador ethernet

Após alimentar o ENC28J60 é importante ver quando o oscilador estabiliza com os parâmetros normais de funcionamento.

```
/*
Esperar que o ENC28J60 Fique pronto para trabalhar espera que o LSb do
ESTAT fique a '1' CLKRDY. No caso deste registo não existe a
preocupação de seleccionar o banco onde se encontra, porque pertence a
um conjunto de 5 registos que se encontram em todos os bancos com o
mesmo endereço a par dos registos ethernet: EIE; EIR; ECON1 e ECON2.
*/
      while(!LerRegETH((byte)ESTAT) & 0b00000001);
```

A partir deste momento o controlador está pronto a funcionar em conformidade com as especificações tendo o oscilador estabilizado.

Nota: para aceder a este registo (ESTAT), assim como aos registos EIE, EIR, ECON1 e ECON2, não existe a necessidade de mudar de banco, uma vez que estes registos

existem nos quatro bancos de registos de controlo do ENC28J60 com o mesmo endereço.

A função LerRegETH serve para ler a informação contida em registos ethernet, i.e. iniciados por 'E' e está definida da seguinte forma:

Ler registo ethernet

Esta é uma função criada com o objective de aceder a um registo ETHernet, com o endereço 'address', que é o seu único argumento e trata-se de uma variável tipo 'byte', e devolver o seu valor.

Para que o PIC aceda ao ENC28J60 tem de comunicar através de SPI. Para tal é necessário em primeiro lugar colocar a '0' o pino de 'Chip Selection' (!CS) do controlador de ethernet, para que este esteja atento e responda aos dados que a ele chegam.

```
byte LerRegETH(byte address)
{
      STAT.SPItempRX = 0;
//coloca o CS do ENC28J60 a '0' para que este fique à escuta
      ChipSelect=0;

/*
Por uma questão de garantia, o procedimento da leitura por SPI é feita
e refeita caso se verifique que aconteceu colisão.
*/
      do
      {
/*
se este bloco tenha de ser repetido devido a colisão, baixa-se a
respectiva bandeira.
*/
            SSPCON1bits.WCOL=0;
//A bandeira que assinala o final da operação SPI também é posta a '0'
            PIR1bits.SSPIF=0;

/*
No registo SSPBUF é escrito o único Byte que é enviado de cada vez
através da comunicação SPI. No caso da comunicação com o ENC28J60, é
necessário colocar nos três bits mais significativos o chamado
'opcode', que codifica a operação que irá ser executada. Neste caso
trata-se de uma operação de leitura de registo de controlo 'RCR' (de
Read Control Register). Nos 5 bits menos significativos coloca-se o
endereço do registo sobre o qual incidirá a acção. Notar que existem
vários registos com o mesmo endereço em bancos diferentes, sendo
portanto necessário antes de executar a função especificar em qual
banco de memória se está a trabalhar.
*/
            SSPBUF = RCR | address;
//espera que o SSPBUF seja transmitido.
            while(!PIR1bits.SSPIF);
//por fim verificar se houve colisão e em caso afirmativo repetir.
      }while(SSPCON1bits.WCOL);

/*
A comunicação por SPI tem a particularidade de sempre que o 'Master'
envia um byte recebe outro byte de volte. Como o buffer é composto por
um único registo comum à transmissão e recepção, é necessário sempre
ler este registo para libertar o buffer, mesmo que a informação
```

```
recebida não tenha qualquer relevância. Portanto o conteúdo do SSPBUF
é lido e atribuído à variável STAT.SPItempRX.
*/
      STAT.SPItempRX = SSPBUF;
/*
Agora para que o controlador de ethernet devolva o valor contido no
respectivo registo tem de receber primeiro um byte (cuja informação é
irrelevante) por parte do PIC.
*/
      do
      {
            SSPCON1bits.WCOL=0;
            PIR1bits.SSPIF=0;
            SSPBUF = 0;
            while(!PIR1bits.SSPIF);
      }while(SSPCON1bits.WCOL);


      STAT.SPItempRX = SSPBUF;
      PIR1bits.SSPIF=0;
//Coloca o CS do ENC28J60 a '1' para fechar a comunicação
      ChipSelect=1;
//Devolve o valor pretendido
      return STAT.SPItempRX;
}
```

Após a espera da estabilização do oscilador, é preferível dar ainda mais cerca de 1ms por precaução.

```
…
for ( STAT.intTemp=0; STAT.intTemp<50000; STAT.intTemp++);
…
```

Configuração da memória de 'buffer' para recepção

O ENC28J60 tem uma memória de buffer de 8Kbytes, dentro da qual se programa um espaço destinado à recepção, ficando a restante memória reservada para o buffer de transmissão.

Para programar a memória do buffer de recepção, tem de se atribuir o endereço de início e de fim aos registos ERXST e ERXND respectivamente, que se encontram no banco0.

O espaço escolhido é compreendido entre o endereço 00h e a posição 0fffh, deixando espaço para um buffer de transmissão capaz de comportar um pacote com as maiores dimensões possíveis que respeitem a especificações IEEE 802.3.

A documentação aconselha à escolha do início de escrita de um pacote na memória de buffer (recepção e transmissão) num endereço par. Também é importante que o processo da configuração da memória do buffer de recepção seja efectuado com o bit de permissão de recepção (ECON1.RXEN) desactivo.

```
…
//mudar para o banco 0, onde estão os registos a operar de seguida
BankSel(0);

//atribuir o valor '0x0000' ao registo ERXST.
EscreverReg((byte)ERXSTH,0x00);
EscreverReg((byte)ERXSTL,0x00);
```

```
//Fim da memória de recepção com o valor '0x0fff' no registo ERXND
EscreverReg((byte)ERXNDH,0x19);
EscreverReg((byte)ERXNDL,0xff);
…
```

Foram utilizadas as funções ‘BankSel’ para mudar para o banco correcto e ‘EscreverReg’.

Selecção de banco

A função ‘BankSel’ serve para mudar de banco, para o valor colocado em argumento (colocar um valor entre 0 e 3) e está definida da seguinte forma:

```
void BankSel(byte banco)
{
/*
Apaga os bits BSEL(1:0), que são os LSb do registo ECON1 responsáveis
pela selecção do banco
*/
BFCReg((byte)ECON1, 3);
//coloca a '1' os bits correspondentes ao banco pretendido.
BFSReg((byte)ECON1, banco);
}
```

Que por sua vez utiliza as funções de ‘bit field clear’ e ‘bit field set’, mostradas de seguida:

Apagar e elevar a ‘1’ bits de um registo

O argumento desta (address) é o endereço do registo a ser operado e os bits a '1' do byte dados irão marcar os bits do registo endereçados a serem apagados.

```
void BFCReg(byte address, byte dados)
{
      byte dummy;
      ChipSelect=0;
      do
      {
            SSPCON1bits.WCOL=0;
            PIR1bits.SSPIF=0;
//opcode para bit field clear seguido do endereço
            SSPBUF = BFC | address;
            while(!PIR1bits.SSPIF);
      }while(SSPCON1bits.WCOL);

      dummy = SSPBUF;
      do
      {
            SSPCON1bits.WCOL=0;
            PIR1bits.SSPIF=0;
//envia os bits a apagar (os que estiverem a '1')
            SSPBUF = dados;
            while(!PIR1bits.SSPIF);
      }while(SSPCON1bits.WCOL);

      dummy = SSPBUF;
      PIR1bits.SSPIF=0;
      ChipSelect=1;
}
```

A função de BFSReg é análoga com a diferença do 'opcode' e da acção sobre os bits a '1' enviados no segundo argumento, que em vez de serem apagados são agora elevados.

Escrever num registo de controlo

A função 'EscreverReg' serve para se aceder a um registo de controlo (seja um registo ethernet ou MAC) do ENC28J60, com um determinado endereço ('address' – primeiro argumento) e colocar o valor pretendido (segundo argumento).

```
void EscreverReg(byte address, byte dados)
{
      byte dummy;
      ChipSelect=0;

      do
      {
            SSPCON1bits.WCOL=0;
            PIR1bits.SSPIF=0;
//opcode para escrever no registo de controlo seguido do endereço
            SSPBUF = WCR | address;
            while(!PIR1bits.SSPIF);
      }while(SSPCON1bits.WCOL);

      dummy = SSPBUF;
      do
      {
            SSPCON1bits.WCOL=0;
            PIR1bits.SSPIF=0;
            SSPBUF = dados;
            while(!PIR1bits.SSPIF);
      }while(SSPCON1bits.WCOL);

      dummy = SSPBUF;
      PIR1bits.SSPIF=0;
      ChipSelect=1;
}
```

Configuração de registos de controlo ethernet

A última função descrita é fundamental para a configuração dos registos de controlo do ENC28J60, que são responsáveis pelo funcionamento deste controlador.
De seguida é descrita a forma como é utilizada na configuração dos registos de controlo ethernet e com que propósito.

```
…
/*
O bit CSUMEN do registo ECON1 é activado permitindo que a DMA calcule
automaticamente checksums. Também é activado o bit RXEN do mesmo
registo, para permitir escrever no buffer de recepção pacotes
consoante a configuração dos filtros. ATENSÃO que este bit ECON1(2) só
deve ser activado após se ter determibado a memória de bufer de
recepção.
*/
BFSReg((byte)ECON1,0b00010100);
```

```
/*No registo ECON2 é activado o bit AUTOINC permitindo um incremento
automatico nos ponteiros ERDPT e EWRPT quando são lidos e escritos
(respectivamente) registos na memória de buffer.
*/
BFSReg((byte)ECON2,0b10000000);

/*
O registo ERXFCON é responsável pela configuração do filtro de
recepção. Os seguintes bits são activados com o objectivo de:
descartar pacotes que não tenham o MAC address do ENC28J60 como
endereço de destino; descartar pacotes que não cumpram todos os
filtros activados; descarta pacotes com CRC inválido
BFSReg((byte)ERXFCON,0b11100000);
…
```

Configuração de registos MAC

O próximo passo é a configuração dos registos MAC (medium access controller).

O primeiro passo é mudar para o banco2, onde se encontram os registos MAC a programar com excepção dos registos de configuração do MAC address (que se encontram no banco 3).

```
…
BankSel(2);

/*
No registo MACON1, é necessário colocar a ‘1’ o bit MACON1.MARXEN para
activar a recepção de frames pelo MAC.
*/
BFSReg((byte)MACON1,0b00000001);

/*
Para que o pacote a enviar cumpra os critérios 802.3 tem de ter no
mínimo 60bytes, acrescido de 4bytes do campo CRC.
Para que isso aconteça sempre, o registo MACON3 é configurado para que
quando o pacote não tenha tamanho suficiente, lhe seja adicionado
automaticamente um ‘padding’ (i.e. completar com ‘uns’ os bytes que
fatam).
*/
EscreverReg((byte)MACON3,0b10110010);

/*
Também para respeitar as especificações 802.3 se deve activar o bit
DEFER do registo MACON4.
*/
BFSReg((byte)MACON4,0b01000000);


/*
Nos registos MAMXFL é colocado o comprimento máximo permitido para os
frames a ser transmitidos ou recebidos. O valor escolhido foi 1518
(05EEh) porá não criar constrangimentos em termos de tamanho a pacotes
considerados pacotes ethernet válidos.
*/
EscreverReg((byte)MAMXFLH,0x05);
EscreverReg((byte)MAMXFLL,0xee);

/*
Os valores introduzidos nos registos seguintes estão de acordo com o
aconselhado no datasheet do ENC28J60, também em função de este estar a
operar em modo full-duplex.
*/
EscreverReg((byte)MABBIPG,0x15);
EscreverReg((byte)MAIPGL,0x12);
```

```
/*
Finalmente a configuração dos registos do source MAC address definidos
num header como 00:04:a3:00:00:01. É importante não esquecer que estes
registos estão num banco diferente. Antes de configura-los é
necessário mudar para o banco 3.
*/
BankSel(3);
EscreverReg((byte)MAADR1,MAC1);
EscreverReg((byte)MAADR2,MAC2);
EscreverReg((byte)MAADR3,MAC3);
EscreverReg((byte)MAADR4,MAC4);
EscreverReg((byte)MAADR5,MAC5);
EscreverReg((byte)MAADR6,MAC6);
```

Configuração dos registos PHY

Ao contrário dos registos de controlo ETH, MAC e MII ou da memória de buffer, os registos PHY não são directamente acessíveis por SPI. Os registos têm de ser acedidos por meio de um conjunto especial de registos de controlo MAC, que implementam 'media independent interface managment' (MIIM).

Para escrever num registo PHY é necessário escrever o seu endereço no registo MIREGADR e colocar o valor em MIWRL (em primeiro) e só depois em MIWRH, isto porque logo após se escrever neste registo se inicia automaticamente a transacção MII. Por fim basta esperar que o bit BUSY, bit 0 do registo MISTAT volte a '0' para ter a certeza que a operação está completa.

```
/*
Para o controlador ethernet funcionar em modo ful-duplex o bit
PHCON1<PDPXMD> tem de estar a '1'.
*/
EscreverRegPHY((byte)PHCON1, 0b00000000, 0b00000001);
/*
Para a configuração dos LEDs são programados os bytes do registo
PHLCON. Para que o LEDA (verde) mostre Link status e o LEDB (amarelo)
mostre a actividade de transmissão, a configuração é a seguinte
*/
EscreverRegPHY((byte)PHLCON, 0b00010010, 0b00110100);
```

Escrever num registo PHY

Os registos PHY do ENC28J60 têm duas paerticularidades: são compostos por dois bytes e não podem ser acedidas directamente como os registos anteriormente referidos. Assim uma função para escrever num registo PHY tem de ter como argumentos: o endereço; o valor a colocar no byte ‘high’ e o valor a colocar no byte ‘low’. O endereço será escrito num registo de controlo MIIM que servirá de interface, designado ‘MIREGADR’. Enquanto que os valores a escrever no registo PHY são atribuídos aos registos MIWRL e MIWRH para escrita dos bytes menos e mais significativo respectivamente.
Para escrever no registo PHY foi utilizada a função ‘EscreverRegPHY’, mostrada de seguida.

```
void EscreverRegPHY(byte address, byte low, byte high)

{
```

```
//mudar para o banco2, onde se encontra o registo MIREGADR
      BankSel(2);
      EscreverReg((byte)MIREGADR, address);
      EscreverReg((byte)MIWRL, low);
      EscreverReg((byte)MIWRH, high);
//mudar para o banco3, onde se encontra o registo MISTAT
      BankSel(3);
/*
Leitura do bit BUSY para esperar que o processo de escrita no registo
PHY se complete.
*/
      while (LerRegMAC((byte)MISTAT) & 0b00000001);

}
```

Esta função por sua vez utiliza a função 'LerRegMAC' para avaliar o bit BUSY, por este bit estar num registo MAC. A única diferença em relação a ler um registo ethernet é que o ENC28J60, antes de enviar a leitura que se pretende envia primeiro um byte 'dummy' sem informação relevante, mas que tem de ser lido antes de se obter o valor que está no registo desejado.

Função para transmitir pacotes ethernet

Esta função tem as seguintes variáveis locais:

```
/*
para guardar a dimenção do frame a ser escrito no buffer de
transmissão.
*/
int txBufLength;

/*
endereço dentro da memória de buffer para o inicio da memória de
transmissão
*/
int TXST = 0x1a00;
/*
onde vai ser colocado o valor do endereço do fim de memoria de
transmissão.
*/
int TXND;

//LSB do TXND
byte TXNDL;

//MSB do TXND
byte TXNDH;

      BankSel(0);
//Programar o ponteiro ETXST
      EscreverReg((byte)ETXSTL,(byte)((TXST<<8)>>8));
      EscreverReg((byte)ETXSTH,(byte)(TXST>>8));
```

Programar o início da memória de buffer de transmissão

Já foi mencionado anteriormente que todo o espaço da memória de buffer que sobra, após a configuração da memória de buffer de recepção fica destinada para a escrita de pacotes a transmitir. Importa portanto garantir que não exista sobreposição.
É necessário indicar onde se inicia o ponteiro de escrita configurando EWRPT. O valor a atribuir será o início da memória de transmissão.

```
//Programar o ponteiro EWRPT
EscreverReg((byte)EWRPTL,(byte)((TXST<<8)>>8));
EscreverReg((byte)EWRPTH,(byte)(TXST>>8));
```

Escrita do pacote a enviar

Para um pacote ser construído e enviado correctamente, é necessária a escrita na memória de buffer destinada à transmissão de um conjunto de campos devidamente ordenados descritos de seguida.

Byte de controlo

```
//byte de controlo
      EscreverBuffer(controlo);
```

Onde o byte de controlo está definido com o valor ‘0b00000110’. O byte de controlo permite que se alterem algumas configurações de transmissão em relação ao que está definido no registo ‘MACON3’. Neste caso o byte de controlo está configurado para não existir alteração.

Escrever um byte na memória de ‘buffer’

Foi utilizada a função ‘EscreverBuffer’ para escrever um byte na memória de ‘buffer’ que é definida da seguinte forma:

```
byte dummy;
//coloca o CS do ENC28J60 a '0' para que este fique à escuta
ChipSelect=0;

//opcode para bit field clear seguido argumento correspondente
SSPBUF = WBM;

//espera que o SSPBUF receba (BF fica a '1')
while(!SSPSTATbits.BF);

//lê o dummy byte, para o BF voltar a '0'
dummy = SSPBUF;

//envia o byte a ser escrito
SSPBUF = dados;
while(!SSPSTATbits.BF);
dummy = SSPBUF;

//coloca o CS do ENC28J60 a '1' para fechar a comunicação
ChipSelect=1;
```

MAC address de destino

Neste campo é colocado o MAC address a que o pacote se destina. Pode ser especifico para um determinado dispositivo, ou para todos os dispositivos em geral ligados á mesma rede (broadcast). Este campo é constituído por seis bytes, e o pedaço de código seguinte exemplifica a transmissão de um pacote Broadcast.

```
//MAC address de destino (braodcast: FF-FF-FF-FF-FF-FF)
EscreverArrayBuffer(destMACaddr,sizeof(destMACaddr));
```

Escrever um ‘array’ na memória de buffer

A função ‘EscreverArrayBuffer’ foi utilizada para automaticamente escrever um ‘array’ na memória de buffer. O primeiro argumento é um ponteiro para o ‘array’, o segundo argumento é o tamanho do referido ‘array’ e está escrita da forma a seguir descrita:

```
void EscreverArrayBuffer(byte *val, unsigned int len)
{
      byte dummy;
ChipSelect=0;
PIR1bits.SSPIF=0;

//opcode para bit field clear seguido argumento correspondente
      SSPBUF = WBM;

//espera que o SSPBUF receba (BF fica a '1')
      while(!SSPSTATbits.BF);

//lê o dummy byte, para o BF //voltar a '0'
      dummy = SSPBUF;

// envio de dados
      while(len)  //enquanto existirem bites a enviar, corre o ciclo
      {
// inicia o envio do byte
            SSPBUF = *val;

// incremento do ponteiro que aponta para //o byte a enviar
            val++;

// decremento do numero de bytes a enviar
            len--;

//espera que o SSPBUF receba (BF fica a '1')
            while(!SSPSTATbits.BF);
            dummy = SSPBUF;
      }

      ChipSelect=1;     //coloca o CS a '1' fecha a comunicação
}
```

MAC address de origem

Neste campo de seis bytes o 'array' sourceMACaddr está configurado com o valor de defeito da placa de demonstração da microchip: 00-04-a3-00-00-00, em hexadecimal. Este é o MAC address do microcontrolador, onde o programador tem a 'liberdade' de escolher, dentro de valores que não existam na rede.

```
      EscreverArrayBuffer(sorceMACaddr,sizeof(sorceMACaddr));
```

Campo Type/Length

Este campo de dois bytes tem duas funcionalidades possíveis. Quando se trata de um pacote ethernet puro e simples, a função é de referir o comprimento total do pacote em número de bytes (sem contar com o padding). Quando se trata de um tipo de protocolo específico, este campo descreve de que tipo se trata. Por exemplo caso se trate de um pacote IP está estandardizado que neste campo deve estar o valor 0x0800. Saber o tipo de protocolo é fundamental para que quem receba o pacote saiba exactamente como o desencapsular e que informação retirar dos campos subsequentes.

```
//prenchimento do campo Type/Length
      EscreverArrayBuffer(typeLength,sizeof(typeLength));
```

Campo de dados

Este campo encerra os campos mínimos necessários para a transmissão de um pacote ethernt. Nele segue todo o conjunto de dados que se pretenda transmitir. Caso se pretenda em cima do protocolo ethernet utilizar outros tipos de protocolos é necessário adicionar campos adicionar que estes exijam.

```
//escrita dos dados a transmitir por ethernet.
      EscreverArrayBuffer(dados,sizeof(dados));
```

Programar o fim da memória de buffer de transmissão

Antes de enviar o pacote é necessário definir o fim da memória de buffer de transmissão através da configuração do ponteiro ETXND.

```
//Programar o ponteiro ETXND
txBufLength = sizeof(destMACaddr)+sizeof(sorceMACaddr)+
              sizeof(typeLength)+sizeof(dados);
      BankSel(0);

/*
o endereço do fim da memoria de buffer de transmissão é obtido somando
ao endereço do seu inicio o valor do numero de bytes que foram
entretanto escritos no buffer, contabilizados na variável txBufLength.
*/
TXND = TXST+txBufLength;

      EscreverReg((byte)ETXNDL,(byte)((TXND<<8)>>8));
      EscreverReg((byte)ETXNDH,(byte)(TXST>>8));
```

Transmitir o pacote

Para se iniciar activa-se o bit ECON1.TXRTS

```
/*
iniciar o processo de transmissão por activação do respectivo bit no
registo de controlo, ECON1, do controlador ethernet ENC28J60.
*/
      BFSReg((byte)ECON1, 0b00001000);
      while(LerRegETH((byte)ECON1) & 0b00001000);
```

Devendo-se por fim aguardar que o mesmo bit accionado para dar inicio á transmissão fique a '0', assinalando o final do processo de envio. Período durante o qual nenhum dos registos associados à transmissão de pacotes por ethernet deve ser alterado.

Comunicações por socket

Existem dois tipos de "socket" (conexão) que podem ser estabelecidos. Através dos protocolos da camada de transporte TCP (Transmission Control Protocol) e UDP (User Datagram Protocol), ambos sobre a camada "Internet Protocol" (IP) que por sua vez está sobre a camada ethernet anteriormente mencionada.
Uma característica das conexões socket é a existência de um servidor (entidade passiva) e de um cliente (entidade activa que toma a iniciativa de iniciar a comunicação). Cada uma destas entidades aquando do estabelecimento do socket destina uma porta específica para essa ligação.

Camada IP (Internet Protocol)

Para que ao receber um pacote ethernet se saiba que se trata do tipo IP, tem de constar no campo do cabeçalho ethernet "type/length" o valor em 0800h.
Este protocolo existe directamente acima do protocolo ethernet e tem as seguintes características:

Permite configurar o tipo de serviço, isto é determinar algumas características de como o pacote enviado circula na rede, em termos de atraso, prioridade fiabilidade e outras características reservadas a aplicações futuras;

Atribui uma identificação de indexação ao pacote;

Permite que o pacote seja fragmentado;

Confere identificação IP que permite que um dispositivo com um único MAC address tenha múltiplas identificações possíveis na rede com um campo de quatro bytes;

Aceita outros protocolos em cima de si.

Protocolo TCP

O protocolo TCP existe sobre o protocolo IP e para que ao receber um pacote se saiba que se trata do tipo TCP, tem de constar no campo do cabeçalho IP “Protocol” o valor 6. Este protocolo é de transporte e tem como principais características a existência de confirmação da integridade de cada pacote enviado e de estabelecer conecção por meio de socket.

Para que estabeleça um socket TCP, o cliente deve enviar um pacote a assinalar a intenção de estabelecer ligação a um servidor. Por sua vez o servidor deve responder com um pacote a assinalar que recebeu o convite e que também pretende a referida ligação. A ligação só é estabelecida após o servidor receber uma confirmação de que o cliente recebeu a sua resposta. A partir daí pode haver envio de pacotes de dados entre ambos, sendo que existe a necessidade de cada mensagem enviada receber uma mensagem de “acknowlegment” (confirmação) em resposta, e cada mensagem de dados recebida ser verificada e reenviar um “acknowlegment” em resposta.

Quando uma das entidades pretende encerar o socket, envia um pacote de finalização o qual para não fugir à norma deve ser confirmado pela outra entidade. A entidade que recebe o pacote de finalização, deve tratar de transmitir os últimos pacotes que ache necessário antes fechar a comunicação. Assim que isso aconteça deve enviar também uma mensagem de finalização assinalando que se encontra pronta para encerrar o socket ficando a aguardar o respectivo ‘acknowlegment’ para poder terminar o socket com a garantia que a outra parte também o fez.

O PIC foi programado para funcionar como servidor, respeitando a descrição protocolar acima descrita e ilustrada no seguinte fluxograma.

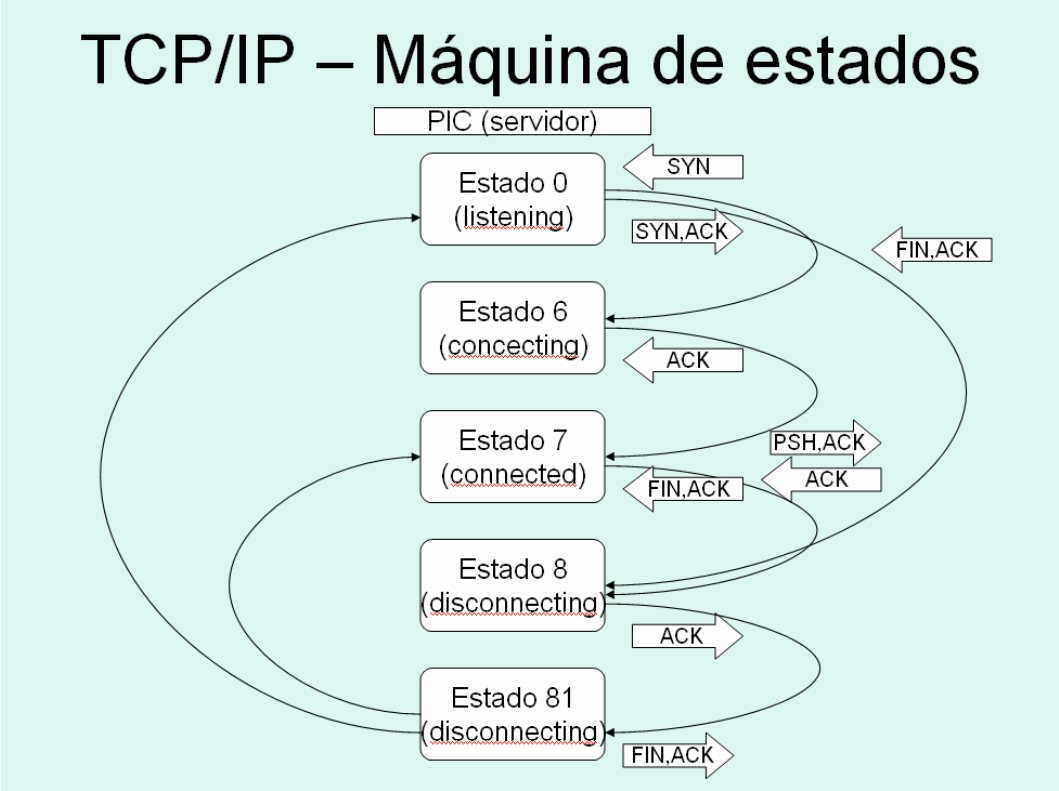

Fig. 5: Fluxograma ilustrativo da máquina de estados utilizada para a programação do microcontroador para comunicar pelo protocolo TCP/IP.

Protocolo UDP

O protocolo UDP existe sobre o protocolo IP e para que ao receber um pacote se saiba que se trata do tipo UDP, tem de constar no campo do cabeçalho IP “Protocol” o valor 17.

O protocolo UDP é muito mais simples de implementar comparativamente com o TCP. Também se trata de um protocolo de transporte, também forma socket para estabelecer ligação, mas não exige a confirmação dos pacotes recebidos. Desta maneira torna-se menos rigoroso, mas mais rápido. Ideal para “streaming” de dados, onde as taxas de transmissão são muito elevadas, e não é critico acontecer falha em algumas transmissões (e não é interessante estar a corrigir pacotes desactualizados).

Recepção do pacote

Para permitir a recepção de pacotes é fundamental que o ‘buffer’ de recepção esteja inicializado, os registos MAC estejam configurados e também o registo ERXFCON de configuração dos filtros de pacotes recebidos.

Para dar permissão à recepção de pacotes é necessário activar o bit RXEN do registo ECON1. É muito importante que após a activação deste bit não se proceda a alterações nos registos de fim e início do ‘buffer’ de recepção ou alterar a configuração dos registos MAC ou da configuração dos filtros de recepção.

Sempre que um pacote seja recebido é prontamente activado uma ‘flag’ de interrupção para o evento de recepção, bit PKTIF do registo EIR. O programa pode ter conhecimento desta ocorrência através da rotina de atendimento às interrupções ou por verificação regular de deste bit. Os pacotes recebidos vão sendo acumulados no ‘buffer’ de recepção e o seu número contabilizado num registo contador EPKTCNT, sendo decrementado automaticamente à medida que os pacotes vão sendo lidos. Só quando o valor deste registo contador for ‘0’ é que o PKTIF volta automaticamente a ‘0’.

À medida que os pacotes chegam ao controlador e passam os critérios dos filtros de recepção vão sendo escritos no buffer de recepção, precedidos por um campo de seis bytes. Neste campo os dois primeiros bytes são um ponteiro cujo valor é o endereço onde o pacote seguinte irá ser escrito. Os restantes 4 bytes constituem um vector de status recepção. O buffer de recepção funciona de forma circular, i.e., os pacotes vão sendo escritos sequencialmente e quando chega ao endereço final continua através do endereço inicia.

O ponteiro de endereço do pacote seguinte a ser recebido é fundamental para se programar os registos de ponteiros para o endereço de leitura no buffer de recepção para ler o pacote seguinte (ERDPT) e também dos registos ponteiros do endereço do buffer de recepção cuja informação já foi lida (ERXRDPT).
Estes registos têm um papel muito importante. Os ERDPT devem ser programados com o valor do início do buffer de recepção para o primeiro pacote após a inicialização do controlador ou com o valor do endereço do ponteiro de próximo pacote contido no pacote anterior. Á medida que os bytes do pacote vão sendo lidos o controlador

encarrega-se de automaticamente ir incrementando o endereço para facilitara a leitura de todos os bytes do pacote de forma sequencial. O ponteiro ERXRDPT marca dentro do buffer de recepção até onde os pacotes recebidos foram lidos. Esta informação é fundamental para o funcionamento de buffer circular pois permite garantir que os pacotes recebidos não são escritos sobre espaço onde estão pacotes recebidos anteriormente ainda por ler. Caso o hardware tente escrever sobre pacotes não processados o pacote em progresso é abortado. Para se escrever o valor no ponteiro ERXRDPT deve-se em primeiro escrever no seu byte menos significativo (o valor fica armazenado internamente) e só depois no mais significativo (porque desencadeia o carregamento dos dois bytes para o registo simultaneamente.

Após cada pacote recebido ser completamente processado deve-se activar o bit PKTDEC do registo ECON2 para que o registo EPKTDEC (anteriormente mencionado) seja decrementado uma unidade. A falta deste procedimento tem como consequência um incremento constante do registo EPKTDEC que ao atingir o valor 0xFF resulta na abortagem da recepção de novos pacotes acompanhado da activação do byte de erro de recepção RXERIF do registo EIR.

A função para a recepção de um pacote é programada de forma a fazer um correcto processamento do pacote recebido, fazer interpretação dos tipos de protocolos envolvidos, armazenar os dados necessários e dar resposta adequada respeitando a máquina de estados.

Circuitos do protótipo

O protótipo foi desenvolvido por módulos com o objectivo de diminuir custos relacionados com melhorias numa parte com funcionalidade específica e também facilitar a inclusão de módulos adicionais para aplicações futuras.

Processamento/controlo e comunicação

Este circuito funciona como uma “placa mãe” onde se conectam os restantes módulos. É composta pelo microcontrolador (master) PIC18F2550, pelo controlador ethernet ENC28J60, respectiva electrónica acessória, conector para o programador MPLAB® ICD2 ligação para uma fonte de alimentação suficiente para garantir 3.3V para alimentar estes integrados, dois conectores para acoplar os circuitos responsáveis pela electroestimulação e pela aquisição dos sinais dos sensores de força e um conector RJ45 para ligar o cabo ethernet.

O PIC escolhido foi o 18F2550 porque com excepção do módulo de comunicação CAN, tem todos os outros módulos disponíveis em microcontroladores de 8-bits da microchip® e por pertencer à única família de PIC com módulo usb. Considerou-se que para efeitos de protótipo seria importante ter o maior leque possível de funcionalidades e módulos de comunicação à disposição no microcontrolador para evitar ter de efectuar migrações.

A comunicação por porta de série chegou a ser ponderada para equipar este circuito, mas foi afastada a possibilidade em função de apresentar conflitos com a comunicação por SPI, esta realmente necessária para a programação do controlador ethernet e dos potenciómetros digitais. Por outro lado a comunicação por porta de série está cada vez mais em desuso (por este próprio motivo também) por possibilitar um numero muito limitado de conecções, ter taxas de transferência de dados muito inferior à das soluções de comunicação alternativas e por não acrescentar nenhuma vantagem sobre estas em termos funcionais embora tenha como principal vantagem a facilidade de implementação.

Outro tipo de comunicação cuja possibilidade de integração foi avaliada foi da comunicação por usb. A principal motivação do estudo desta solução que representou uma fatia de tempo considerável da duração do projecto foi a grande versatilidade do "plug and play", que significa que um dispositivo usb se configura automaticamente quando é ligado a um sistema com controlador de usb em funcionamento. No entanto foi previsto de que a sua implementação e principalmente a programação dos respectivos "drivers" iria consumir quantidade de tempo não compatível com a duração do projecto. Juntamente com a descoberta que o modulo usb do PDA apenas servia para que este funciona-se como periférico, por exemplo para efeitos de sincronização e que não servia para funcionar como controlador usb anfitrião, pôs-se de parte esta solução.

A solução de integrar comunicação por ethernet apresenta o seguinte conjunto de vantagens. Permite taxas de transmissão bastante mais elevadas que por porta de série e mesmo comparativamente com soluções sem fio como bluetoth. Permite a criação de um número elevado de conecções. Utiliza protocolos que permitem que a informação seja transmitida pela Internet facilitando a comunicação a grandes distancia sem grandes custos. Utiliza protocolos de comuns à comunicação por wireless facilitando a futura implementação deste tipo de comunicação. Com recurso a um router permite a comunicação por wireless o que diminui o peso da desvantagem necessitar de utilizar cabo, principalmente numa fase de protótipo. A principal desvantagem reside no tempo requerido para a implementação de raiz.

A utilização de tipos de comunicação sem fios não foi integrada nesta fase do desenvolvimento por se entender que envolveria maior complexidade em termos de circuito acessório e consequente tempo de implementação. Comunicar directamente sem fios não deixa de ser de extrema importância para um produto final. Assim está previsto a sua implementação futura na continuação do trabalho realizado neste projecto, contando com a vantagem de parte da programação do protocolo estar já realizada e da mais valia que consiste o conhecimento adquirido com implementação da comunicação por ethernet.

Existem companhias que disponibilizam soluções de comunicação sem fios, bluetoth e wireless, para integração em sistemas. Estas consistem em módulos com o circuito completo e software prontos a funcionar com pouco custo de implementação, que representa a principal vantagem. Como desvantagens apresentam-se: o elevado custo por unidade. As dimensões físicas podem implicar aumento no tamanho do equipamento final. Não dá conhecimento de como se faz.
O principal contraponto a este tipo de soluções é, que aprendizagem de implementar de raiz tem gastos associados principalmente em termos de tempo, mas depois de se saber

fazer pode-se ajustar melhor o resultado final ás necessidades e o custo por unidade é muito mais baixo.

A seguir é mostrado o esquema, circuito e imagem da placa pronta.

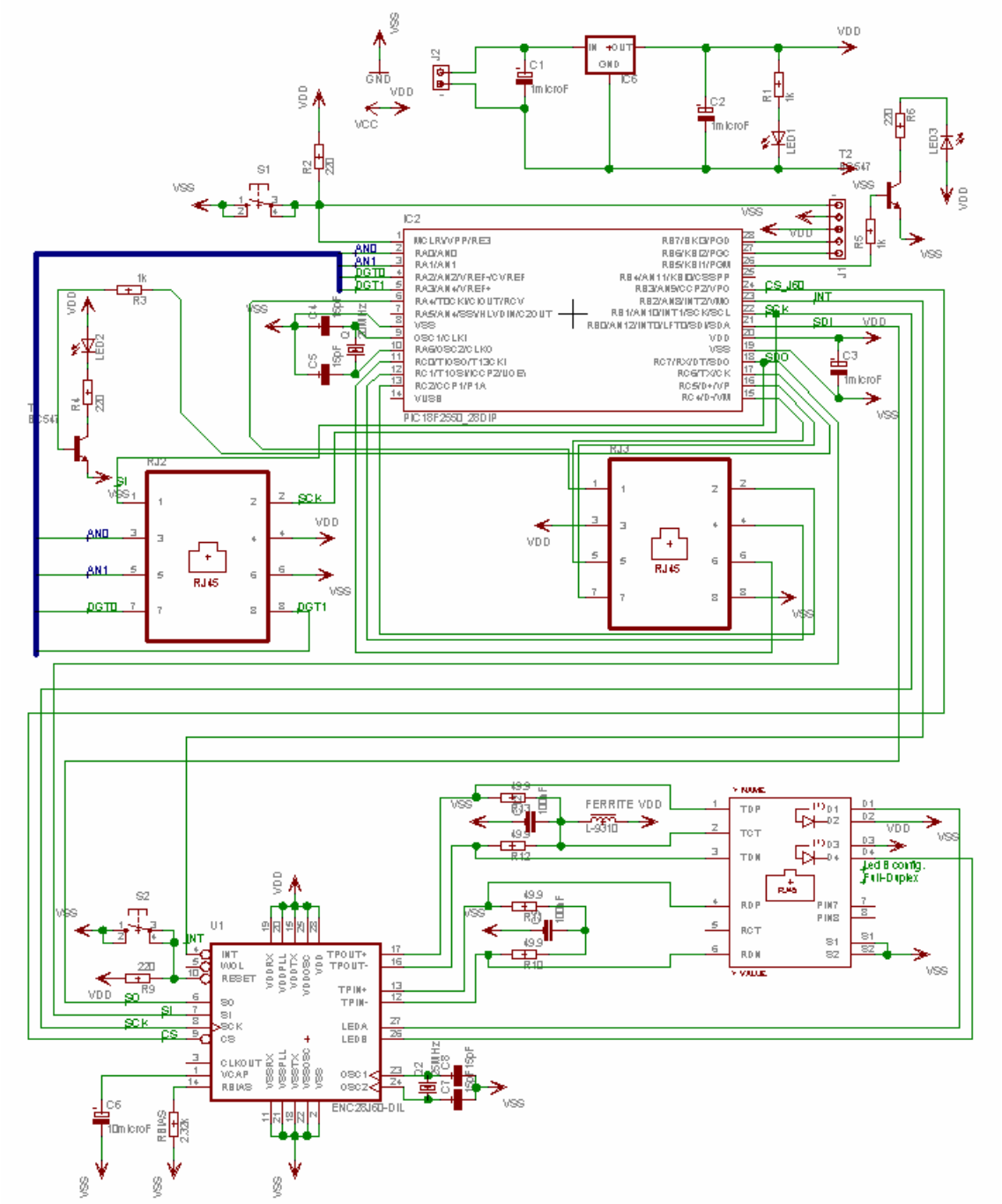

Fig. 6: Esquema do circuito da placa de controlo (PIC) e comunicação (controlador ethernet ENC28J60).

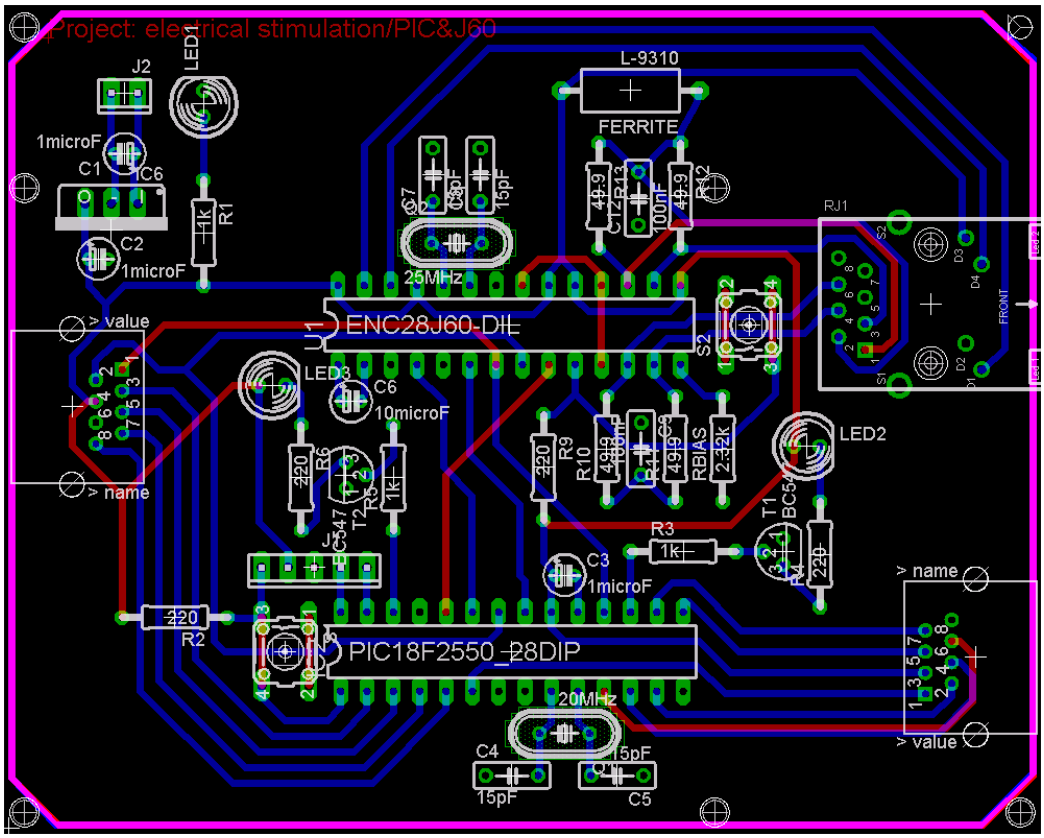

Fig. 7: Circuito da placa de controlo e comunicação

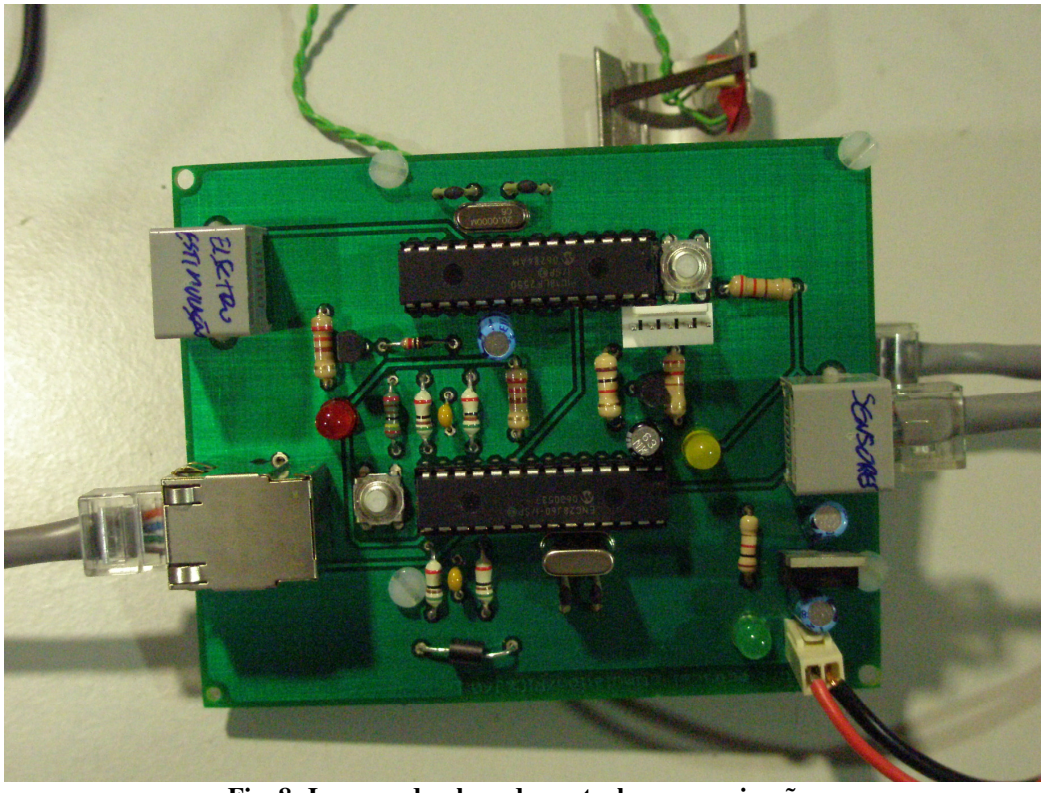

Fig. 8: Imagem da placa de controlo e comunicação.

Aquisição dos sinais dos sensores de força

Com o desenvolvimento de sensores de força pretende-se avaliar a capacidade motora de pacientes com paralisia parcial, resultante de lesão na espinal-medula. Será utilizado em exercícios que possam evidenciar as limitações e ajudem a monitorar a evolução como resposta ás terapias aplicadas. Pode também ser utilizado na própria terapia associado a exercícios de resposta visuo-motora. Em qualquer dos casos pode ser aplicada conjuntamente electroestimulação e tentar perceber o seu efeito.

O sensor desenvolvido tem a possibilidade de ficar solidário com uma parte do corpo do paciente para permitir maior liberdade no tipo de exercício a desempenhar.

O princípio de funcionamento do sensor baseia-se no estensómetro. O estensómetro é um sensor resistivo, i.e., é um filamento com uma determinada resistência que tem e que se encontra numa configuração em que praticamente todo e seu comprimento se encontra numa direcção, a direcção de deformação. A quantidade de deformação nesta direcção vai provocar aumento do seu comprimento e diminuição do seu diâmetro e consequentemente aumento da sua resistensia.

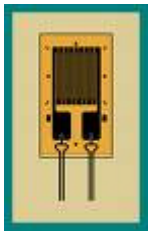

Fig. 9: Ilustração de um estensómetro.

Quando se faz passar corrente no estensómetro, uma deformação ao longo da sua direcção de deformação por acção de uma força vai resultar no aumento da queda de tensão nos terminais do estensómetro. Assim a razão entre a queda de tensão após e antes da deformação dá, dentro de certos limites de deformação uma quantificação da força responsável pela deformação.

Como o estensómetro é muito frágil e não oferece grande resistência à deformação foi necessário encontrar um material com propriedades elásticas apropriadas para o colar de maneira a que se deforme convenientemente para medir uma gama de forças pretendida. A deformação deste material depende da distância dos seus apoios.

No sensor desenvolvido os apoios são rasgados numa superfície de aço hemicilíndrica, com 26mm de diâmetro e 1mm de espessura, destinada a ser abraçada aos dedos, funcionando como dedeira. O material onde o estensómetro foi colado é uma lâmina de aço com 3mm de largura e 0.5mm de espessura e que respondeu de forma satisfatória aos testes efectuados.
A pressão do dedo é feita directamente sobre a lâmina de aço onde o estensómetro está colado.

O circuito típico para montar os estensómetro é a ponte de Whetstone, ilustrado na figura seguinte.

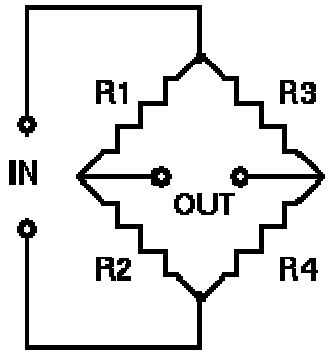

Fig. 10: Esquema de uma ponte de Whetstone.

A ponte de whetstone é alimentada por uma tenção IN e a medida na saída OUT vai depender da relação entre o valor das quatro resistências R1, R2, R3 e R4. normalmente a valor da tensão da ponte é muito reduzida então utiliza-se um amplificador de instrumentação para amplificar esta diferença de tenção para valores que tenham a maior excursão possível que o PIC consiga medir por conversão analógica digital.
No circuito concebido, as resistências R1 e R3 correspondem a potenciómetros digitais e as resistências R2 e R4 correspondem a estensómetros, dos quais R2 é o estensómetro activo (sensor) e R4 é um estensómetro que permanece indeformável. O facto de se utilizar dois estensómetros e dois potenciómetros serva para garantir simetria na ponte. Desta forma qualquer variação numa das resistências em face a alterações ambientais é compensada pela simétrica não se verificando desequilíbrio na ponte por esse motivo. Na inicialização os estensómetros são programados para que a ponte fique com um desequilíbrio inicial desejado (sempre medido pelo PIC). Durante o funcionamento todas as resistências permanecem fixas excepto o estensómetro do sensor. Este ao variara sua resistência por acção de uma força vai desequilibrar a ponte e a respectiva diferença de potencial é lida pelo PIC. Mediante prévia calibração o PIC fica a conhecer a intensidade da força aplicada no estensómetro.

O contacto com pacientes com lesões medulares no Centro de Medicina de Reabilitação da Região Centro - Rovisco Pais permitiu perceber que as capacidades motoras entre pacientes é bastante variável, também o mesma pessoa pode aumentar a sua força durante o período de fisioterapia. Em resposta a este facto o circuito tem um potenciómetro digital como resistência de ganho do amplificador da saída da ponte de whetstone permitindo controlar a sensibilidade do sensor, conferindo-lhe a versatilidade de servir para medir uma ampla gama de forças. Por outro lado a alteração da sensibilidade do sensor pode ter utilidade na realização de determinados estudos.

As imagens seguintes mostram o esquema eléctrico, o circuito e a placa da electroestimulação.

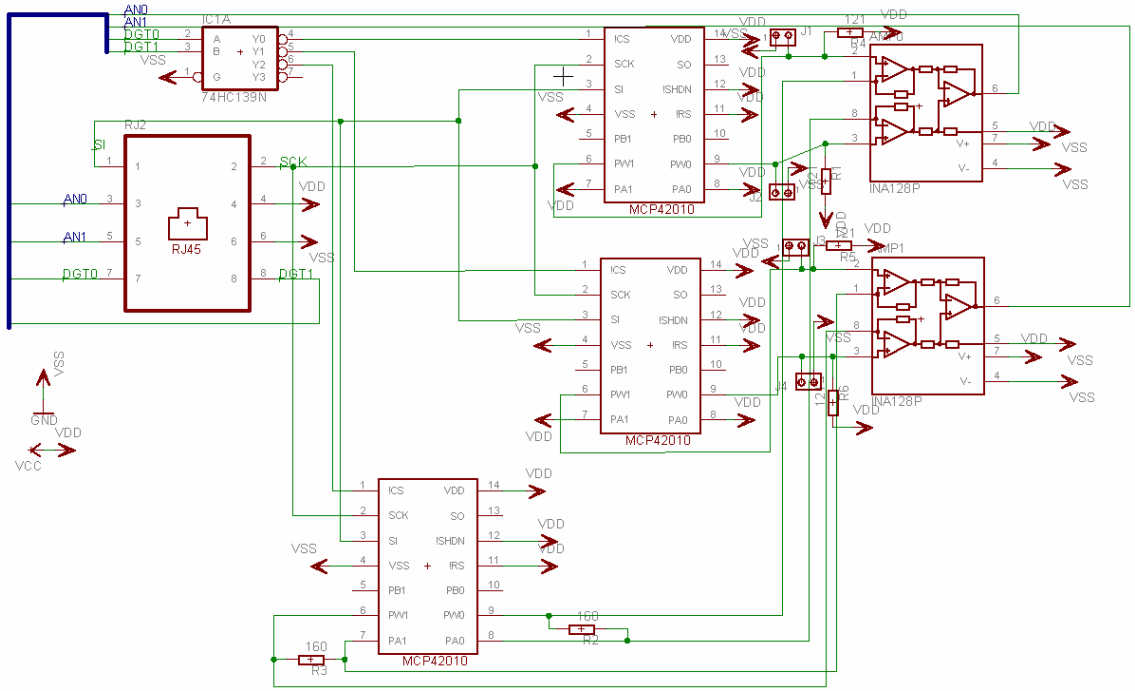

Fig. 11: Esquema eléctrico do circuito dos sensores de força.

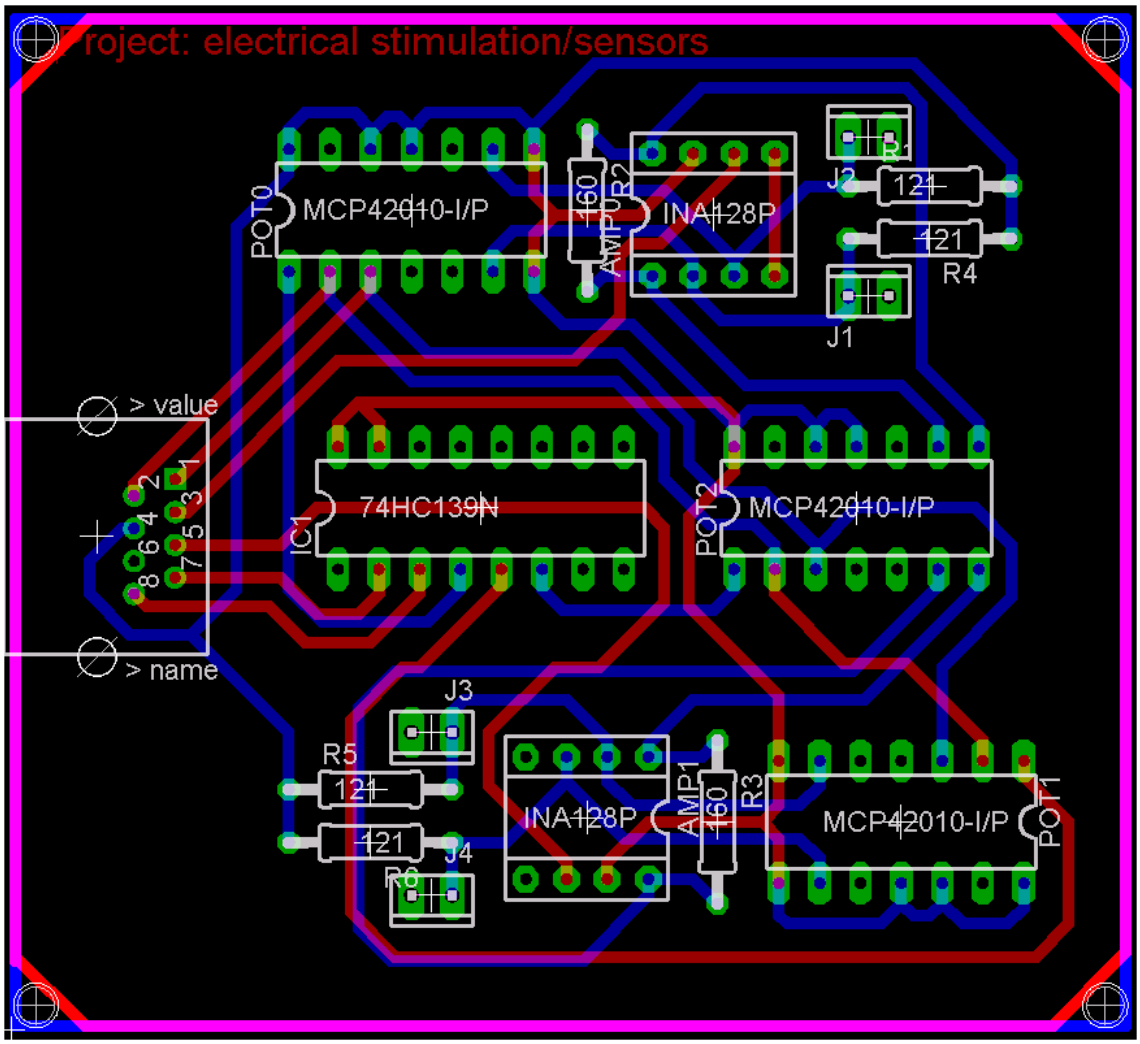

Fig. 12: Circuito dos sensores de força.

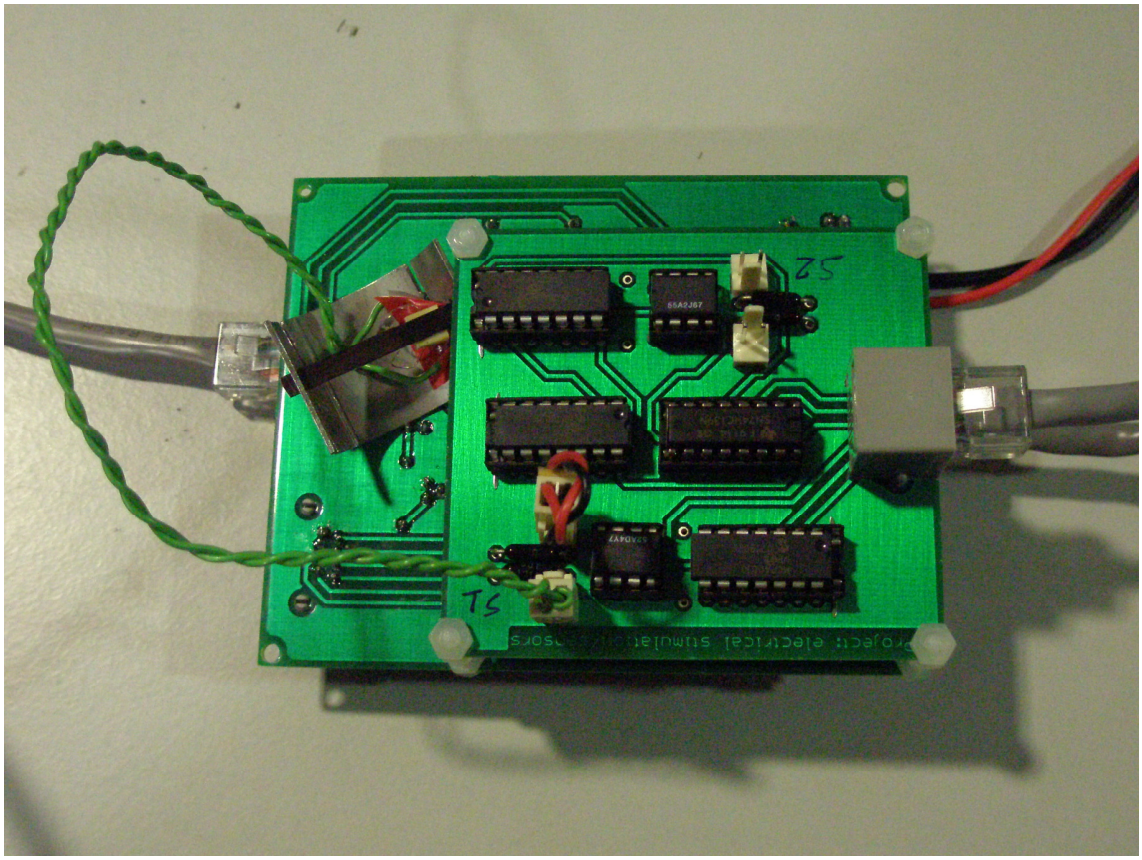

Fig. 13: Imagem da placa dos sensores de força com um sensor.

Electroestimulação

O objectivo da electroestimulação neuro-muscular é a obtenção de uma contracção muscular pela aplicação de uma diferença de potencial eléctrico aos terminais de dois eléctrodos para que se crie uma corrente entre estes de maneira a provocar excitação das células nervosas e musculares. Para que o resultado corresponda ao pretendido e de maneira a que não resulte nenhum tipo de lesão para os tecidos envolvidos, é importante ter conhecimento da fisiologia das células excitáveis e como respondem a estímulos eléctrico com diferente natureza.

Quando uma célula excitável é sujeita a um estímulo químico, eléctrico ou físico pode sofrer um aumento da permeabilidade da sua membrana aos iões de Na^+. Este aumento de iões positivos contribui para a diminuição da carga negativa da membrana (despolarização). A partir de um determinado valor critico o aumento da permeabilidade da membrana ao Na^+ dispara de forma que o potencial de membrana atinge rapidamente (cerca de 0.5ms) valores entre +25 a 35mV.

A despolarização inicial da membrana também aumenta a permeabilidade aos iões K+, responsável pela hiper polarização da membrana (potencial mais negativo). No entanto este aumento da permeabilidade tem o seu pico num instante temporal um pouco posterior ao dos iões Na+, voltando a aproximar o potencial da membrana ao potencial de equilíbrio para o potássio. o resultado é uma alteração muito rápida no potencial da membrana, conhecida por **potencial de acção**.

A concentração de iões volta à configuração original por acção do transporte de iões de Na^+ para fora e dos iões de K^+ para dentro da célula através de bombas de sódio-potássio.

Uma célula excitável pode adaptar-se (acomodar-se) ao estímulo, passando a gerar potencial de acção com maior dificuldade ou chegando mesmo a perder essa capacidade. A acomodação a estímulos depende do tipo de célula. De uma forma geral as células musculares acomodam menos que as nervosas.

A ocorrência de um potencial de acção numa determinada região da célula pode desencadear novo potencial numa região adjacente. Este efeito permite que o sinal em resposta a um estímulo viaje ao longo de toda a célula. Em células nervosas onde o estímulo tem de se propagar por distâncias muito elevadas, o revestimento da membrana dos seus axónios por myelina, permite ao sinal propagar-se de forma saltatória, diminuindo a atenuação e aumentando a velocidade.

Com recurso a um par de eléctrodos externos pode-se aplicar correntes eléctricas sobre membranas excitáveis. Esta corrente poderá criar despolarização na membrana e caso atinja um valor crítico de despolarização ("threshold") irá desencadear um potencial de acção. No entanto para que o estímulo produza uma despolarização na membrana até ao threshold depende do tipo e estado de célula, amplitude e duração da sua aplicação.

As membranas têm propriedades resistivas (dificultam o fluxo de cargas) e capacitivas (requerem tempo para alterar a sua carga). Para fibras com elevada resistência eléctrica é necessário que o estímulo tenha uma amplitude acima de um determinado nível e a duração da sua aplicação depende directamente de sua resistividade e capacidade.

Células nervosas (fibras com menor diâmetro) requerem estímulos com maior amplitude que as células musculares (com maior diâmetro). Por outro lado células musculares demoram mais a carregar que as células nervosas.

Períodos refractários de uma célula excitável correspondem ao tempo em que estas estão fisiologicamente inibidas/impedidas de realizar potenciais de acção, devido a recuperação da excitabilidade.

Normalmente após um potencial de acção segue um período de tempo (típico de 0.5ms) em que membrana simplesmente é incapaz de gerar novo potencial de acção (**período refractário absoluto**). Após este tempo durante um período de 0.5ms adicionais a membrana é capaz de criar potenciais de acção mas só mediante estímulos com uma amplitude acima do normal (**período refractário relativo)**. Só então a membrana recupera completamente a sua excitabilidade e passa a responder de forma típica aos estímulos eléctricos.

Um músculo devidamente enervado responde a estímulos eléctricos contraindo. Normalmente a resposta é evocada por excitação directa das células nervosas, por terem um threshold de estimulação inferior às células musculares. Só quando desenervadas as fibras musculares são directamente excitadas pelas electrostimulação.

Uma sequência de estímulos seguidos a partir de determinada frequência pode ter um efeito aditivo em termos da contracção muscular, caso não exista tempo suficiente para

haver relaxação muscular antes do novo estimulo. Neste, passa a existir fusão na contracção e a tensão muscular resultante é persistente superior à resultante de um estímulo isolado. Estas contracções são designadas de tetânicas ou fundidas.

Numa célula nervosa o estímulo propaga-se ao longo do axónio numa determinada direcção. Sob acção de electroestimulação os potenciais de acção podem mover-se em ambas as direcções.

A forma da onda de corrente descreve muito melhor a resposta de células excitáveis que a forma da onda em tensão. Estes dois tipos de onda diferem pelo facto de estarem ligadas a um sistema complexo de impedância bioelectrico. Ao administrar um sinal controlado em tensão qualquer variação na impedância dos tecidos envolvidos resulta numa alteração da corrente que os atravessa, podendo tem consequências indesejáveis.

Em correntes terapêuticas é muito importante o conceito de fase. Que corresponde ao fluxo de corrente numa direcção durante um determinado tempo. Corrente monofásica varia a sua intensidade mas sempre na mesma direcção em relação a uma referencia de corrente nula. Ao passo que corrente bifásica a corrente existem períodos em que a corrente assume sentidos contrários. Para bifásicas correntes desequilibradas a intensidade de um pulso numa fase difere da intensidade do pulso na outra fase.

A corrente efectiva aplicada aos tecidos é a RMS (Root Mean Square – raiz da média do quadrado) da onda de corrente. É a corrente efectiva que permite determinar a taxa a que a energia está a ser transferida que provoca aquecimento dos tecidos. O valor do RMS da corrente é melhor indicativo da quantidade de estímulo que a simples média. O RMS da intensidade da corrente deve estar dentro dos parâmetros de segurança para evitar lesões nos tecidos.

A modulação da forma da onda pode ser feita de diversas maneiras: modulando a amplitude; modulando a duração da fase ou pulso; modulando a frequência; modulando a temporização da descarga de padrões de pulsos (período de estimulação) e períodos de repouso.

Os três componentes funcionais comuns a todos os estimuladores são: fonte de alimentação; gerador de sinais; amplificador de saída.

Normalmente as fontes de alimentação dos estimuladores são baterias ou a rede doméstica. No entanto para se conseguir as elevadas tensões necessárias para a electroestimulação tem de existir elevação da tensão da alimentação.

A introdução de computadores nos estimuladores, permite substituir os circuitos de electrónica analógica na geração de sinais, tornando os sistemas bastante mais versáteis. Permitem maior liberdade na modelação dos estímulos, contabilização dos pulsos e no controlo das temporizações. Podem ser programados e reprogramados, para alterar as características da onda gerada e garantir que os níveis de segurança são respeitados.

Por fim existe o amplificador de saída cuja função é amplificar a onda que sai do gerador de sinais com base na potência gerada na fonte de alimentação.

Quando ligado ao corpo, que contém impedâncias de origem capacitiva e indutiva, qualquer forma de onda que não seja sinusoidal vai verificar uma deformação entre as ondas de corrente e de tensão.
Usando transformador obtém-se picos de tensão elevada pela relação entre as indutâncias do enrolamento primário e secundário, com a vantagem de conferir isolamento gavânico (mportante para evitar choque).
Outra possibilidade é o recurso a conversores de tensão para corrente, que exigem elevações de tensão, por exemplo através de circuitos com condensadores, e que constituem uma fonte de corrente controlável.

O estimulador deve permitir controlar os seguintes parâmetros: amplitude; “duty cycle”; frequência; duração do pulso.

Em termos de impedância eléctrica os tecidos adiposos, ósseos; e da pele são menos condutores que os tecidos musculares e nervosos. A pele é o tecido que mais se opõe ao fluxo de cargas, por ter keratina e ter pouca quantidade da água. Feridas abertas e até irritação podem baixar consideravelmente a impedância da pele. Técnicas para baixar a impedância da pele passam por hidratação, provocar pequenas abrasões, aquecimento aplicação de tensões de cerca de 100V e aplicação de electrostimulos com elevadas frequências (alguns kHz).

Impedância típica dos eléctrodos 500ohm
A impedância da pele humana pode variar dos 500k ohm para pele seca até 1000ohm para pele húmida.

Os Standards sugerem para segurança eléctrica que a fuga de corrente entre o chassis e a terra seja inferior a 300uA, entre o paciente e a terra seja menor que50uA.
A resitencia entre o chassis do equipamento ou elemento condutor exposto e o pino de massa do cabo de alimentação seja menor que 0.50ohm.
Para equipamento com inputs isolados da terra, a fuga terá de ser inferior a 10uA ligado à massa e 50uA com a massa aberta.

Para a construção do circuito de electroestimulação foram testados alguns circuitos típicos com utilização de transfromador, cujos circuitos e esquemas são apresentados nas figuras seguintes.

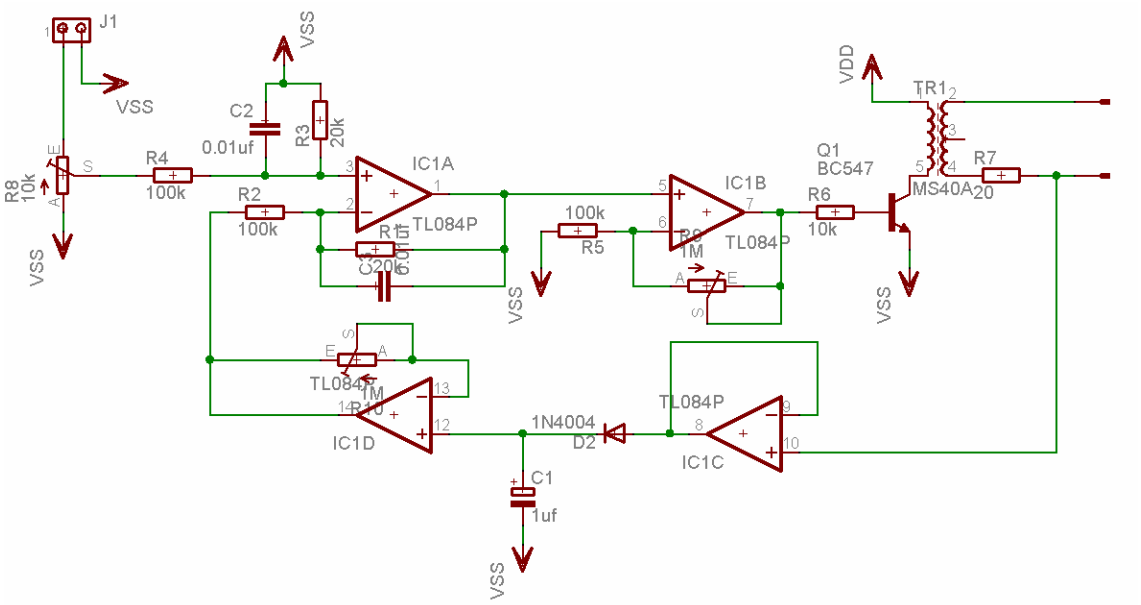

Fig. 14: Esquema de um dos circuito de electroestimulação testados.

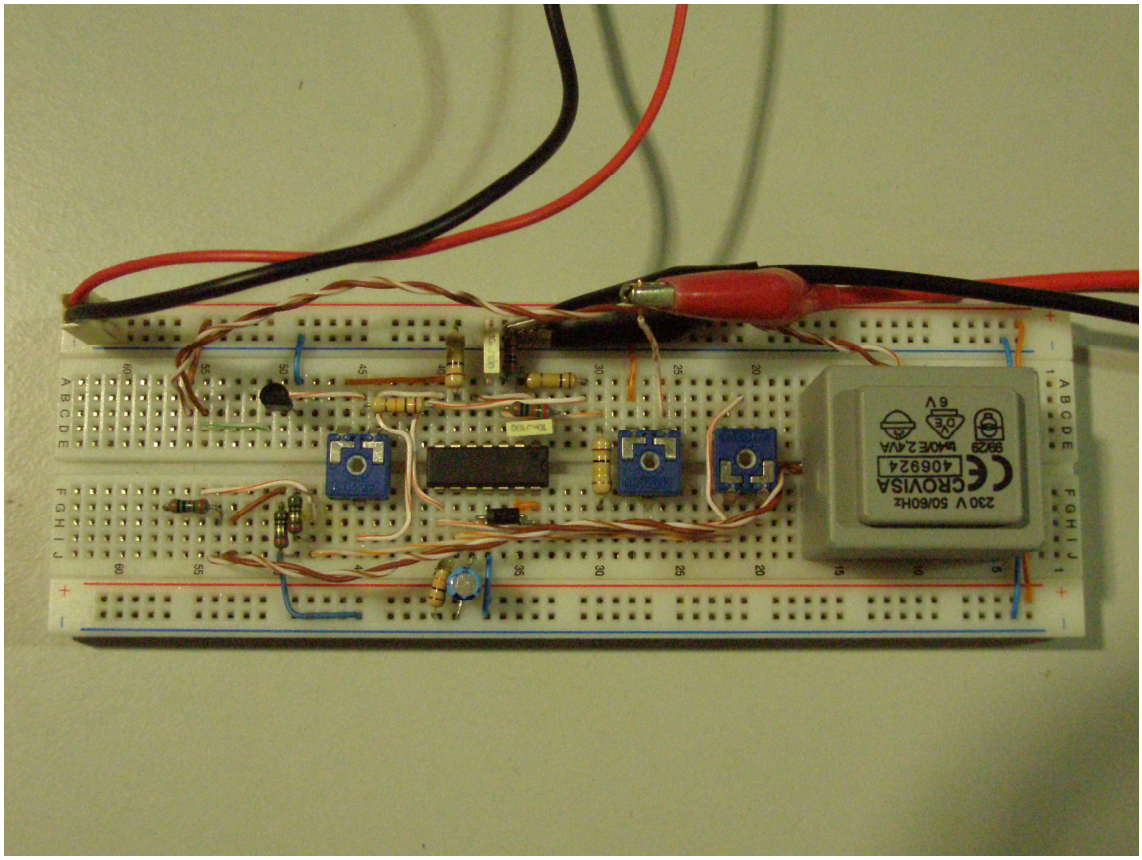

Fig. 15: Imagem da placa respectiva ao esquema do circuito apresentado na figura anterior.

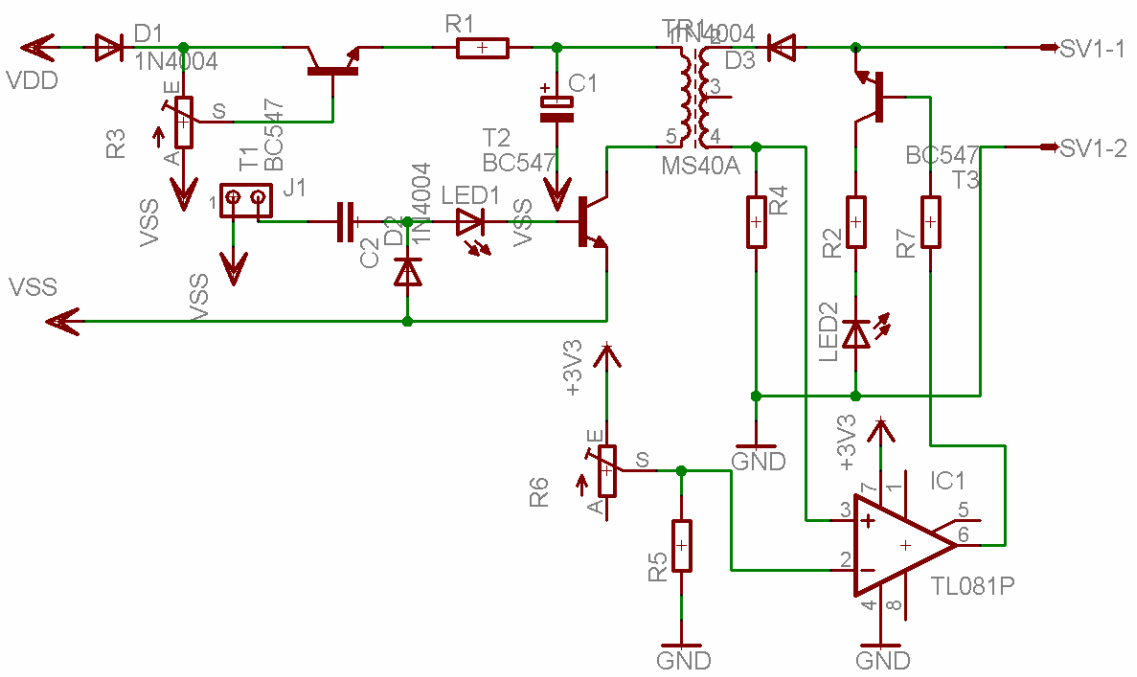

Fig. 16: Esquema do circuito de electroestimulação utilizado.

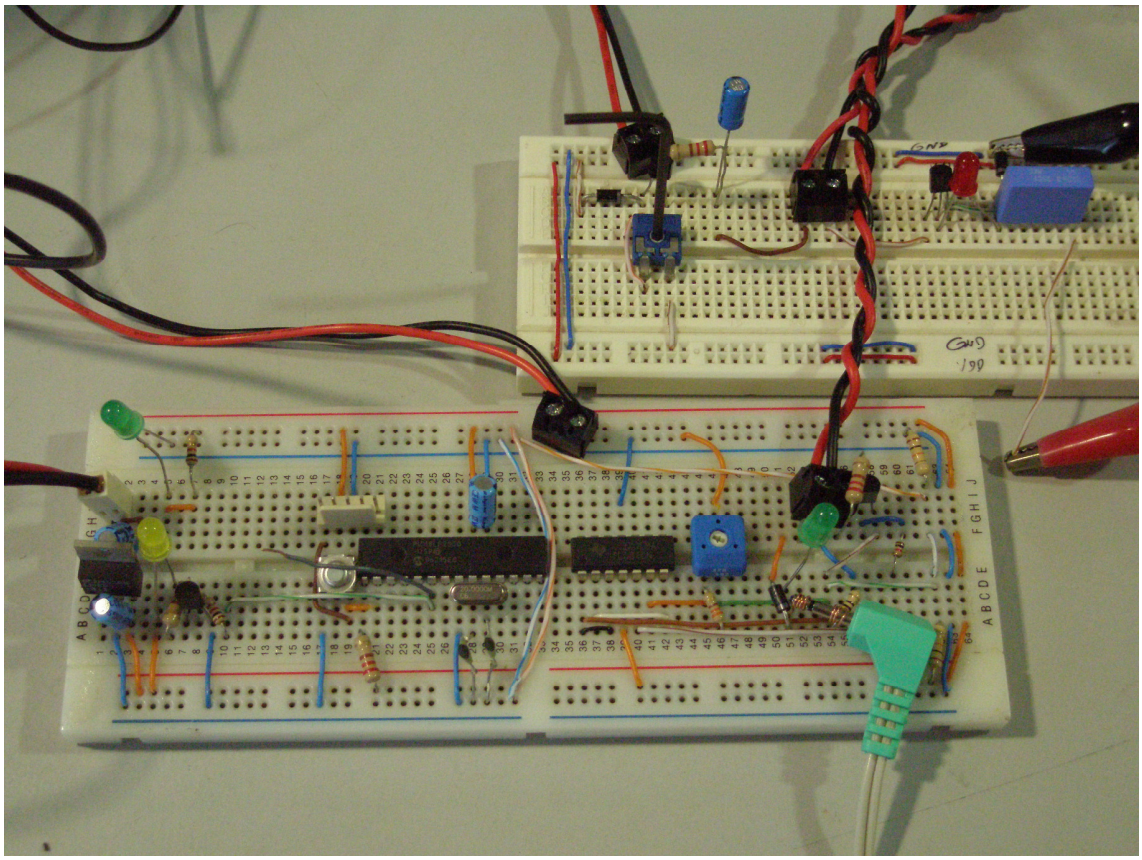

Fig. 17: Imagem da placa utilizada para a electroestimulação.

O segundo circuito mostrado foi escolhido para gerar os pulsos para a electroestimulação por ser o que depois de devidamente dimensionado permitiu obter o sinal com melhores características.

O circuito do electroestimulador é alimentado por uma fonte de 9V que vão ser utilizados para a geração do pulso. A amplitude máxima é controlada por um transístor e por um potenciómetro que futuramente será digital para que o PIC possa programar e assim controlar a amplitude do sinal e as rampas de subida e de descida. O controlo da passagem de corrente pelo enrolamento primário do transformador é feito por um sinal gerado pelo microcontrolador ligado a um segundo transístor. O sinal gerado pelo PIC é criado por meio de temporizadores que permitem controlar a frequência, a largura do pulso e os tempos de trabalho e repouso (períodos de tempo em que é administrado a electroestimuação e onde se deixam as fibras musculares relaxarem, respectivamente).

Com base na informação recolhida no Centro de Medicina de Reabilitação da Região Centro - Rovisco Pais, os parâmetros que consideram mais eficientes são:

Largura de pulso de 300 micro segundos;
Frequência 35Hz
Tempo de trabalho 6 segundos;
Tempo de repouso 10 segundos;
A intensidade de corrente é variável mas 7-8mA já permite obter sensação de contracção forte.

O PIC foi programado com estes valores em termos de tempos e em relação à intensidade houve a preocupação de a controlar no sentido de não exceder o patamar de 10mA para evitar lesões nos tecidos.
Para controlar a intensidade da corrente colocou-se no circuito, fechado pelo enrolamento secundário e pelos eléctrodos, uma resistência de teste de baixo valor para monitorar a corrente. Utilizando um comparador com tensão equivalente à queda de tensão que esta resistência provoca à passagem de uma corrente de 10mA. Este comparador é configurado para quando correntes superiores a 10mA atravessem a resistência de teste fique activo. A saída do comparador está ligada à base de um transístor que serve de interruptor para fechar uma manha com uma resistência baixa e com um LED em paralelo com os eléctrodos. Desta forma quando se geram correntes superiores a 10mA a maioria da corrente vai escapar por esta malha, acendendo o LED para se ter uma informação visual, diminuindo drasticamente a corrente a passar pelos tecidos envolvidos na electroestimulação.

As seguintes figuras ilustram o mecanismo de escapa da corrente.

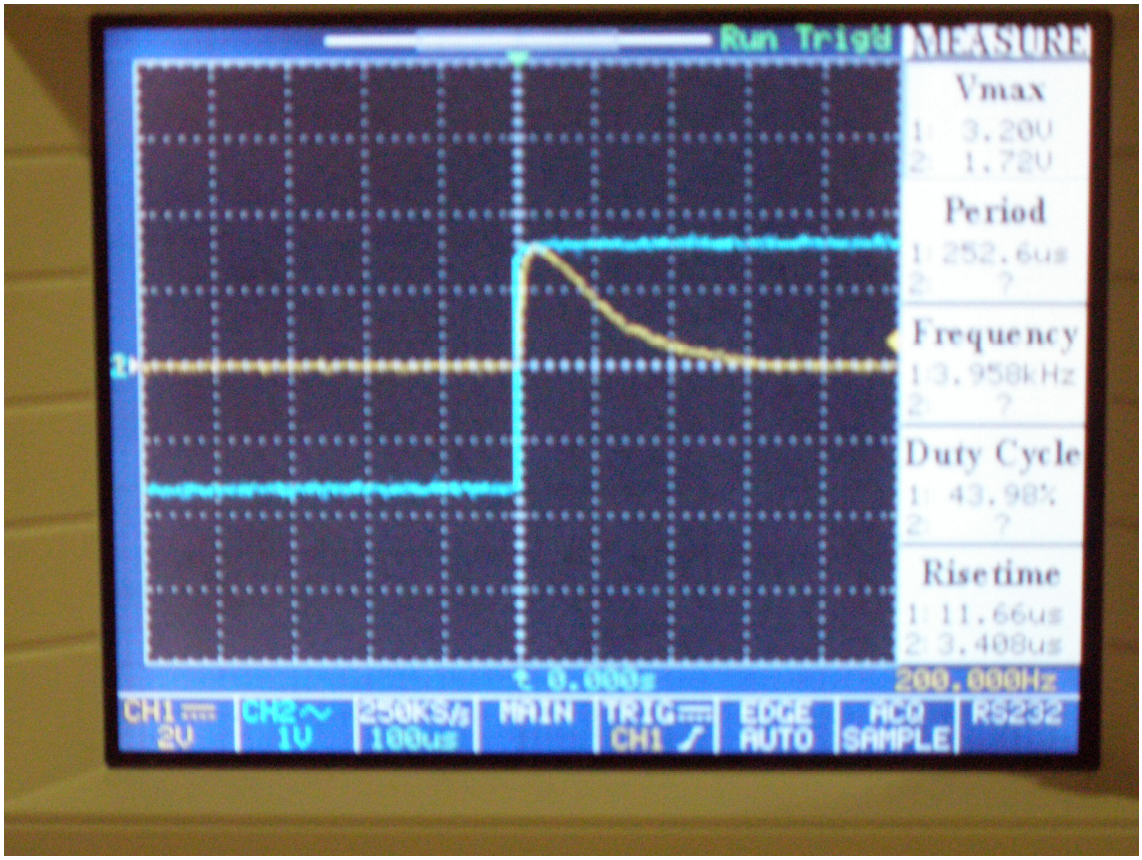

Fig. 18: Pulso medido em tensão nos terminais dos eléctrodos de um pulso onde a corrente é inferior a 10mA.

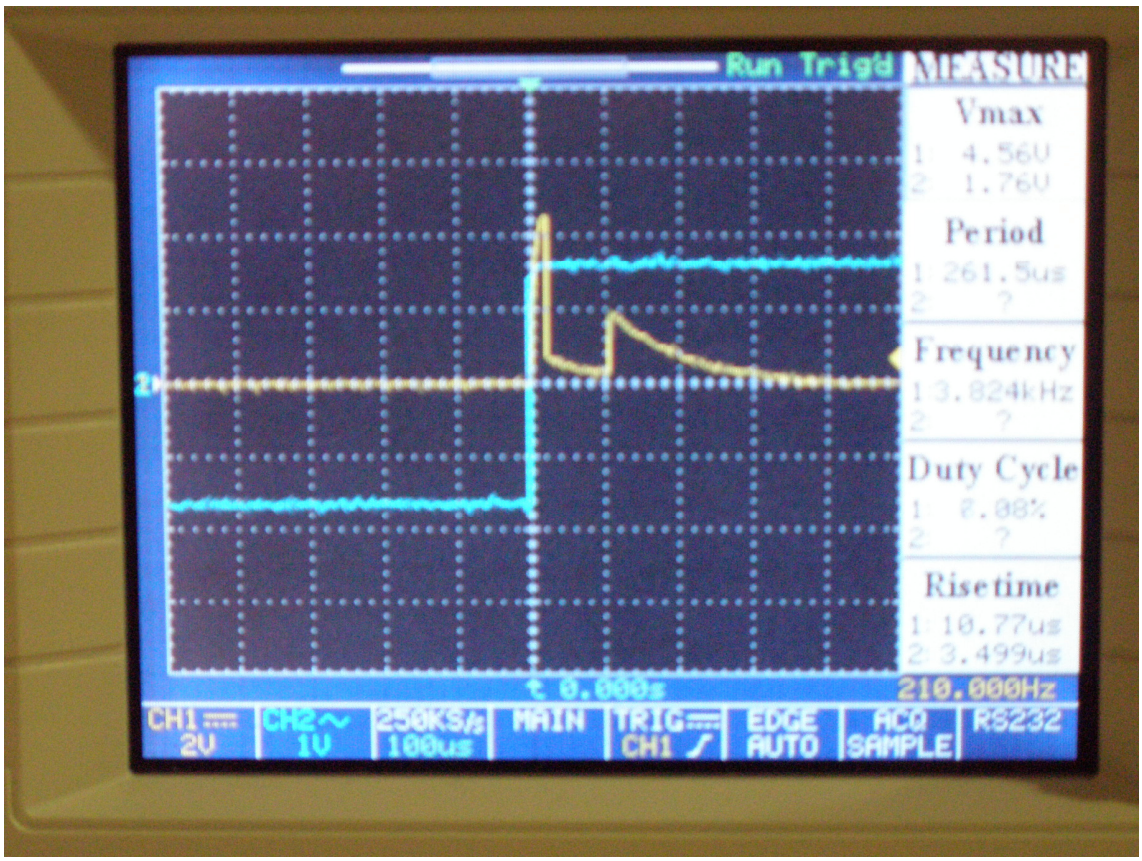

Fig. 19: : Pulso medido em tensão nos terminais dos eléctrodos de um pulso onde a corrente é superior a 10mA.

Está em preparação em sistema de o PIC ler o valor da corrente que passa no sujeito electroestimulado e automaticamente alterar o valor da intensidade do sinal para o valor pretendido.

Foram realizados alguns testes de electroestimuação com o circuito de electroestimuação desenvolvido, onde se obteve sensação de contracção muscular. Os eléctrodos utilizados foram os Cefar Plus electrod para electroestimulação neuromuscular.

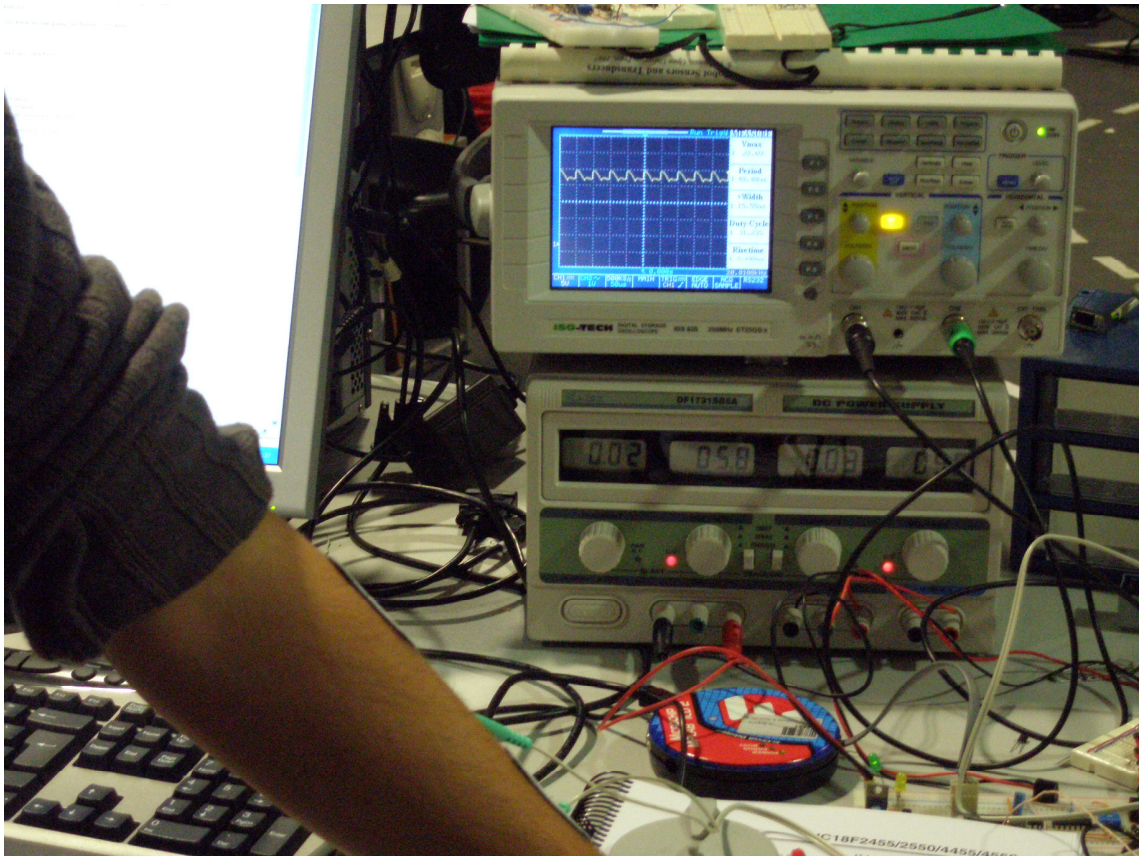

Fig. 20: Imagem da aplicação de electroestimulação.

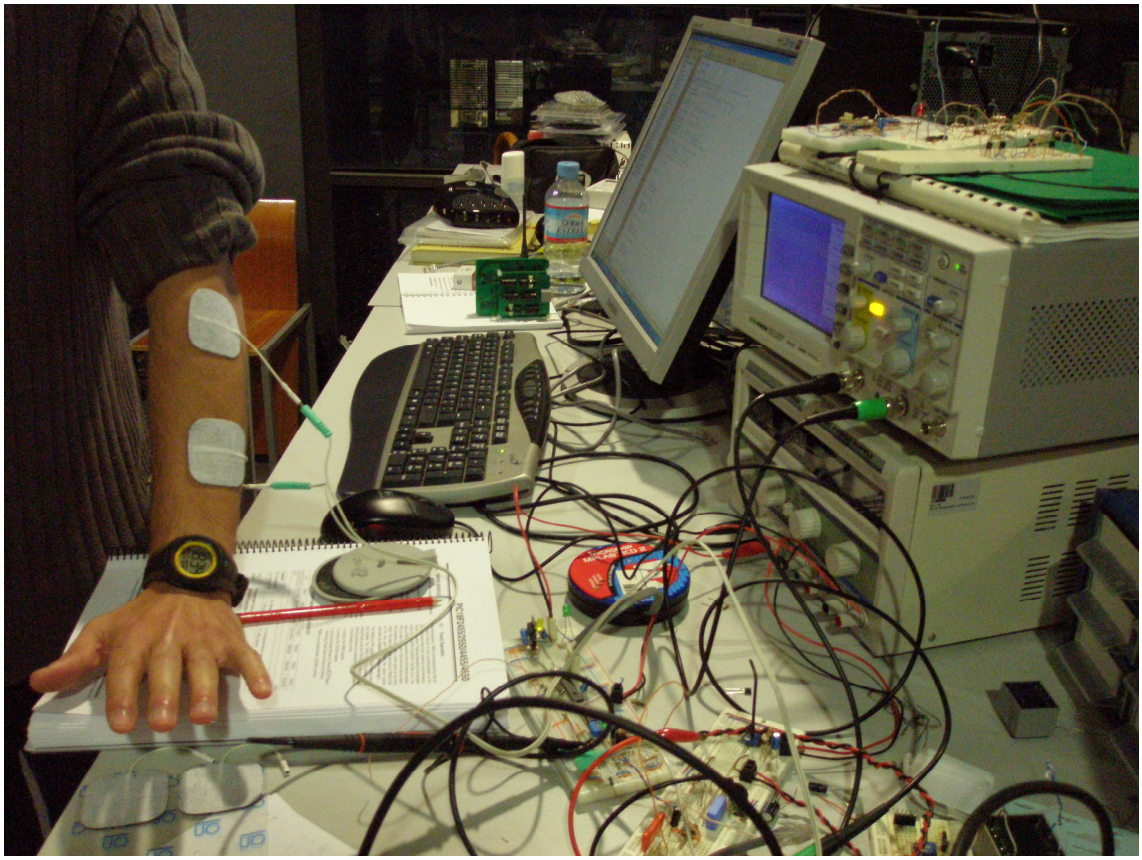

Fig. 21: Imagem de aplicação de electroestimulação.

Trabalhos em preparação relacionados com electroestimulação:
Utilização de um conversor de tensão para corrente para se obter uma fonte de corrente controlável para a alimentar os pulsos de electroestimulação;
A criação de sinal de electroestimuação bifásico.
Garantir o cumprimento das normas de segurança.

Forma da onda com possibilidade em optar entre: monofásica; bifásica simétrica; bifásica assimétrica. A amplitude máxima da onda entre 1mA a 50mA (normalmente ajustável) para a carga de 1k ohm. A duração do pulso (em alguns casos fixo em outros casos ajustável) a variar entre 10us e 1ms. Com frequência de pulso (normalmente ajustável) a variar entre 1pps a 200pps. O padrão de pulsos pode ser continuo ou com séries (‘burst’). Por fim alguns dispositivos têm pulsos com frequência aleatória ou podem modular os pulsos em amplitude, frequência, e ou duração. O número de canais é variável e a alimentação é normalmente feita por baterias de 9V [3].

Aplicação para o PDA

A utilização de comunicação entre o micro controlador e um PDA oferece diversas vantagens. Possibilita controlar e configurar à distância e sem fios o funcionamento do microcontrolador em termos de alteração de parâmetros e accionamento de todas as funcionalidades que os módulos do PIC oferecem. Ecrã táctil e/ou teclas facilitam a interacção com o microcontrolados, evitam a necessidade de hardware e software

adicional e permitem que o sistema seja mais compacto. Permite visualização gráfica qualquer informação recolhida pelos sensores do dispositivo directamente ou de resultado de processamento por parte do PIC, sem que para isso tenha de ter ele próprio qualquer tipo de ecrã. O facto de o dispositivo não ter de possuir ecrã próprio reduz tempo e custos de concepção (hardware e software), permite tamanhos mais reduzidos aumentando a portabilidade do sistema. A capacidade de armazenamento do PDA pode servir para guardar dados. Os dados podem ser transferidos para qualquer parte do mundo através do sistema gsm/gprs (Global System for Mobile Communications/ General Packet Radio Service) ou por Internet. O sistema fica automaticamente com todas as funcionalidades do PDA (comunicação, processador, memória, sistema operativo, localização, aquisição de imagem). Em termos de tamanho/peso o próprio PDA é vantajoso em relação aos restantes tipos de computadores actualmente disponíveis no mercado.

A escolha do PDA a adoptar for determinada com base na potencialidade que este confere ao sistema. Principalmente para desenvolvimento de protótipo foi considerado que era importante ter o máximo de possibilidades em aberto. As características fundamentais que se pretenderam foram:

- GSM/GPRS;
- Bluetoth;
- Porta de série;
- Wifi;
- Usb;
- Expansibilidade de memória;
- Sistema operativo Windows mobile 5.0;
- Maquina virtual Java.

Outras características importantes (em segundo plano) na escolha foram a capacidade de processamento e a autonomia.

Na altura da compra o PDA HP hw6915 era o único disponível no mercado que reunia todas as características. Os únicos pontos negativos a apontar foram:

- A porta de série embora anunciada não estava disponível;
- O usb apenas permite que o dispositivo funcione como periférico;
- A máquina virtual Java verificou ser limitada para os objectivos pretendidos.

Foi desenvolvida uma aplicação em Java na plataforma J2ME (micro edition) com a configuração CLDC (Connected Limited Device Configuration) com o perfil MIDP (Mobile Information Device Profile) através do ambiente de desenvolvimento Netbeans IDE. A aplicação foi criada nesta configuração por ser a compatível com a máquina virtual Java que equipa o PDA de origem. Esta aplicação foi criada como objectivo de comunicar entre o PDA e o microcontrolador por wireless. Onde o PIC é o servidor que espera passivamente que o PDA como cliente solicite o início da comunicação.

A escolha do Java para desenvolvimento da aplicação relacionou-se com o facto de ser software livre e de ser independente do sistema operativo. No entanto ao aplicar verificou-se que a configuração CLDC preparada para funcionar em vulgares telemóveis limita a utilização dos recursos do PDA em termos de capacidade de processamento e memória, verificando ser insuficiente para o armazenamento de dados e gestão da informação às taxas pretendidas.

Para contornar esta dificuldade está previsto, no seguimento dos trabalhos realizados no decurso deste projecto a criação de aplicações em C++ e face ás vantagens inicialmente consideradas, voltar eventualmente a aplicações em Java quando existir oferta de configurações mais potentes.

Um exemplo ilustrativo da aplicação Java para monitorização da pressão de um sensor de força é mostrado nas figuras seguintes. Utilizando uma dedeira com sensor de força o PIC adquire os dados através de conversão analógica digital e transmite por ethernet através do protocolo TCP/IP por intermédio de um 'router wireless' comunicando com o PDA por WIFI.

Fig. 22: Imagem do emulador do ambiente de desenvolvimento Netbeans a mostrar a monitorização de sinal de um sensor de força usando a aplicação em J2ME para o PDA.

Fig. 23: Monitorização da medição do sinal do sensor de Força usando a aplicação no PDA.