Trabalho prático N.º 5

Objetivos

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

Introdução

Um conversor analógico-digital (A/D) é um dispositivo electrónico que efetua a conversão de uma quantidade contínua (uma tensão) numa representação digital com **n** bits (uma quantidade numérica com **n** bits). O número de bits que o conversor usa para a representação numérica designa-se por resolução e representa o logaritmo na base 2 do número de níveis em que o conversor divide a gama de tensão de entrada (quanto maior for a resolução, melhor será a exactidão da conversão). Por exemplo, um conversor A/D de 10 bits divide a gama de tensão de entrada em 1024 níveis (2^{10}), fazendo corresponder à tensão mínima o valor 0x000 e à tensão máxima admissível o valor 0x3FF (1023). Se os valores mínimo e máximo dessas tensões forem 0V e 3.3V, respetivamente, as correspondentes representações digitais serão 0x000 e 0x3FF. A Figura 9 mostra um exemplo de codificação de uma gama de tensão Vmax com 3 bits, isto é, com 8 níveis.

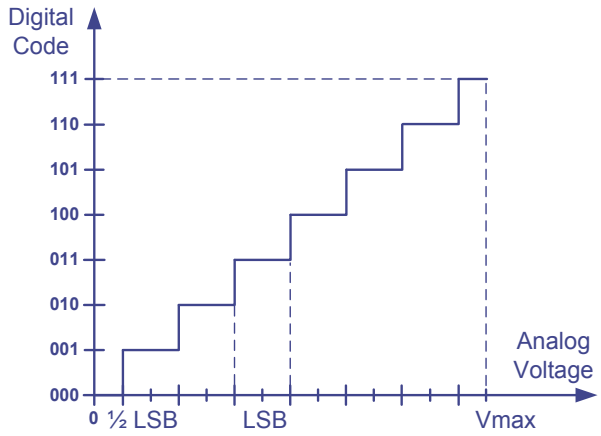

Figura 9. Conversão de uma gama de tensão 0 - Vmax com 3 bits (8 níveis).

O incremento na tensão de entrada a que corresponde uma alteração no bit menos significativo da codificação designa-se por LSB e é dado por LSB = VR / (2^N-1), em que VR é a gama de tensão de entrada e N o número de bits usado para a codificação. No exemplo da Figura 9, se Vmax for igual a 7V então LSB será LSB=7 / (2^3-1) = 1 V. Para um conversor de 10 bits com uma gama de tensão de entrada de 3.3V, o valor do LSB é dado por LSB = 3.3V / (2^{10} -1), aproximadamente 3.2 mV.

O PIC32 disponibiliza um módulo de conversão analógico-digital, com um modelo de programação que permite múltiplas possibilidades de configuração. No essencial, o módulo A/D é constituído por um conversor analógico-digital de 10 bits (um conversor de aproximações sucessivas), um *multiplexer* analógico de 16 entradas e uma zona de memória onde o conversor coloca o(s) resultado(s) da conversão, tal como esquematizado no diagrama de blocos da Figura 10. A zona de memória (designada por *buffer*) é constituída por 16 registos de

32 bits, referenciados pelos nomes **ADC1BUF0**, **ADC1BUF1**, ..., **ADC1BUFF**, que podem ser acedidos nos endereços **0xBF809070**, **0xBF809080**, ..., **0xBF809160**.

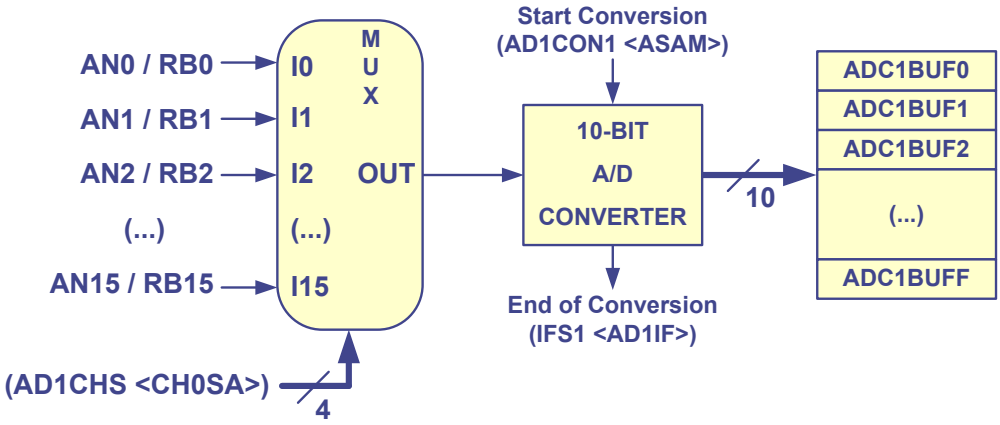

Figura 10. Diagrama de blocos simplificado do módulo A/D do PIC32.

O objetivo deste trabalho prático não é o estudo aprofundado do módulo A/D, pelo que vamos usar apenas um dos vários modos de funcionamento possíveis. Algumas das indicações que serão dadas ao longo deste texto são válidas apenas para o modo de funcionamento escolhido (para informações mais detalhadas deve ser consultado o manual do fabricante disponível no site de AC2). Para esse modo de funcionamento, o processo de conversão envolve:

1) selecionar a entrada analógica a converter, isto é, escolher o canal de entrada (registo **AD1CHS**, bits **CH0SA**);
2) dar ordem de início de conversão ao conversor A/D (registo **AD1CON1**, bit **ASAM**);
3) esperar que o sinal que indica fim de conversão fique ativo (registo **IFS1**, bit **AD1IF**);
4) ler o resultado da conversão em um ou mais registos **ADC1BUFx**.

Configuração das entradas analógicas

As 16 entradas analógicas do PIC 32 são fisicamente coincidentes com os 16 bits do porto B. Qualquer um destes 16 pinos pode ser configurado como entrada analógica ou como porto digital, sendo que na placa DETPIC32 o porto B, por omissão, está configurado como porto digital. A configuração completa de um dado bit **x** do porto B como entrada analógica envolve sempre dois passos: 1) desligar a componente digital de saída do porto, isto é, fazer **TRISBx=1** e 2) configurar o porto como entrada analógica. A configuração como entrada analógica é efetuada através do registo **AD1PCFG**. Por exemplo, para a configuração do bit 4 do porto B como entrada analógica (AN4) pode fazer-se:

```
TRISBbits.TRISB4  = 1;  // RB4 digital output disconnected
AD1PCFGbits.PCFG4 = 0;  // RB4 configured as analog input (AN4)
```

Selecção do canal de entrada

O *multiplexer* analógico permite selecionar, em cada instante, qual a entrada analógica que é encaminhada para o conversor A/D. A selecção do canal de entrada é efetuada através dos 4 bits do campo **CH0SA** do registo **AD1CHS**. Por exemplo, para a seleção da entrada AN4 como entrada para o conversor A/D pode fazer-se:

```
AD1CHSbits.CH0SA = 4;   // Selects AN4 as input for the A/D converter
```

Configuração do número de conversões consecutivas do mesmo canal

O módulo A/D permite configurar o número de conversões consecutivas do mesmo canal antes de o conversor gerar o evento de fim de conversão. Essa configuração é efetuada no registo **AD1CON2** nos 4 bits do campo **SMPI**. Os valores possíveis de configuração, em binário, vão, assim, desde **0000** a **1111**, a que correspondem 1 e 16 conversões consecutivas, respectivamente. Por exemplo, para efetuar 4 conversões consecutivas no canal 7 pode fazer-se:

```
AD1CHSbits.CH0SA = 7;    // Selects AN7 as input for the A/D converter
AD1CON2bits.SMPI = 3;    // 4 samples will be converted and stored
                         // in buffer locations ADC1BUF0 to ADC1BUF3
```

O *buffer* de armazenamento referido anteriormente, é preenchido em função do número de conversões que tiver sido previamente configurado, começando sempre em **ADC1BUF0**. No exemplo de cima, o sinal de fim de conversão só é gerado quando o conversor A/D tiver efetuado as 4 conversões.

Início de conversão e deteção de fim de conversão

Na configuração adoptada para estas aulas práticas, o módulo A/D funciona em modo manual. Significa isto que um processo de conversão só começa quando há uma ordem específica de início. Para essa configuração, a ordem de conversão é dada através da instrução:

```
AD1CON1bits.ASAM = 1;     // Start conversion
```

Quando o módulo A/D termina uma sequência de conversão gera um pedido de interrupção (activa o bit **AD1IF** do registo **IFS1)**. Este pedido de interrupção pode ou não ter seguimento, dependendo da configuração do sistema de interrupções. Se não estiverem a ser usadas interrupções, a deteção do evento de fim de conversão terá quer ser feita por *polling*, esperando, em ciclo, que o bit **AD1IF** transite do nível lógico 0 para o nível lógico 1. O ciclo de *polling* pode ser efetuado do seguinte modo:

```
while( IFS1bits.AD1IF == 0 );   // Wait while conversion not done
```

O bit **IFS1** é automaticamente ativado pelo módulo A/D, mas a sua desativação é manual. Assim, terminada a operação de leitura dos valores da sequência de conversão, é sempre necessário fazer o *reset* desse bit (**IFS1bits.AD1IF = 0**).

Configuração completa do módulo A/D

Para além das apresentadas anteriormente, há ainda um conjunto de configurações adicionais que têm que ser efetuadas para que o módulo A/D funcione de acordo com o pretendido. A lista completa de configurações do módulo A/D fica então:

```
AD1CON1bits.SSRC = 7;     // Conversion trigger selection bits: in this
                          //  mode an internal counter ends sampling and
                          //  starts conversion
AD1CON1bits.CLRASAM = 1;  // Stop conversions when the 1st A/D converter
                          //  interrupt is generated. At the same time,
                          //  hardware clears the ASAM bit
AD1CON3bits.SAMC = 16;    // Sample time is 16 TAD (TAD = 100 ns)
AD1CON2bits.SMPI = XX-1;  // Interrupt is generated after XX samples
                          //  replace XX by the desired number of
                          //  consecutive samples
AD1CHSbits.CH0SA = YY;    // replace YY by the desired input
                          //  analog channel (0 to 15)
AD1CON1bits.ON = 1;       // Enable A/D converter
                          //  This must the last command of the A/D
                          //  configuration sequence
```

Trabalho a realizar

1. No circuito da figura seguinte uma resistência variável (designada por potenciómetro) está ligada ao bit AN14 do PIC32 (RB14). Na configuração em que está montada, a resistência variável permite variar, de forma linear, a tensão no seu ponto intermédio entre 0 V e 3.3 V. Monte esse circuito e ligue o ponto intermédio do potenciómetro à entrada RB14 da placa DETPIC32.

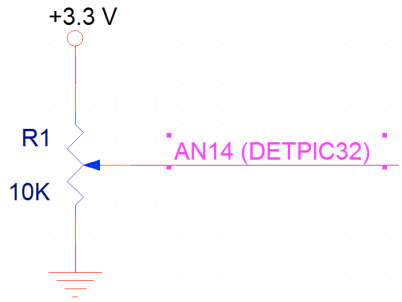

Figura 11. Ligação da resistência variável à placa DETPIC32.

2. Escreva um programa que:
   a) configure o porto RB14 como entrada analógica; configure o módulo A/D de acordo com as indicações dadas anteriormente e de modo a que o número de conversões consecutivas seja 1;
   b) em ciclo infinito: 1) dê ordem de conversão; 2) espere pelo fim de conversão; 3) utilizando o *system call* **`printInt()`**, imprima o resultado da conversão (disponível em **`ADC1BUF0`**) em hexadecimal, formatado com 3 dígitos.

```
int main(void)
{
   // Configure A/D module and RB14 as analog input
   while(1)
   {
      // Start conversion
      // Wait while conversion not done (AD1IF == 0)
      // Read conversion result (ADC1BUF0 value) and print it
      // Reset AD1IF
   }
}
```

   Rode gradualmente o potenciómetro e observe os valores produzidos pelo programa.

3. Meça o tempo de conversão do conversor A/D. Para isso, configure um bit como saída digital (por exemplo RE0) e faça a ativação desse porto durante o tempo em que a conversão está a ser realizada (comente a linha de código correspondente à impressão do valor). Com o osciloscópio meça o tempo de ativação de RE0, a que corresponde, com boa aproximação, o tempo total de conversão, e tome nota desse valor.

```
int main(void)
{
   volatile int aux;
   // Configure A/D module; configure RE0 as digital output
   while(1)
   {
      // Set RE0
      // Start conversion
      // Wait while conversion not done (AD1IF == 0)
      // Reset RE0
      // Read conversion result (ADC1BUF0) to "aux" variable
      // Reset AD1IF
   }
}
```

4. Altere o programa anterior de modo a imprimir as 16 posições do *buffer* do módulo A/D (valores impressos em decimal, formatados com 4 dígitos, separados por 1 espaço). Relembre que os endereços dos 16 registos do *buffer* de armazenamento dos resultados são múltiplos de 16 (**0xBF809070**, **0xBF809080**, ..., **0xBF809160)**. Pode fazer do seguinte modo:

```
int *p = (int *)(&ADC1BUF0);
for( i = 0; i < 16; i++ )
{
   printInt( p[i*4], ... )
   ...
}
```

5. Altere a configuração do módulo A/D de modo a que o conversor efetue 4 conversões consecutivas. Observe o resultado.

6. O valor fornecido pelo conversor é, como já foi referido anteriormente, a representação digital com 10 bits da amplitude da tensão na sua entrada. Sabendo que a tensão máxima à entrada é 3.3V, pode facilmente determinar-se a amplitude da tensão que deu origem a um valor produzido pelo conversor A/D: **V=(VAL_AD*3.3)/1023**. O cálculo de V utilizando apenas inteiros obriga a usar uma representação em vírgula fixa com, por exemplo, uma casa decimal. Para isso, a expressão anterior pode ser re-escrita do seguinte modo: **V=(VAL_AD*33)/1023** ou, com arredondamento, **V=(VAL_AD*33+511)/1023**.

   Retome o programa que escreveu no exercício 5 e acrescente código para calcular a média das 4 amostras retiradas e para determinar a amplitude da tensão e imprimir o seu valor usando *system calls*.

7. Integre no programa anterior o sistema de visualização que desenvolveu no trabalho prático n.º 4. 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 delas com 4 amostras) e o sistema de visualização deverá funcionar com uma frequência de refrescamento de 100 Hz (10 ms).

```
void main(void)
{
   // Configure all (digital I/O, analog input, A/D module)
   i = 0;
   while(1)
   {
      // Wait 10 ms using the core timer
      if(i++ == 25)  // 250 ms
      {
         // Convert analog input (4 samples)
         // Calculate buffer average
         // Calculate voltage amplitude
         // Convert voltage amplitude to decimal
         // i = 0;
      }
      // Send voltage value to displays
   }
}
```

Note: Antes de se iniciar uma nova sequência de conversão, o ou os valores resultantes da sequência de conversão anterior têm que ser obrigatoriamente lidos, na sua totalidade, do *buffer* de resultados. Enquanto essa operação de leitura não for feita, o módulo A/D ignora qualquer comando posterior de início de conversão.