Trabalho prático N.º 6

Objetivos

- Familiarização com o modo de funcionamento de um periférico com capacidade de produzir informação.
- Utilização da técnica de interrupção para detetar a ocorrência de um evento e efetuar o consequente processamento.
- Efetuar a conversão analógica/digital de um sinal de entrada e mostrar o resultado no sistema de visualização implementado anteriormente.

Introdução

Como mencionado no trabalho prático anterior, quando o módulo A/D termina uma sequência de conversão gera um pedido de interrupção (ativa o bit **AD1IF** do registo **IFS1)**. Para que este pedido de interrupção tenha seguimento, o sistema de interrupções do microcontrolador terá que estar devidamente configurado, de modo a que, na ocorrência do evento de fim de conversão, a rotina de serviço à interrupção (*Interrupt Service Routine*, ISR) seja executada.

Para isso, para além das configurações do módulo A/D, já efetuadas anteriormente, é ainda necessário configurar o sistema de interrupções, procedendo do seguinte modo:

1. configurar o nível de prioridade das interrupções geradas pelo módulo A/D – registo **IPC6**[1], nos 3 bits **AD1IP**; terá que ser escrito um valor entre 1 e 6; o valor 7, a que corresponde a prioridade máxima, não deve ser usado; para o valor 0 os pedidos de interrupção nunca são aceites, o que equivale a desativar essa fonte de interrupção:

```
IPC6bits.AD1IP = 2;  // configure priority of A/D interrupts to 2
```

2. autorizar as interrupções geradas pelo módulo A/D – registo **IEC1**, bit **AD1IE**;

```
IEC1bits.AD1IE = 1;  // enable A/D interrupts
```

3. fazer o *reset* de alguma interrupção pendente – registo **IFS1**, bit **AD1IF**;

```
IFS1bits.AD1IF = 0;  // clear A/D interrupt flag
```

4. ativar globalmente o sistema de interrupções.

A rotina de serviço à interrupção, para este exemplo, terá a seguinte organização:

```
// Interrupt Handler
void _int_(VECTOR) isr_adc(void)  // Replace VECTOR by the A/D vector
                                  // number - see "PIC32 family data
                                  // sheet" (pages 74-76)
{
   // ISR actions
   (...)
   IFS1bits.AD1IF = 0;            // Reset AD1IF flag
}
```

O prefixo "**_int_(VECTOR)**" indica ao compilador que se trata de uma função de serviço a uma interrupção. Entre outras coisas, o compilador gera o código necessário para salvaguardar todos os registos que são usados por essa função (*prolog*) e para repor esses valores no final (*epilog*). Consulte anexo sobre como consultar a tabela de vetores do PIC32.

[1] A informação relativa a cada fonte de interrupção, nomeadamente o vetor associado e registos de configuração, está condensada na tabela das páginas 74 a 76 do PIC32MX5XX/6XX/7XX, Family Data Sheet, ou no documento "Vetores de Interrupção do DetPIC32" (disponíveis no *moodle* de AC2).

Trabalho a realizar

1. No trabalho prático anterior fizemos a deteção do evento de fim de conversão do módulo A/D por *polling*, isto é, num ciclo que espera pela passagem a 1 do bit **AD1IF**. O que se pretende agora é que o atendimento ao evento de fim de conversão seja feito por interrupção e não por *polling*.

   O esqueleto de programa que se apresenta de seguida mostra a estrutura-base do programa para interagir com o módulo A/D por interrupção. Neste primeiro exercício pretende-se, tal como já se fez no exercício 1 do trabalho prático anterior, que o módulo A/D gere a interrupção ao fim de 1 conversão (**SMPI=0**). A rotina de serviço à interrupção imprime o valor lido do conversor e dá nova ordem de aquisição ao módulo A/D.

```
void main(void)
{
   // Configure all (digital I/O, analog input, A/D module)
   // Configure interrupt system
   IFS1bits.AD1IF = 0;            // Reset AD1IF flag
   EnableInterrupts();            // Global Interrupt Enable
   // Start A/D conversion
   while(1) { }                   // all activity is done by the ISR
}
```

   A rotina de serviço à interrupção para interação com o módulo A/D e impressão do valor lido poderá ter a seguinte organização:

```
// Interrupt Handler

void _int_(VECTOR) isr_adc(void)  // Replace VECTOR by the A/D vector
                                  // number – see "PIC32 family data
                                  // sheet" (pages 74-76)
{
   // Print ADC1BUF0 value        // Hexadecimal (3 digits format)
   // Start A/D conversion
   IFS1bits.AD1IF = 0;            // Reset AD1IF flag
}
```

2. No trabalho prático anterior mediu-se o tempo de conversão do conversor A/D. Sabendo esse tempo podemos agora estimar a latência no atendimento a uma interrupção no PIC32 (intervalo de tempo que decorre desde o pedido de interrupção até à execução da primeira instrução "útil" da rotina de serviço à interrupção). Para isso, vamos usar novamente um porto digital configurado como saída (por exemplo o **RB6**). Desative o bit **RB6** à entrada da rotina de serviço à interrupção e ative-o à saída.

```
void _int_(VECTOR) isr_adc(void)
{
   // Reset RB6                   // LATB6 = 0
   // Print ADC1BUF0 value        // Hexadecimal (3 digits format)
   // Set RB6                     // LATB6 = 1
   // Start A/D conversion
   IFS1bits.AD1IF = 0;            // Reset AD1IF flag
}
```

   Execute o programa e, com um osciloscópio, meça o tempo durante o qual o bit **RB6** permanece ao nível lógico 1 e tome nota desse valor. Se subtrair a esse tempo o tempo de conversão medido no trabalho prático anterior, obtém a latência do atendimento a uma interrupção no PIC32. Sabendo que a frequência do CPU é 40 MHz, poderá explicitar o resultado em termos do número de ciclos de relógio.

3. Pretende-se agora estimar o *overhead* global do atendimento a uma interrupção no PIC32. Para isso, e para além da latência, temos ainda de considerar o tempo necessário para o regresso ao programa interrompido, essencialmente constituído pelo tempo necessário para repor o contexto salvaguardado no início da rotina de serviço à interrupção.

   Para medir esse tempo podemos ativar o porto de saída no fim da rotina de serviço à interrupção (deve ser a última instrução dessa rotina) e desativar esse mesmo porto no ciclo infinito do programa principal. Meça, com o osciloscópio, o tempo durante o qual o porto **RB6** está ativo e expresse esse tempo em número de ciclos de relógio. Adicionando esse valor ao obtido no ponto anterior, obtém uma boa estimativa para o *overhead* global no atendimento a uma interrupção no PIC32.

4. Integre no programa anterior o sistema de visualização. Faça as alterações que permitam a visualização do valor da amplitude da tensão nos *displays* de 7 segmentos. O programa deverá efetuar 4 sequências de conversão A/D por segundo (cada uma com 8 amostras consecutivas) e o sistema de visualização deverá funcionar com uma frequência de refrescamento de 100 Hz (10 ms). Utilize, na organização do seu código, o programa-esqueleto que se apresenta de seguida:

```
volatile unsigned char voltage = 0;   // Global variable

void main(void)
{
   // Configure all (digital I/O, analog input, A/D module, interrupts)
   ...
   IFS1bits.AD1IF = 0;           // Reset AD1IF flag
   EnableInterrupts();           // Global Interrupt Enable
   i = 0;
   while(1)
   {
      // Wait 10 ms using the core timer
      if(i++ == 25)  // 250 ms (4 samples/second)
      {
         // Start A/D conversion
         // i = 0;
      }
      // Send "voltage" variable to displays
   }
}

void _int_(VECTOR) isr_adc(void)
{
   // Calculate buffer average (8 samples)
   // Calculate voltage amplitude
   // Convert voltage amplitude to decimal. Assign it to "voltage"
   IFS1bits.AD1IF = 0;           // Reset AD1IF flag
}
```

A palavra-chave **volatile** dá a indicação ao compilador que a variável pode ser alterada de forma não explicitada na zona de código onde está a ser usada (i.e., noutra zona de código, como por exemplo numa rotina de serviço à interrupção). Com esta palavra-chave força-se o compilador a, sempre que o valor da variável seja necessário, efetuar o acesso à posição de memória onde essa variável reside, em vez de usar uma eventual cópia, potencialmente com um valor desatualizado, residente num registo interno do CPU.

Elementos de apoio

- Slides das aulas teóricas.
- PIC32 Family Reference Manual, Section 17 – A/D Module.
- PIC32 Family Reference Manual, Section 08 – Interrupts.
- PIC32MX5XX/6XX/7XX, Family Data Sheet, Pág. 74 a 76.

ANEXO:

Exemplo de interpretação da tabela de vetores de interrupção do PIC32MX