Universidade de Aveiro

Departamentos de Física e Mecânica 2006

PROJECTO DE MONITORIZAÇÃO E ELECTRO ESTIMULAÇÃO

Relatório 1: Primeira fase de trabalhos

Eduardo Durana

Aveiro, 20 de Junho de 2006

Índice

MPLAB/COMPILADOR C

O primeiro passo foi instalar o MPLAB IDE que é um ambiente de desenvolvimento integrado que permite desenvolver de aplicações para micro controladores da microchip. Permite escrever código de programas a serem carregados para o micro controlador, compilar, simular, fazer detecção de erros (‘debugging’) e fazer a linkagem para código de máquina apropriado para o controlador a ser usado.

Para que a linkagem seja adequada é importante recorrer à livraria correcta. A este nível o linker cria um ficheiro executável em linguagem de máquina que é transferido para o PIC.

O debugging pode ser realizado através de software ou hardware, tendo o último a vantagem de avaliar exactamente o modo como o hardware está a funcionar, que pode não corresponder ao simulado através de software. Nos trabalhos realizados executou-se os dois tipos de debugging.

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

Antes de utilizar o MPLAB com o desenvolvimento do código em C é necessário ter instalado um compilador C. Neste caso o compilador utilizado foi o MPLAB C18 (o guia aconselha a instalar todos os componentes, para evitar mau funcionamento). Esta necessidade prende-se com o facto de o MPLAB não incluir por defeito um compilador C, apenas tem integrado um compilador Assembler.

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’, no caso dos trabalhos realizados até agora, foi utilizado o PIC18F258.
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 até agora foi utilizado o compilador mcc18, no ‘set language tools’ os executavais 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’).

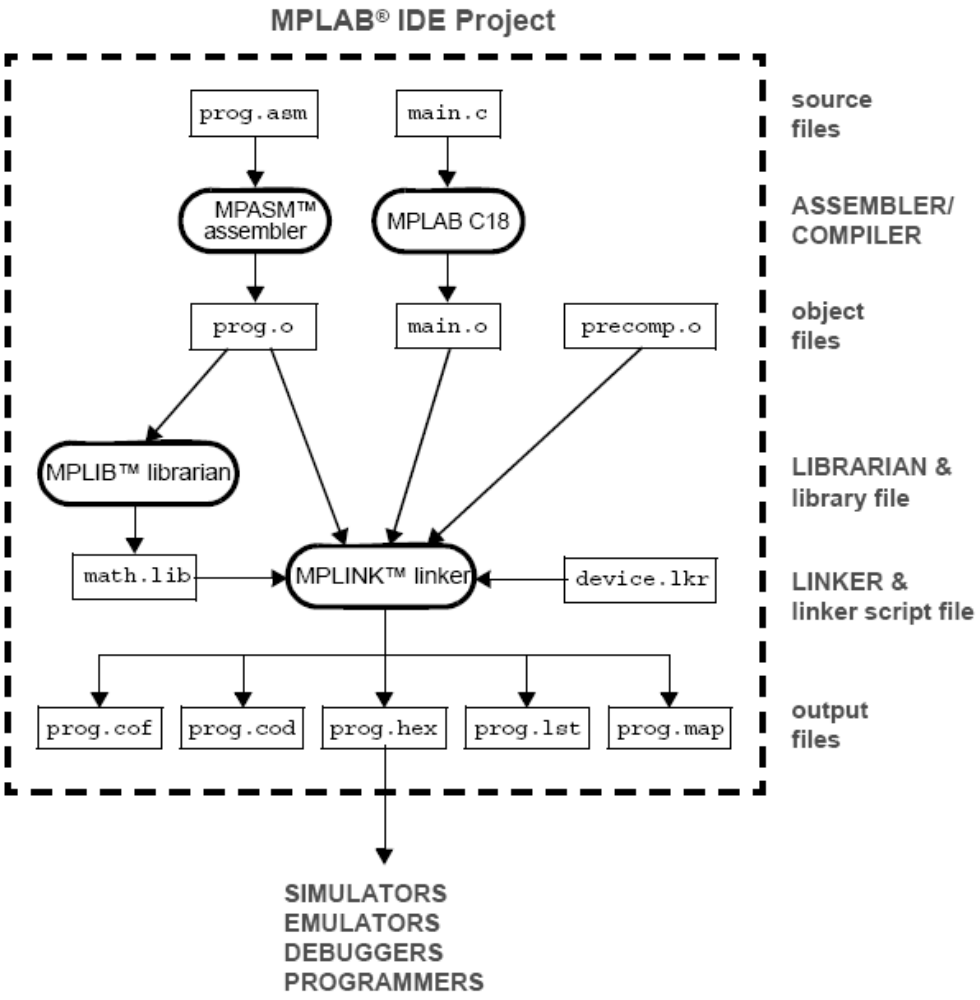

Fig. 1: Esquema retirado do MPASM/MPLINK/MPLIB User's Guide. Este esquema descreve o processo desenvolvido pelo MPLAB IDE projet, mostrando a função do MPASM, MPLAB C18, MPLIB e MPLINK, donde se pode concluir que é fundamental a correcta definição dos executáveis para estas ferramentas para que a programação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

As configurações feitas nos trabalhos realizados foram:

- Oscilador configurado como HS-PLL Enable, isto por estar a ser utilizado um cristal de 10Mhz que portanto é considerado de High Speed (HS – Fosc>4Mhz) e com PLL (Phase Locked Loop) activa (só possível para osciladores até 10MHz). O que a PLL faz é multiplicar a Fosc por 4, o que neste caso equivale a trabalhar com um oscilador de 40MHz.
- O Power Up Timer foi configurado como Enable. Este timer funciona com um temporizador interno RC (resistência/condensador) que está activo o tempo suficiente para que a tensão VDD atinja um valor aceitável, após o microcontrolador ser ligado, apartir do qual sai do modo reset.
- Brown Out Detect foi configurado como enable. Permite fazer Brown-out reset caso VDD cai abaixo do valor de tensão Brown Out Voltage que foi configurada como 2.0V.

- O Watchdog timer (WDT) esta configurado como disable controlado pelo bit SWDTEN com um postscaler de 1:128. o WDT é um temporizador interno do microcontrolador independente de qualquer componente externa e que portanto continua a trabalhar mesmo em ‘modo sleep’ (modo em que o micro controlador se encontra inactivo em que tem um consumo de energia mais baixo), sendo útil para acordar o micro controlador deste modo.
- O Stack Overflow Reset foi configurado como enable. Faz com que quando o número de ‘stack adress’ for superior a 32 (a localização é definida por 5bits) haja reset do micro controlador, evitando que uma nova chamada de programa ou interrupt venha a sobre escrever o que está no espaço de stack da posição 31.
- Os ‘restantes configuration’ bits foram definidos como disable.

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

```
#include <p18f258.h>    //inclui a ‘header file’ p18f128.h que contém
                        protótipos de //funções para o microcontrolador
                        utilizado. Também se //poderia por um ‘header
                        file’ para processador genérico //para todos os
                        PIC18XXXX chamado p18CXXX.

void main(void)   //é a função principal.
{
//inicialização das variáveis.
int i;
const unsigned int T=50000;

//configuração do PORTA em termos de input/output.
TRISA = 0b00000100;          //RA2 - entrada os outros pinos são saída

//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;

while(1)                     //ciclo com o objective de manter o
                             programa a //correr ininterruptamente
{

//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;
      }

}

}
```

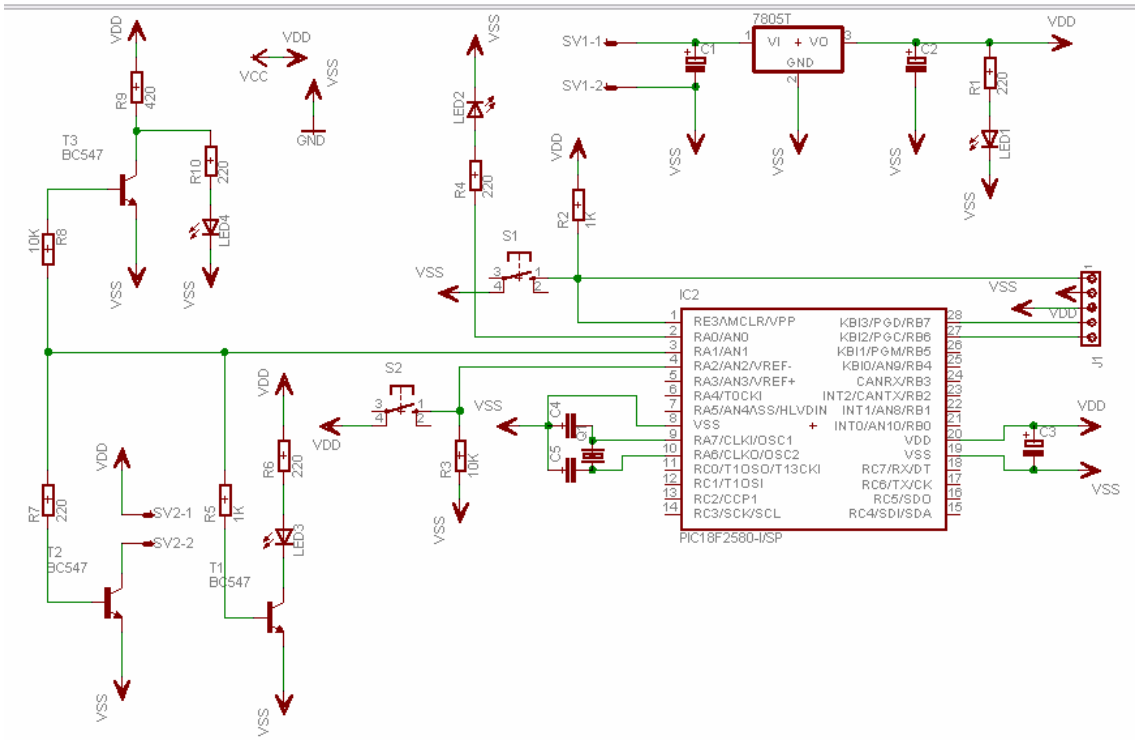

Fig. 2: Esquema de LEDs e motor ligados ao PIC através de transístores por forma a diminuir a corrente solicitada ao microcontrolador. Outra das atenções importantes a ter em conta é a necessidade de condensadores para a alimentação do PIC e para o cristal de modo a estabilizar ao máximo a tensão aplicada,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

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

Outro exemplo consistiu na construção de uma PWM
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 delays através 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.

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'.

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;


            while (!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;

            while (!INTCONbits.TMR0IF);
```

…

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 caracter 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.

```
…
//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 (foi escolhido low speed
                              devido
                              //à elevada frequência do oscilador, para
                              recorrer ao valor tabelado no Datasheet)

SPBRG = 64;                   //valor calculado para Fosc=40MHz e
                              //baud rate=9600. este valor é
                              fundamental para //estipular a que taxas
                              se está a trabalhar.

PIE1bits.RCIE = 1;            //activa o interrup enable do receptor
                        //o interrupt acontece quando RCIF estiver a
'1'
                        //o que acontece quando a recepção está
completa.

RCSTAbits.CREN = 1;           //continuous receive enable bit (activa
                              recepção //continuamente).

RCONbits.IPEN = 1;            //set priority levels on interrupts
IPR1bits.RCIP = 1;      //receive interrupt priority bit as HIGH (a
                        //vantagem de utilizar o interruptor de alta
                        prioridade é que //assim o contexto é salvo
                        automaticamente.)

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

Para que o PIC execute uma acção especifica ao receber informação de fora é necessário criar um interrupt. No inicio do código, fora da função principal é definido o interrupt. Os protótipos das funções a serem utilizadas são definidas da seguinte forma.

```
…
void high_priority_interrupt (void);
void HighVector(void);
…
```

o código do interrupt deve ser atribuido ao espaço da memória destinado a esse fim. Como este espaço é limitado atribui-se o código do interrup 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 á posição correcta. Da forma a seguir descrita.

```
…
#pragma code HighVector=0x8        //define-se a posição onde o código
                                   vai ser //atribuído
void HighVector (void)
{
 _asm goto high_priority_interrupt _endasm     //código em assembler
                                         para ir para //onde o código
                                         está realmente a ser
                                         //escrito (fora do espaço
                                         //originalmente destinado ao
                                         //interrupt).
}
#pragma code // return to default code section
#pragma interrupt high_priority_interrupt
void high_priority_interrupt (void)
{
            PIE1bits.RCIE=0;  //desactiva o interrupt de recepção da
                              USART, para //evitar sobreposição de
                              interrupts

            c = RCREG;        //lê e atribui a uma variável o valor
recebido
            RA0 = val_potenc;       //atribui a variáveis os valores a
transmitir
            RA2 = PORTAbits.RA2;
            if (c == 'T')                 //apenas vais transmitir se
receber ‘T’
            {
            while (!PIR1bits.TXIF); //espera que TXIF fique a ‘1’ que
                                    acontece //assim que o TXREG acaba
                                    de transmitir a //informação
                                    precedente e é esvaziado.
            TXREG = (1<<5)|(RA0*31/255); //só então o TXREG volta a
                                    ser //carregado com nova informação
                                    e o TXIF //volta a ficar a ‘0’.

            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 sempre necessário que TXREG seja lido //para que RCIF seja
                              posto a ‘0’.
```

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

As directivas #pragma 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.

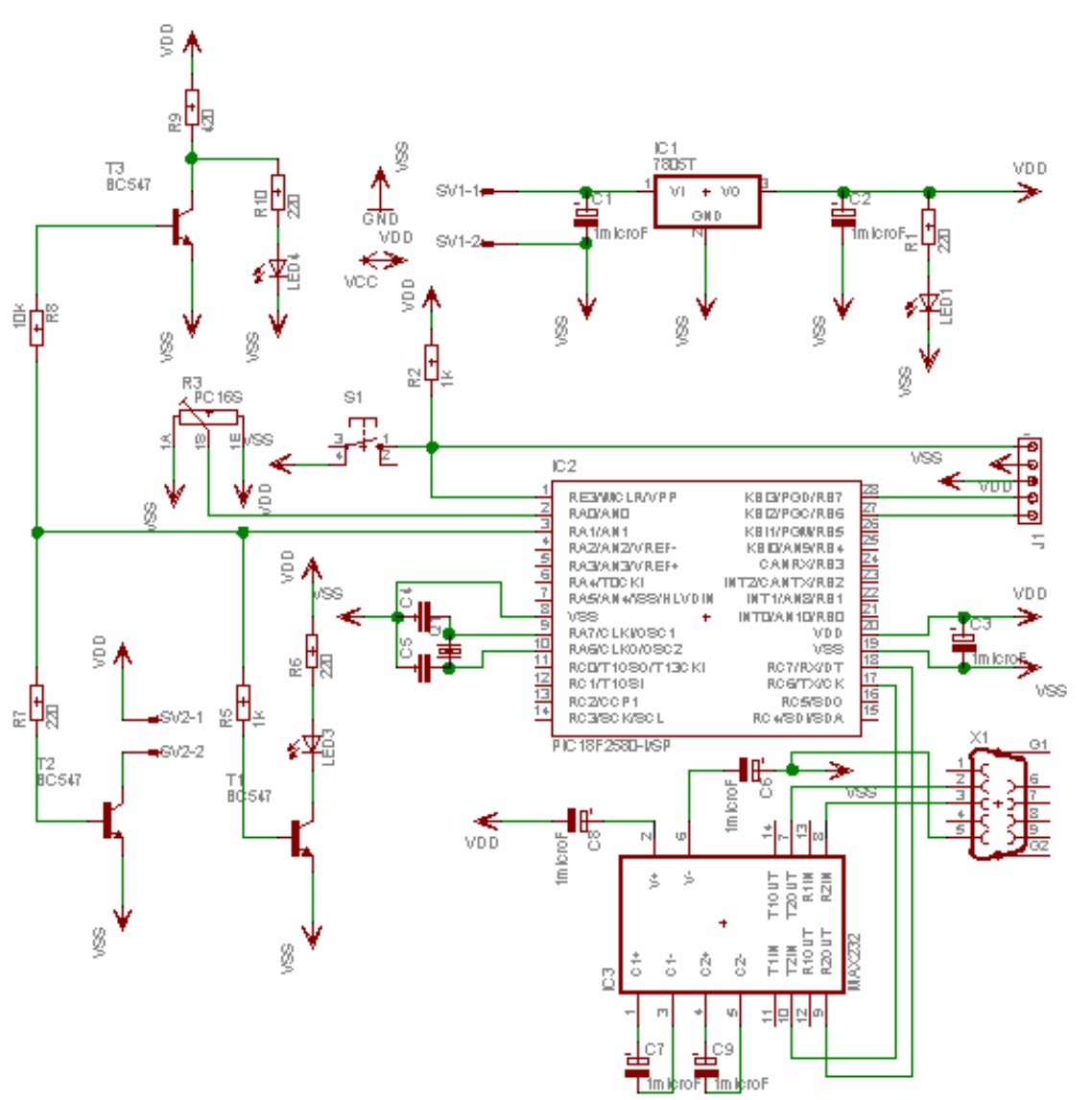

Fig. 3: Esquema de ligação de uma porta RS232 ao PIC18F258.

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;

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

start (t)
```

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

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.

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

      //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 às 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;
      }
}
```