

Relatório de Instrumentação Avançada

Henrique Lopes A85953

Janeiro de 2023

No âmbito da componente prática da UC de Instrumentação Avançada desenvolveu-se um sistema composto por um Microcontrolador PIC18F47Q10 e um acelerómetro capaz de detetar quedas. Com o auxílio de um computador com o programa MPLAB pelo qual é possível programar o Microcontrolador, juntamente, com os códigos em Assembly e C dados previamente pela equipa docente e também pelo Datasheet do próprio Microcontrolador. Deste modo, serão apresentados todos os passos cruciais que passam, essencialmente, por programar o PIC18F47Q10.

CLOCK

O PIC é um circuito digital síncrono o qual necessita de uma fonte de relógio (Clock). Este sinal permite ao PIC sincronizar os diferentes módulos implementados. Para configurar o sinal de relógio do PIC deve ser escrito nos CONFIG registers do mesmo, o seguinte:

```
#pragma config FEXTOSC = 0b100      // No External Oscillator (CONFIG1)
#pragma config CSWEN = ON           // Writing to NOSC and NDIV is allowed (CONFIG1)
#pragma config WDTE = OFF          // WDT operating mode->WDT Disabled
```

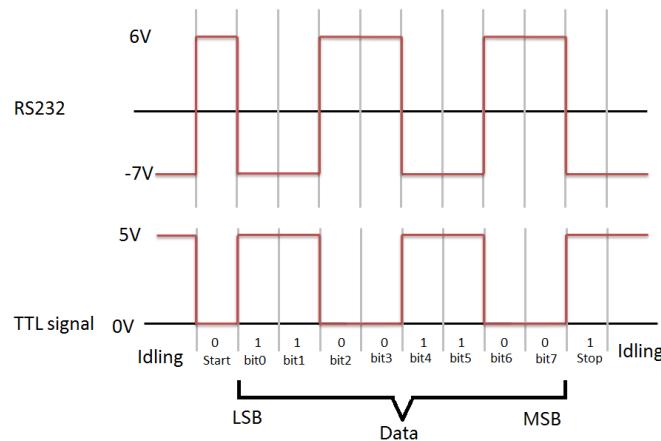
Deste modo, obtemos uma frequência de relógio de 32MHz, o que faz com que a frequência do processador seja 4 vezes inferior à frequência do oscilador por se tratar de uma arquitetura RISC.

$$F_{tcy} = F_{osc}/4$$

Esta fonte de relógio, também, garante sincronismos para os diferentes módulos do PIC.

EUSART - Porta Série

O módulo EUSART permite realizar comunicações com o exterior. Essas comunicações podem realizar-se entre o Microcontrolador e o PC ou entre dois Microcontroladores, respeitando o protocolo RS-232, que padroniza características elétricas como níveis de tensão e características mecânicas para interface (as fichas). No protocolo de comunicação RS-232, os caracteres são enviados um a um como um conjunto de bits, do bit menos significativo para o bit mais significativo. A codificação usada é o "start-stop assíncrono" que usa um bit de início, seguido por oito bits de dados, possivelmente um bit de paridade, e um bit de paragem, sendo então necessários pelo menos 10 bits para enviar um único caracter. Associado ao protocolo, existe a taxa de transferência de bits designada por baud-rate, que permite estabelecer uma velocidade de transferência de dados. É comum estabelecer um baud-rate de 9600 bps.



Exemplo de configuração do módulo EUSART em Assembly para um baud-rate de 9600 bps sem bit de paridade:

```

=====
;CONFIGURE USART
=====
;BR=9600 @ CLK=32 MHz
BANKSEL SPIBRGL
movlw 51
movwf SPIBRGL
movlw 0
movwf SPIBRGH

BANKSEL TX1STA
movlw 0b00100000 ;8 data bits, TX enabled, ASYNC
movwf TX1STA

; Q: HERE YOU MUST ENABLE THE USART AND THE RECEIVER WITH REGISTER RC1STA
movlw 0b10010000 ; USART enabled, 8 data bits / enable receiver 0b10010000
BANKSEL RC1STA
movwf RC1STA ; serial port enabled

```

Exemplo de implementação em C, de funções que permitem enviar um dado pelo módulo e receber um dado:

```
void putch(uint8_t byte) //this function is required for the printf. It te
{
    while(PIR3bits.TX1IF == 0) {};
    TX1REG = byte;           // transmit a character to Serial IO
}

void getch(void) //this function gets the received char from RC1REG
{
    *get_char = RC1REG; //Q: WHERE DOES THE RECEIVED BYTE GO TO?
    flag=1;
    return;
}
```

Exemplo de implementação em C de uma função que permite enviar um vetor de caracteres(uma palavra, uma frase):

```
void putvetor(uint8_t *p){

    while(*p!=0){ //ciclo ate encontrar terminador de string
        putch(*p); //envia o caracter apontado pelo ponteiro
        p++; // incrementa o ponteiro para a proxima posicao
    }
}
```

Exemplo de uma função em C que permite enviar um número de 8 bits (até ao valor 255):

A função implementa um algoritmo que permite determinar os diferentes algarismos que constituem o valor de 8 bits, convertendo para o código correspondente de acordo com a tabela de ASCII.

```
void putadc(uint8_t value){
    unsigned char centena,dezena,unidade;
    centena = value /100 + 0x30; //determina o algarismo das centenas e converte para código ASCII
    value = value %100;
    dezena = value /10 + 0x30; //determina o algarismo das dezenas e converte para código ASCII
    unidade = value % 10 + 0x30;//determina o algarismo das unidades e converte para código ASCII
    putch(centena);
    putch(dezena);
    putch(unidade);
    putch(0x0a); putch(0x0d); // o enter
}
```

ADC - Conversor de sinal analógico para digital

O módulo ADC é um hardware que permite converter sinais analógicos para sinais digitais. O módulo deste PIC tem uma profundidade binária de 10 bits, o que faz com que tenhamos 1024 combinações binárias representativas de valores de tensão. O intervalo de valores analógicos é de 0 a Vcc, o que corresponde ao intervalo de 0 a 5V, respetivamente. O módulo converte 0 volts para o valor digital zero e 5 volts para o valor digital máximo 1023. O valor analógico do intervalo corresponde a 4,8 mV, dado pela seguinte fórmula:

$$\frac{V_{max} - V_{min}}{2^{10}} = \frac{5}{1024} = Tensão/intervalo(resolução) = 4,8mV$$

O valor digital ADC correspondente ao valor analógico é dado pela seguinte fórmula:

$$ADC = \frac{(2^{10} - 1).tensão}{V_{max}} = \frac{1023}{5}.tensão$$

A função getadc foi implementada da seguinte forma:

```
//channel E [0,3]
unsigned char getadc(unsigned char channel){
    ADCLK = 0x3f; //ADC clock Fosc/128 -> conversion at 4us...
    ADCS = 0; //Clock supplied by Fosc, divided according to ADCLK register
    //ADCS=1;//select FRC dedicated oscillator
    ADREF =0; //Vref+ = Vcc and Vref- = gnd
    ADPCH = channel;

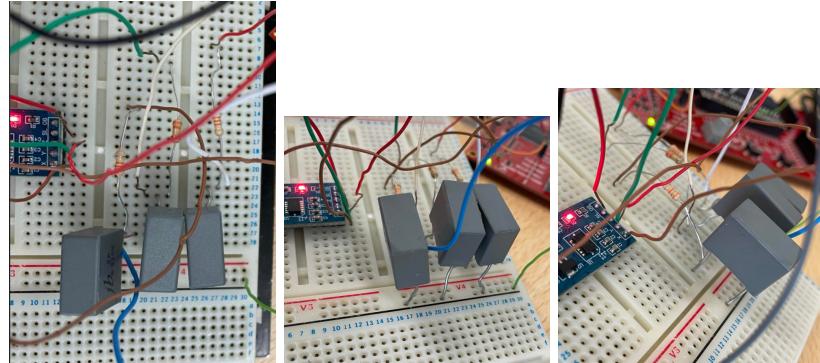
    ADON =1;//adc on
    ADGO=1; // initializes the conversion
    while(ADGO==1){} // tests if the conversion has already ended. approx 4us
    ADON =0;//adc off
    return ADRESH; // returns the 8 most significant bits of the 10-bit result
}
```

O registo ADREF permite selecionar o intervalo de valores analógicos. O bit ADON permite ligar ou desligar o módulo ADC. O bit ADGO permite inicializar a conversão do valor analógico. Uma vez realizada a conversão, o resultado binário fica disponível nos registos ADRESH e ADRESL. Destes dois registos, só é obtido o valor de ADRESH, o que faz com que o resultado final varie no intervalo de 0 a 255.

$$255 = 2^8 - 1$$

Filtro Passa-Baixo

Tal como foi solicitado nas aulas práticas desta UC, foi implementado um filtro passa-baixo em hardware, com o objetivo de controlar/normalizar a frequência de saída dos canais do acelerômetro. Aqui, tomou-se em conta o Teorema de Amostragem, que nos diz que a frequência de corte só pode ter no máximo metade do valor da frequência de amostragem. Como foi visto anteriormente, a frequência de amostragem é de 5Hz (200ms). Então, a frequência de corte não pode ser maior que 2,5Hz. Nas figuras abaixo está ilustrada como seria a implementação do filtro nos 3 canais e a frequência de corte dos filtros, tendo em conta que os condensadores e as resistências têm o mesmo valor: $R = 31,3\text{K}\Omega$ e $C = 2,2\mu\text{F}$.



$$F_c = \frac{1}{2 * \pi * R * C} = \frac{1}{2 * \pi * 31,3 * 10^3 * 2,2 * 10^{-6}} = 2,31\text{Hz}$$

É de salientar que o módulo ADC não suporta frequências de corte acima de 0,25MHz.

$$F_c = \frac{1}{4us} = 0,25\text{MHz}$$

Demonstração gráfica da resposta em frequência do filtro:



Como se pode observar, quando o ganho é de -3db a frequência no eixo do x corresponde à frequência de corte anteriormente calculada, como seria de esperar.

O gráfico abaixo representa a resposta em frequência do filtro que foi obtido experimentalmente. Podem constatar-se as similaridades entre o gráfico anterior e este, o que nos leva a concluir que todos os filtros dos 3 canais estão a funcionar devidamente.



Deteção da queda pormenorizada

O aviso de um registo de queda do acelerómetro é representado pelo piscar do LED RA5. O LED pisca uma vez para cada queda detetada. Para criar um método eficaz sobre a deteção de uma queda, foi necessário tomar em conta para cada eixo do acelerómetro (x,y e z) o valor atual e o último valor, fazendo assim a diferença entre ambos os valores. Considerando que existe uma grande limitação criada pelo hardware (uma grande sensibilidade), levou-se em conta que o eixo mais relevante seria o eixo z. Não descartando a diferença entre o valor atual e o último valor em x e y. Pois, apenas existe movimento nestes eixos se esta diferença for maior que zero. Valorizou-se este movimento para tornar a deteção mais realista. É possível manipular a frequência de amostragem através da função delay (o segundo "delay ms" que aparece na imagem abaixo) que recebe valores em microssegundos. Neste caso a frequência de amostragem é de 5Hz.

```

valorX = getadc(1);
valorY = getadc(2);
valorZ = getadc(3);
if(valorZ - lastZ>3 && valorX-lastX > 0 && valorY-lastY > 0 ){

    led1=1;
    __delay_ms(50);
    led1=0;
}

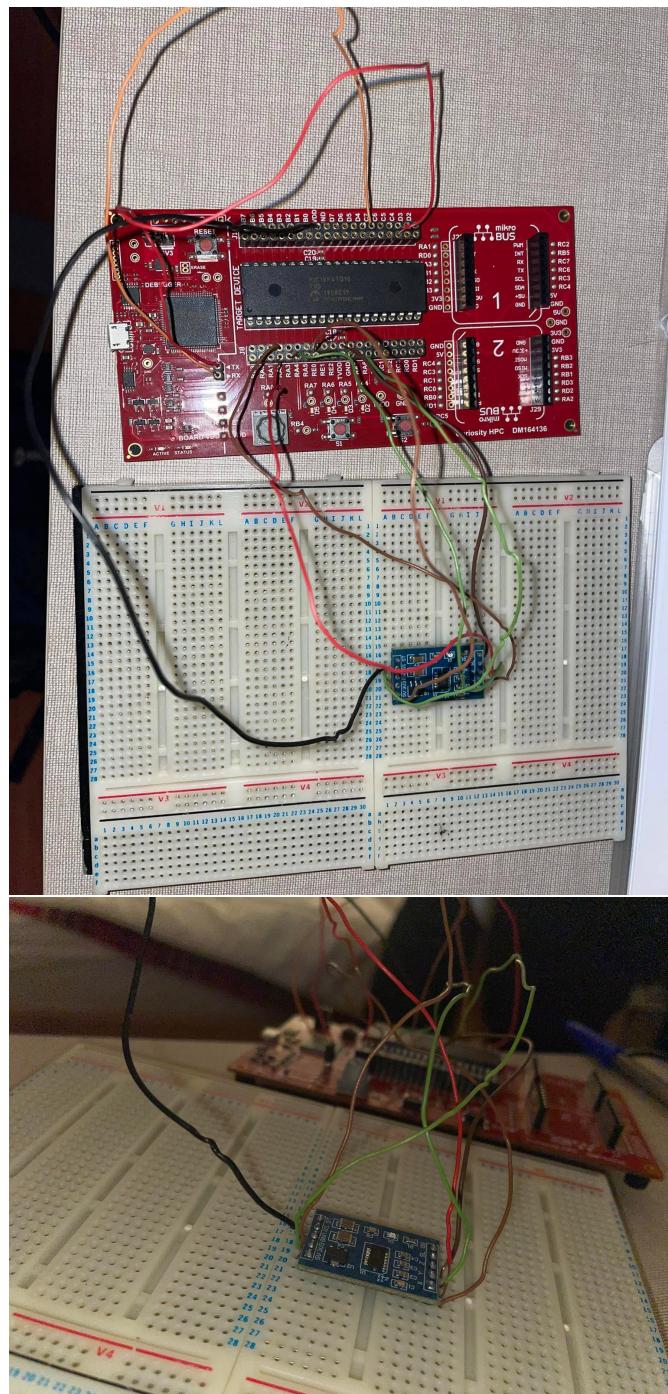
lastZ=valorZ;
lastX = valorX ;
lastY = valorY ;

__delay_ms(200);

```

Todo o Hardware (ligações por fios)

O TX está ligado ao pino RC7 e o RX ligado ao pino RC4. O Ground do acelerómetro está, obviamente, ligado ao Ground do PIC. O G-Select e o Sleep Mode do acelerómetro estão ligados ao TVDD do PIC, o G-Select para fornecer mais sensibilidade para leitura de dados e o Sleep Mode para estar inativo. O pino do acelerómetro de 3,3V também se encontra conectado ao TVDD do PIC. Os três eixos do acelerómetro x, y e z estão, respetivamente, ligados aos pinos RA1, RA2 e RA3. Aqui, não se está a considerar o Harware dos filtros.

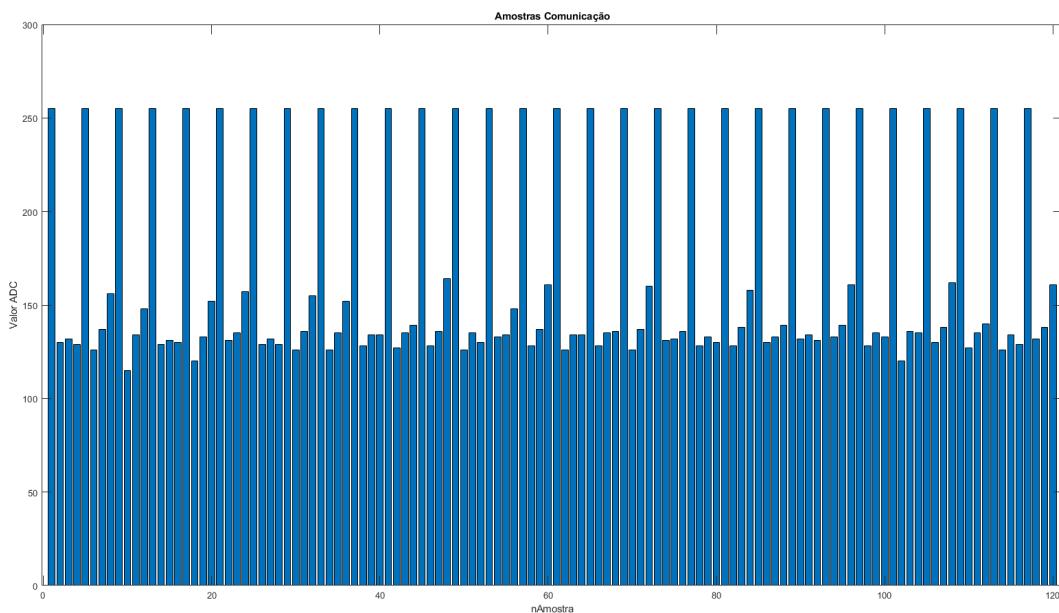


Interface MATLAB-MPLAB

Nesta interface foi possível observar graficamente os valores dos canais x, y e z como também a deteção das quedas num certo intervalo de tempo.

O gráfico abaixo ilustra os dados obtidos durante a comunicação. Neste caso, utilizou-se uma espécie de baliza. Cada amostra retira 4 valores, o primeiro é sempre 255 e os próximos três correspondem aos valores de x, y e z, respectivamente. Assim, cria-se uma sequência de valores fácil de analizar. Exemplo:

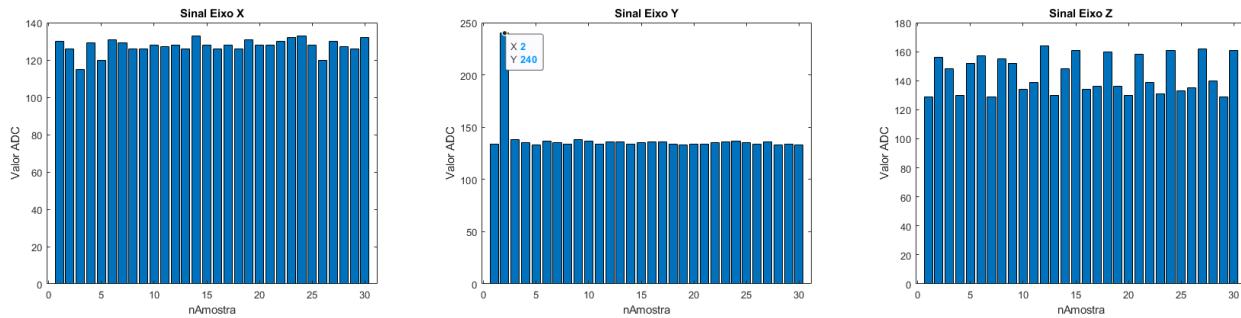
[255, x1, y1, z1, 255, x2, y2, z2, 255, x3, y3, z3, 255, ...]



Deste modo, foi possível isolar os valores de cada canal a partir do vetor índice (posição de cada valor). Abaixo está a demonstração gráfica, não só do código correspondente a este fenómeno, como também, os gráficos dos canais x, y e z.

```
%encontra os indices de amostras com valor 255 (flag de comunicação)
indice = find(samples==255);

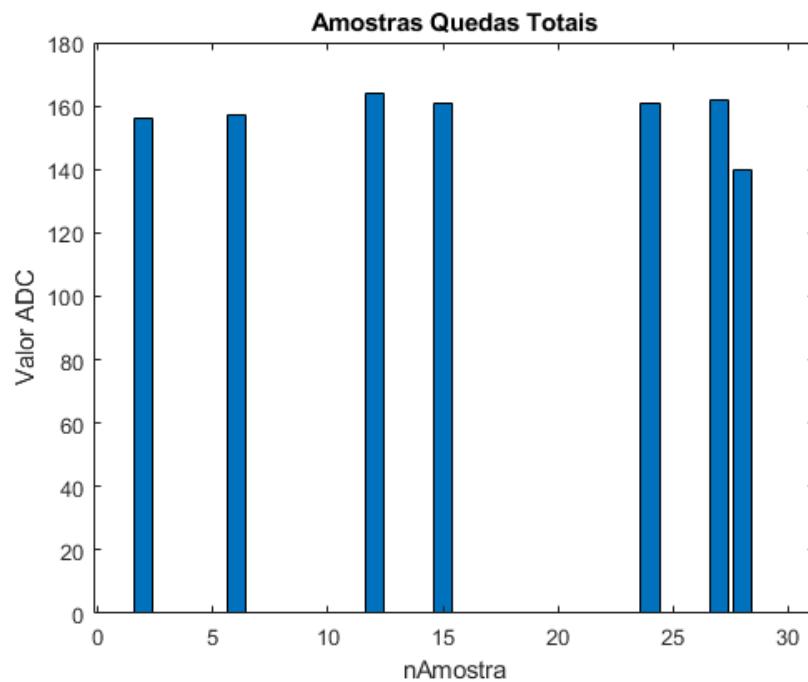
%%%%%
%alocação de memoria para array das amostras individuais
valork = zeros(1,numeroamostras, 'uint8');
valory = zeros(1,numeroamostras, 'uint8');
valorz = zeros(1,numeroamostras, 'uint8');
%obtem as samples individuais de cada canal
%samples por canal corresponde a indice+1 ou indice+2 ou indice+3 (x,y,z)
for i=1:numeroamostras
    k = indice(i)+1;
    valork(i)=samples(k);
    valory(i)=samples(k+1);
    valorz(i)=samples(k+2);
|
end
```



Posteriormente, obteve-se o gráfico correspondente às quedas, criando um código em MATLAB à luz do código (já anteriormente ilustrado) em MPLAB. Neste código, também está presente uma instrução para acender o LED RA6 no final do período de aquisição de valores, caso haja no mínimo uma queda. Se não houver nenhuma queda registrada, então o LED RA6 permanece apagado. Isto tem como objetivo provar que há uma clara comunicação entre o Microcontrolador e o MATLAB. É de esperar que o número de quedas contabilizadas no final do período de aquisição de dados seja igual ao número de vezes que o LED RA5 piscou no mesmo período.

```
%obtenção de amostras para quedas totais
droptotal = zeros(1,numeroamostras, 'uint8');%aloca array
enviarvalordrop=100;
for i=1:numeroamostras-1
    if ( valorz(i)- valorz(i+1))> drop_diff && ( valorx(i)- valorx(i+1) ) >0 && ( valory(i)- valory(i+1) ) > 0
        droptotal(i) = valorz(i);
        enviarvalordrop = 240;%detecta se existe uma queda
    end
end

%se existir alguma queda total, envia valor para o pic (para acender o
%led RA6)
fopen(myserial);
fwrite(myserial,enviarvalordrop,'uint8');
fclose(instrfind);
clear myserial;
```



Para haver uma interface que comunicasse convenientemente foi preciso ajustar código no projeto MPLAB. Este código em C no MPLAB acaba por demonstrar como é feita a comunicação (transmissão de dados) e também a intrução do LED RA6 por completo.

```
putch(255); // (255 é o inicio de trama)
putch(lastX);
putch(lastY);
putch(lastZ);

//detetar se o matlab envia um valor para acender o led RA6
if(flag==1){
    flag=0;
    if(valor == 240)
        led2=1;
    else if (valor==100)
        led2=0;
}
```

Referências Webgráficas

Por fim, é sensato colocar aqui alguns links que foram usados como fontes muito úteis para concluir este trabalho:

<https://www.pantechsolutions.net/matlab-code-to-receive-data-from-a-serial-port>

<https://www.mathworks.com/help/matlab/ref/bar.html>

https://www.mathworks.com/help/matlab/matlab_prog/integers.html