Aula prática N.º 8

Objetivos

- Programação e utilização de *timers*.
- Utilização das técnicas de *polling* e de interrupção para detetar a ocorrência de um evento e efetuar o consequente processamento.

Introdução

Timers são dispositivos periféricos de grande utilidade em aplicações baseadas em microcontroladores permitindo, por exemplo, a geração de eventos de interrupção periódicos ou a geração de sinais PWM (*Pulse Width Modulation*) com *duty-cycle* variável. O seu funcionamento baseia-se na contagem de ciclos de relógio de um sinal com frequência conhecida. O PIC32 disponibiliza 5 *timers*, T1 a T5, que podem ser usados para a geração periódica de eventos de interrupção ou como base de tempo para a geração de sinais PWM. Esta última funcionalidade está reservada aos timers T2 e T3 e é implementada recorrendo ainda a um módulo designado pelo fabricante por *Output Compare Module*.

No PIC32MX795F512H (versão usada na placa DETPIC32), os *timers* T2 a T5 são do tipo B e o T1 é do tipo A. A principal diferença entre o *timer* de tipo A e os de tipo B reside no módulo *prescaler* (pré-divisor) que apenas permite, no de tipo A, a divisão por 1, 8, 64 ou 256. Nos de tipo B a constante de divisão pode ser 1, 2, 4, 8, 16, 32, 64 ou 256. Os *timers* do tipo B podem ser agrupados dois a dois implementando, desse modo, um *timer* de 32 bits. A Figura 1 apresenta o diagrama de blocos simplificado de um *timer* tipo B do PIC32.

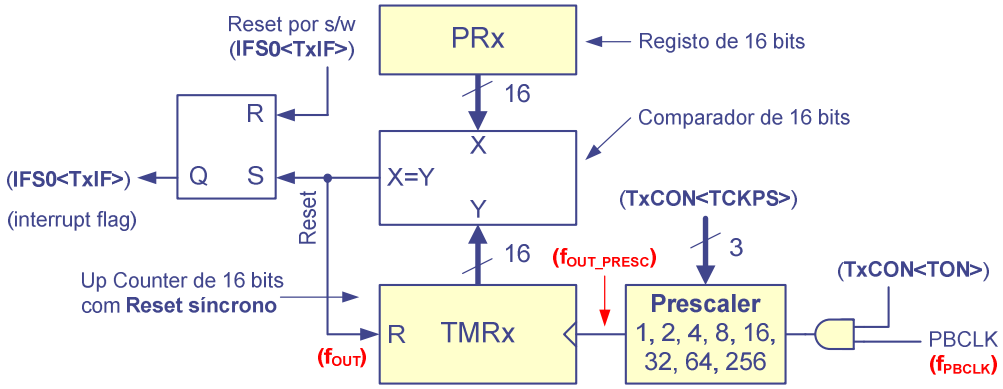

Figura 1. Diagrama de blocos simplificado de um timer tipo B.

Nesta visão simplificada, a fonte de relógio para os *timers* é apenas o *Peripheral Bus Clock* (**`PBCLK`**) que, na placa DETPIC32, está configurado para ter uma frequência igual a metade da frequência do sistema, isto é, f_{PBCLK} = 20 MHz (**`FREQ/2`**, ou **`PBCLK`** em C).

Cálculo das constantes para geração de um evento periódico

O módulo de pré-divisão (*prescaler*) faz uma divisão da frequência $\mathbf{f_{PBCLK}}$ por uma constante configurável nos 3 bits **`<TCKPS>`** do registo **`TxCON`**[1] (designada mais à frente por $\mathbf{K_{PRESCALER}}$), para os *timers* T2 a T5, ou nos 2 bits **`TCKPS`** do registo **`T1CON`**, para o *timer* T1. Por exemplo, no *timer* tipo A, se **`<TCKPS>`** for configurado com o valor 3, a que corresponde uma constante de divisão de 256, o valor de f_{OUT_PRESC} obtido é:

$$f_{OUT\_PRESC} = \frac{f_{PBCLK}}{256}$$

[1] A letra "x" deve ser substituída pelo número do *timer* (2 a 5). Para informação completa sobre o modelo de programação, deve ser consultado o manual do fabricante "PIC32 Family Reference Manual, Section 14 – Timers".

Conhecida a frequência do sinal à saída do *prescaler*, pode determinar-se a frequência do sinal gerado pelo *timer*, do seguinte modo:

$$f_{OUT} = \frac{f_{OUT\_PRESC}}{PRx + 1}$$

em que **PRx** é o valor da constante de 16 bits armazenada num dos registos **PR1** a **PR5** (*timers* T1 a T5).

Exemplo: determinar o valor de **PR2** e da constante de divisão do *prescaler* de modo a que o *timer* T2 gere eventos de "fim de contagem" a uma frequência de 10 Hz (i.e. a cada 100 ms).

Se o *prescaler* for configurado com o valor 1, então $f_{OUT\_PRESC} = f_{PBCLK}$ = 20 MHz e **PR2** fica:

$$PR2 = \left(\frac{20 \times 10^6}{10}\right) - 1 \qquad \texttt{(em C, PR2=PBCLK/10 - 1;)}$$

Ora, uma vez que o registo **PR2** é de 16 bits, o valor máximo da constante de divisão é **65535** (2^{16}-1), pelo que a solução anterior é impossível. Será então necessário configurar o módulo *prescaler* para baixar a frequência do sinal à entrada do contador do *timer*, de modo a tornar possível a divisão usando uma constante de 16 bits.

$$f_{OUT} = \frac{\left(f_{PBCLK} / K_{PRESCALER}\right)}{(PR2 + 1)}$$

Usando para **PR2** o valor máximo possível (**65535**), podemos determinar o valor mínimo para a constante de divisão do *prescaler* como:

$$K_{PRESCALER} = \left\lceil \frac{f_{PBCLK}}{((65535 + 1) \times f_{OUT})} \right\rceil = \lceil 30.51 \rceil = 31$$

Se, por exemplo, se usar uma constante de divisão de 32 (os valores possíveis seriam 32, 64 ou 256), f_{OUT_PRESC} = 20 MHz / 32 = 625 KHz. Refazendo o cálculo para o valor de **PR2** obtém-se:

$$PR2 = \left(\frac{625 \times 10^3}{10}\right) - 1 = 62499$$

valor que já é possível armazenar num registo de 16 bits.

A obtenção de um evento com a mesma frequência no *timer* T1 obrigaria à utilização de uma constante de divisão de 64, uma vez que o valor 32 não está disponível nesse *timer* (tipo A).

Configuração do *timer*

A programação dos *timers* envolve: **i**) configuração da constante de divisão do *prescaler* (registo **TxCON**, bits **<TCKPS>**), **ii**) configuração da constante de divisão **PRx**., **iii**) ativação do timer (registo **TxCON**, bit **<TON>**). A sequência para a configuração do *timer* T2 com os parâmetros do exemplo anterior é:

```
T2CONbits.TCKPS = 5; // 1:32 prescaler (i.e. fout_presc = 625 KHz)
PR2 = 62499;         // Fout = 20MHz / (32 * (62499 + 1)) = 10 Hz
TMR2 = 0;            // Clear timer T2 count register
T2CONbits.TON = 1;   // Enable timer T2 (must be the last command of the
                     //  timer configuration sequence)
```

Configuração do *timer* para gerar interrupções

Se se pretender que o *timer* gere interrupções é necessário, para além da configuração-base apresentada no ponto anterior, configurar o sistema de interrupções na parte respeitante ao *timer* ou *timers* que estão a ser usados, nomeadamente, prioridade (registo **`IPCn`**, bits **`<TxIP>`**), *enable* das interrupções geradas pelo *timer* pretendido (registo **`IEC0`**, bits **`<TxIE>`**) e *reset* inicial do bit **`<TxIF>`** (registo **`IFS0`**)[2]. Para o *timer* T2, a sequência de comandos que configura o sistema de interrupções fica então:

```
IPC2bits.T2IP = 2;   // Interrupt priority (must be in range [1..6])
IEC0bits.T2IE = 1;   // Enable timer T2 interrupts
IFS0bits.T2IF = 0;   // Reset timer T2 interrupt flag
```

[2] Para saber quais os registos que deve configurar para um *timer* em particular deve consultar o manual do fabricante "PIC32, Family Reference Manual Section 14-Timers", e o "PIC32MX5XX/6XX/7XX, Family Data Sheet", Pág. 74 a 76 (ambos disponíveis no site da UC).

Trabalho a realizar

Parte I

1. Calcule as constantes relevantes e configure o *timer* T3, de modo a gerar eventos com uma frequência de 2 Hz. Em ciclo infinito, faça *polling* do bit de fim de contagem **`T3IF`** (**`IFS0<T3IF>`**) e envie para o ecrã o caracter **`'.'`** sempre que esse bit fique ativo:

```
int main(void)
{
   // Configure Timer T3 (2 Hz with interrupts disabled)
   while(1)
   {
      // Wait while T3IF = 0
      // Reset T3IF
      putChar('.');
   }
   return 0;
}
```

2. Substitua o atendimento por *polling* por atendimento por interrupção, configurando o *timer* T3 para gerar interrupções à frequência de 2 Hz.

```
int main(void)
{
   // Configure Timer T3 with interrupts enabled
   EnableInterrupts();
   while(1)
   {
      IdleMode()[3]; // CPU enters Idle mode (CPU is halted,
                     // but peripherals continue to operate)
   }
   return 0;
}

void _int_(VECTOR) isr_T3(void)   // Replace VECTOR by the timer T3
                                  //  vector number
{
   putChar('.');
   // Reset T3 interrupt flag
}
```

3. Altere o programa anterior de modo a que o *system call* **`putChar()`** seja evocado com uma frequência de 1 Hz (como poderá facilmente verificar não é possível obter diretamente, através do timer, a frequência de 1 Hz; uma solução será chamar o *system call* a cada 2 interrupções).
4. O objetivo deste exercício é fazer a configuração do sistema de interrupções e dos *timers* T1 e T3: o *timer* T1 a gerar interrupções à frequência de 5 Hz e o *timer* T3 a gerar interrupções à frequência de 25 Hz.
   a) Determine as constantes relevantes para que o *timer* T1 (tipo A) gere eventos de interrupção a cada 200 ms (5 Hz) e o *timer* T3 (tipo B) gere eventos de interrupção a cada 40 ms (25 Hz).
   b) Escreva o programa principal com todas as configurações necessárias e as Rotinas de Serviço à Interrupção dos *timers* T1 e T3. Nas rotinas de serviço à interrupção deve apenas imprimir um caracter: '1' na RSI do *timer* T1 e '3' na RSI do *timer* T3.

[3] Macro **`IdleMode()`** definida no ficheiro **`detpic32.h`**. Consulte o anexo no final do guião.

```
int main(void)
{
   // Configure Timers T1 and T3 with interrupts enabled)
   // Reset T1IF and T3IF flags
   EnableInterrupts();            // Global Interrupt Enable
   while(1)
   {
      IdleMode();
   }
   return 0;
}

void _int_(VECTOR_TIMER1) isr_T1(void)
{
   // print character '1'
   // Reset T1IF flag
}

void _int_(VECTOR_TIMER3) isr_T3(void)
{
   // print character '3'
   // Reset T3IF flag
}
```

Verifique o correto funcionamento do sistema, observando no ecrã do PC a sequência de números impressa. Interprete essa sequência.

Altere a frequência das interrupções do *timer* 3 para 50 Hz e verifique novamente o funcionamento do sistema.

5. Reponha a frequência das interrupções do *timer* 3 em 25 Hz, configure os portos **RD0** e **RD2** como saídas e inicialize-os com o valor lógico 0. Na RSI do timer T1 faça o *toggle* (mudar o estado lógico) do porto **RD0** e na RSI do timer T3 faça o *toggle* do porto **RD2**. Com o osciloscópio observe os sinais nestes dois portos e meça os tempos a 1 e a 0 dos dois sinais (**RD2** e **RD0** estão disponíveis nos pontos de teste **OC3** e **OC1**, respetivamente).

6. Configure os portos **RE1** e **RE3** como saídas e na RSI do timer 3 faça o *toggle* do porto **RE3** e na RSI do timer 1 faça o *toggle* do porto **RE1**. Observe nos LEDs 3 e 1 o comportamento do sistema.

Parte II

O PIC32 tem 5 entradas externas de interrupção (**INT0** a **INT4**) mapeadas em 5 pinos do porto **RD**. A placa DETPIC32 tem disponível um pulsador ligado à entrada **RD8**, a que corresponde a entrada de interrupção externa **INT1**.

1. Faça um programa que, usando *polling* e a função **delay()**, ligue o **LED0** durante 3 segundos, quando se prime o pulsador **INT1** (o **LED0** está ligado ao porto **RE0**).

2. Pretende-se agora fazer todo o processamento por interrupção (deteção e temporização). Para isso, deve usar o *timer* 2 para gerar a temporização (3 segundos) e configurar o sistema de interrupções para dar seguimento a interrupções do *timer* T2 e da entrada externa **INT1**. A entrada **INT1** deve ser configurada para gerar a interrupção na transição descendente (registo **INTCON<INT1EP>**, ver página 89 do PIC32MX7XX Family Data Sheet).

```
int main(void)
{
   // Configure ports, Timer T2, interrupts and external interrupt INT1
   EnableInterrupts();
   while(1)
   {
      IdleMode();
   }
   return 0;
}

void _int_(??) isr_T2(void)
   ...
void _int_(??) isr_INT1(void)
   ...
```

Elementos de apoio

- Slides das aulas teóricas (aulas 10 e 11).
- PIC32 Family Reference Manual, Section 08 – Interrupts.
- PIC32 Family Reference Manual, Section 14 – Timers.
- PIC32 Family Reference Manual, Section 17 – A/D Module.
- PIC32MX5XX/6XX/7XX, Family Datasheet, Pág. 74 a 76, Pág. 89.

Anexo – Modo de poupança de energia no PIC32

No microcontrolador PIC32, o modo "**`Idle`**" é uma funcionalidade que permite reduzir o consumo de energia, mantendo apenas os periféricos essenciais em funcionamento enquanto o CPU suspende temporariamente as suas operações ativas. Esta transição ocorre quando a instrução *assembly* "**`wait`**" é executada e o registo "**`OSCCON`**" tem o bit "**`SLPEN`**" configurado como 0 (configuração por defeito após um Power-On Reset ou POR).

Quando o modo "**`Idle`**" é ativado:

- O CPU é colocado num estado de inatividade, reduzindo o consumo de energia.
- Os periféricos ativos continuam a funcionar normalmente (por exemplo, portos de I/O, *timers*, ADC, UART, …).

A saída do modo "**`Idle`**" ocorre através de uma interrupção ou *reset*, permitindo ao sistema retomar rapidamente o funcionamento normal.

Em sistemas alimentados por bateria, como dispositivos IoT, sensores remotos ou dispositivos eletrónicos portáteis, a eficiência energética é crucial. As razões principais para utilizar o modo "**`Idle`**" são:

- **Redução do consumo energético**: o CPU suspende as operações ativas, diminuindo significativamente o consumo de energia, enquanto os periféricos continuam operacionais. Isto é útil quando o microcontrolador precisa apenas de monitorizar eventos ou esperar por dados, sem processamento ativo.
- **Prolongamento da vida útil da bateria**: a redução do consumo energético traduz-se numa maior autonomia para dispositivos alimentados por bateria, especialmente em aplicações que passam longos períodos em estado inativo.

Ao utilizar o modo "**`Idle`**", é possível alcançar um equilíbrio entre desempenho e eficiência energética, permitindo que o sistema opere de forma sustentável e económica durante mais tempo evitando, simultaneamente, o desperdício de energia durante períodos de inatividade

A macro **`IdleMode()`**, declarada no ficheiro **`detpic32.h`**:

```
#define IdleMode()   asm volatile("wait")
```

é uma forma simples de encapsular a instrução *assembly* "**`wait`**" na linguagem C, facilitando o seu uso no código da aplicação.

PDF criado em 24/02/2025