

## Aula prática N.º 7

### 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 foi já observado na aula prática 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 (RSI) 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**<sup>1</sup>, 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
```

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

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

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

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

4. ativar globalmente o sistema de interrupções;

```
EnableInterrupts(); // Macro defined in "detpic32.h"
```

A rotina de serviço à interrupção terá a seguinte estrutura:

```
// Interrupt service routine (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\_"** indica ao compilador que se trata de uma rotina de serviço a uma interrupção. Com essa indicação, o compilador gera o código necessário para salvaguardar todos os registos que são usados pela RSI (*prolog*) e para repor o valor desses registos no final (*epilog*). **"VECTOR"** deve ser substituído pelo **vetor associado à fonte de interrupção**. Com essa informação, o compilador preenche a tabela de vetores de interrupção (*interrupt vector table*) com uma instrução de salto incondicional para o endereço da primeira instrução da RSI (na posição da tabela correspondente ao vetor).

<sup>1</sup> 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 (disponível no *moodle* de AC2).

## Trabalho a realizar

No trabalho prático anterior fez-se a deteção do evento de fim de conversão do módulo A/D (ADC) 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*.

1. O programa-esqueleto que se apresenta de seguida mostra a estrutura-base do programa para interagir com a ADC por interrupção, ainda numa versão simplificada. Na função **main()**, após as configurações iniciais, é dada ordem de conversão à ADC e de seguida o programa entra num ciclo infinito. Quando a ADC terminar a conversão gera uma interrupção e o programa salta para a rotina de serviço, e aí é feita a leitura e a impressão do valor convertido. Antes de terminar, a rotina de serviço dá nova ordem de conversão à ADC e regressa ao ciclo infinito do programa principal. Quando a ADC terminar a conversão gera novo pedido de interrupção e o processo repete-se (*free run mode*).

Neste primeiro exercício pretende-se que a ADC gere a interrupção ao fim de 1 conversão (**SMPI=0**).

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

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)
{
    // Read conversion result (ADC1BUF0) and print it
    // Start A/D conversion
    IFS1bits.AD1IF = 0;           // Reset AD1IF flag
}
```

Guarde esta versão do seu programa para que possa ser usada na parte 2 deste guião.

2. 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, em decimal. O programa deverá: i) efetuar 5 sequências de conversão A/D por segundo (frequência de amostragem de 5 Hz), cada uma delas com 8 amostras; ii) enviar informação para o sistema de visualização a cada 10 ms (frequência de refrescamento de 100 Hz). Utilize, na organização do seu código, o programa-esqueleto que se apresenta de seguida:

```

volatile unsigned char voltage = 0;    // Global variable

int main(void)
{
    unsigned int cnt = 0;
    // Configure all (digital I/O, analog input, A/D module, interrupts)
    ...
    EnableInterrupts();    // Global Interrupt Enable
    while(1)
    {
        if(cnt == 0)        // 0, 200 ms, 400 ms, ... (5 samples/second)
        {
            // Start A/D conversion
        }
        // Send "voltage" value to displays
        cnt = (cnt + 1) % ??;
        // Wait ?? ms
    }
    return 0;
}

void _int_(VECTOR) isr_adc(void)
{
    // Read 8 samples (ADC1BUF0, ..., ADC1BUF7) and calculate average
    // Calculate voltage amplitude
    // Convert voltage amplitude to decimal and store the result in the
    // global variable "voltage"
    // Reset AD1IF flag
}

```

**Nota:**

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, fazer 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.

**Parte II**

Na aula prática anterior mediu-se o tempo de conversão do conversor A/D. Sabendo esse tempo podemos agora estimar o tempo que decorre desde o pedido de interrupção (feito pela ADC), até à execução da primeira instrução "útil" da rotina de serviço à interrupção (inclui a latência e o tempo de execução do "prólogo"). Para isso, vamos usar novamente o porto digital **RD11** configurado como saída (disponível no ponto de teste **INT4**).

1. Retome o programa que guardou no final do exercício 1 da parte 1 e faça as seguintes alterações: i) no programa principal configure o porto **RD11** como saída; ii) na rotina de serviço à interrupção desative o porto **RD11** no início e ative-o imediatamente antes de dar ordem de início de conversão ao conversor A/D; iii) retire o *system call* de impressão do valor lido (substitua-o por simples leitura para a variável "**adc\_value**", declarada como **volatile**).

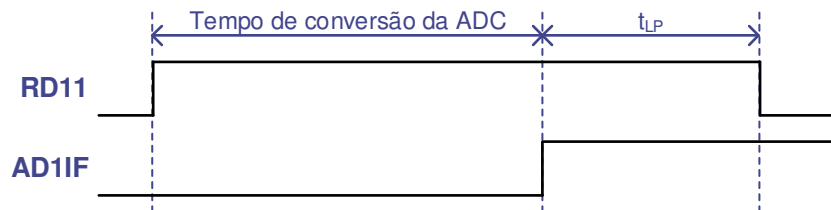
```

void _int_(VECTOR) isr_adc(void)
{
    volatile int adc_value;
    // Reset RD11 (LATD11 = 0)
    // Read ADC1BUF0 value to "adc_value"
    // Start A/D conversion
    // Set RD11 (LATD11 = 1)
    // Reset AD1IF flag
}

```

Execute o programa e, com um osciloscópio, meça o tempo durante o qual o porto **RD11** 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 o tempo que decorre desde o pedido de interrupção até à execução da primeira instrução "útil" da RSI (designado na figura 1 por  $t_{LP}$ ).

Sabendo que a frequência do CPU é 40 MHz, poderá explicitar o resultado em ciclos de relógio ( $\#cycles = t_{LP} * 40E6$ ).



**Figura 1. Medição do tempo que decorre desde o pedido de interrupção até à execução da primeira instrução útil da RSI.**

2. Pretende-se agora estimar o *overhead* global do atendimento a uma interrupção no PIC32. Para isso, 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 ("epílogo").

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 **RD11** 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.

```
void _int_(VECTOR) isr_adc(void)
{
    ...
    // Reset AD1IF flag
    // Set RD11    (LATD11 = 1)
}

int main(void)
{
    ...
    while(1)
    {
        // Reset RD11    (LATD11 = 0)
    }
    return 0;
}
```

### 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.