# Aula prática N.º 9

## *Objetivos*

- Programação e utilização de *timers* com interrupções.
- Geração de sinais PWM.

## *Introdução*

### Geração de um sinal PWM

PWM (*Pulse Width Modulation*, ou modulação por largura de pulso) é uma técnica usada em múltiplas aplicações, desde o controlo de potência a fornecer a uma carga à geração de efeitos de áudio ou à modulação digital em sistemas de telecomunicações. Esta técnica utiliza sinais retangulares, como o apresentado na Figura 1, em que, mantendo o período T, se pode alterar dinamicamente a duração a 1, $t_{ON}$, do sinal.

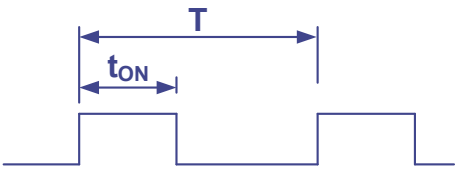


**Figura 1. Exemplo de sinal retangular com um período T e um tempo a 1 $t_{ON}$.**

O *duty-cycle* de um sinal PWM é definido pela relação entre o tempo durante o qual o sinal está no nível lógico 1 (num período) e o período desse sinal, e expressa-se em percentagem:

$$\boldsymbol{Duty\text{-}cycle} = \frac{\boldsymbol{t_{ON}}}{\boldsymbol{T}} \times \mathbf{100}[\%]$$

No PIC32 a geração de sinais PWM é feita usando os *timers* T2 ou T3 e o *Output Compare Module* (OC). A Figura 2 apresenta o diagrama de blocos desse sistema, onde se evidencia a interligação entre o módulo correspondente aos *timers* T2 e T3 e o módulo OC.

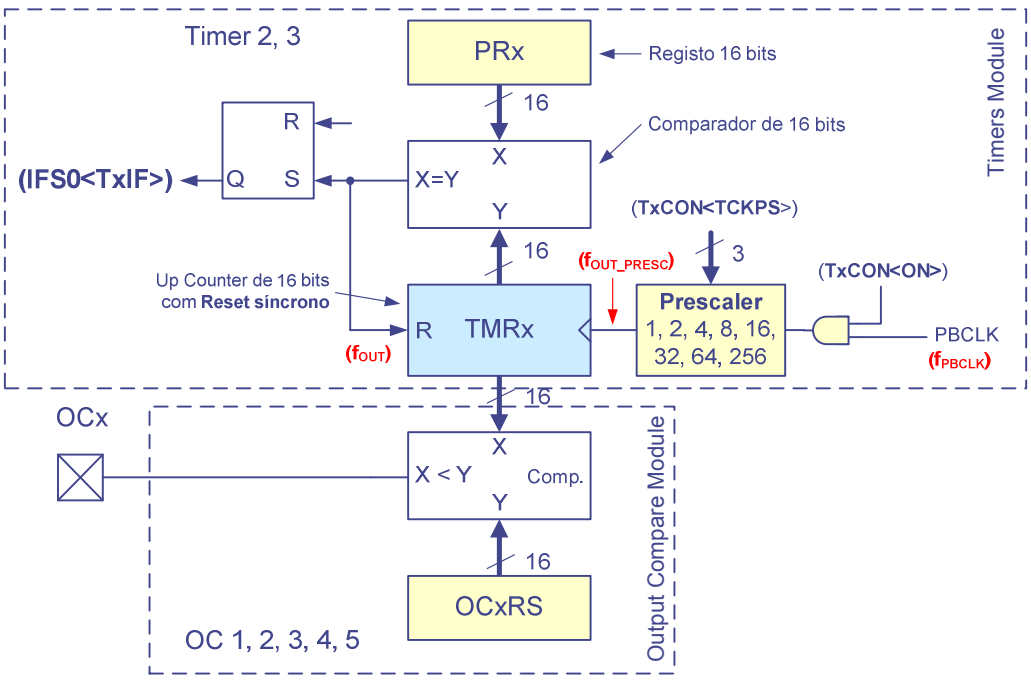


**Figura 2. Diagrama de blocos do sistema de geração de sinais PWM.**

Nesta forma de organização do sistema de geração de sinais PWM, um dos *timers* T2 ou T3 funciona como base de tempo, isto é, define o período T do sinal, enquanto que o módulo OC permite configurar, através do registo **OCxRS**, a duração a 1 desse sinal, isto é, o tempo $t_{ON}$.

**Exemplo**: determinar as constantes relevantes para a geração, na saída **OC1**, de um sinal com uma frequência de 10 HZ e um *duty-cycle* de 20%, usando como base de tempo o *timer* T2.

O valor de **PR2**, que determina a frequência do sinal de saída, foi já calculado na parte introdutória da aula anterior (**PR2=62499**). Temos então que calcular o valor da constante a colocar no registo **OC1RS**:

$$t_{ON} = 0.2 \times TPWM = 0.2 \times \left(\frac{1}{10}\right) = 20ms$$

$$f_{OUT\_PRESC} = 625KHz, \qquad T_{OUT\_PRESC} = \frac{1}{625000} = 1.6\mu s$$

Então **OC1RS** deverá ser configurado com:

$$OC1RS = \frac{20 \times 10^{-3}}{1.6 \times 10^{-6}} = 12500$$

Alternativamente, poderemos simplesmente multiplicar o valor de (**PRx + 1**) pelo valor do *duty-cycle* pretendido. Neste caso ficaria:

$$OC1RS = \frac{\left((PR2 + 1) * duty\text{-}cycle\right)}{100} = \frac{\left((62499 + 1) * 20\right)}{100} = 12500$$

Conhecendo os valores da frequência do sinal de saída (PWM) e do sinal à entrada do contador, pode calcular-se a resolução com que o sinal PWM pode ser gerado:

$$\text{Resolução} = \log_2\left(\frac{T_{PWM}}{T_{OUT\_PRESC}}\right) = \log_2\left(\frac{f_{OUT\_PRESC}}{f_{OUT}}\right)$$

Para as frequências do exemplo anterior a resolução é então: **$\log_2$(625000/10) = 15 bits**

A sequência completa de programação para obter o sinal de 10 Hz e *duty-cycle* de 20% na saída **OC1** fica então:

```
T2CONbits.TCKPS = 5; // 1:32 prescaler (i.e Fout_presc = 625 KHz)
PR2 = 62499;         // Fout = 20MHz / (32 * (62499 + 1)) = 10 Hz
TMR2 = 0;            // Reset timer T2 count register
T2CONbits.TON = 1;   // Enable timer T2 (must be the last command of the
                     //  timer configuration sequence)
OC1CONbits.OCM = 6;  // PWM mode on OCx; fault pin disabled
OC1CONbits.OCTSEL =0;// Use timer T2 as the time base for PWM generation
OC1RS = 12500;       // Ton constant
OC1CONbits.ON = 1;   // Enable OC1 module
```

O valor do registo **OC1RS** pode ser modificado, sem qualquer problema, em qualquer altura, sem necessidade de se alterar qualquer um dos outros registos. Isso permite a alteração dinâmica do *duty-cycle* do sinal gerado, em função das necessidades.

As saídas **OC1** a **OC5** estão fisicamente multiplexadas com os bits **RD0** a **RD4** do porto D (pela mesma ordem). A ativação do *Output Compare Module* **OCx** configura automaticamente o pino correspondente como saída, não sendo necessária qualquer configuração adicional (ou seja, esta configuração sobrepõe-se à efetuada através do registo **TRISD**).

## *Trabalho a realizar*

**Parte I**

1. Retome o exercício 6 da aula prática n.º 6. Nesse exercício implementou-se um sistema para adquirir o valor da tensão na entrada **AN4** da ADC e para visualizar o valor dessa tensão nos *displays* de 7 segmentos. A frequência de amostragem da ADC era 5 Hz e a frequência de refrescamento do sistema de visualização era 100 Hz. Estas frequências eram obtidas através do *Core Timer*, usando *polling*.

   Pretende-se agora a utilização de *timers* com atendimento por interrupção para controlar o funcionamento do sistema:

   - *timer* T1: determina a frequência de amostragem, i.e., o ritmo de leitura da entrada analógica;
   - *timer* T3: determina a frequência de refrescamento do sistema de visualização.

   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 10 ms (100 Hz).

   b) Escreva o programa principal, onde, no essencial, se faz a configuração de todos os dispositivos em utilização e se ativam globalmente as interrupções.

```
volatile int voltage = 0;   // Global variable

int main(void)
{
   configureAll(); // Function to configure all (digital I/O, analog
                   // input, A/D module, timers T1 and T3, interrupts)
   // Reset AD1IF, T1IF and T3IF flags
   EnableInterrupts();            // Global Interrupt Enable
   while(1);
   return 0;
}
```

   c) Escreva a rotina de serviço à interrupção do *timer* T1, onde deve ser dada a ordem de início de conversão à ADC.

```
void _int_(VECTOR_TIMER1) isr_T1(void)
{
   // Start A/D conversion
   // Reset T1IF flag
}
```

   d) Escreva a rotina de serviço à interrupção do *timer* T3, onde deve ser feito o envio para o sistema de visualização do valor de tensão calculado pela rotina de serviço à interrupção da ADC.

```
void _int_(VECTOR_TIMER3) isr_T3(void)
{
   // Send "voltage" value to displays (global variable)
   // Reset T3IF flag
}
```

   e) Integre no conjunto a rotina de serviço à interrupção da ADC (já implementada anteriormente).

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

Nesta fase o sistema deverá estar a funcionar integralmente por interrupção, convertendo o valor da tensão analógica presente na entrada **AN4** e a mostrar o respetivo valor nos dois *displays*.

## Parte II

1. Escreva um programa que gere na saída **OC1** um sinal com uma frequência de 100 Hz e um *duty-cycle* de 25%, utilizando como base de tempo o *timer* T3. Observe o sinal com o osciloscópio (no ponto de teste **OC1** da placa DETPIC32-IO) e verifique se os tempos do sinal (período e tempo a 1, $t_{ON}$) estão de acordo com o programado.
2. Escreva uma função que permita (para a frequência de 100 Hz) configurar o módulo **OC1** para gerar qualquer valor de *duty-cycle* entre 0 e 100, passado como argumento.

```
void setPWM(unsigned int dutyCycle)
{
   // duty_cycle must be in the range [0, 100]
   OC1RS = ...;   // Determine OC1RS as a function of "dutyCycle"
}
```

3. Teste a função anterior com outros valores de *duty-cycle*, por exemplo, 10%, 65% e 80%. Para todos os valores de *duty-cycle* meça, com o osciloscópio, o tempo $t_{ON}$ e o período do sinal.
4. Observe, para os diferentes valores de *duty-cycle* da alínea anterior, que o brilho do **LED D11** (ligado ao porto **RC14** da placa DETPIC32-IO) depende do valor do *duty-cycle* do sinal de PWM gerado. Para isso configure como saída o porto **RC14** e altere o programa principal de modo a ler, em ciclo infinito, o valor do porto **RD0** e a escrever o valor lido no porto **RC14**.

```
while(1)
{
   // Read the value of port RD0 and write it on port RC14
}
```

## *Exercícios adicionais*

1. Pretende-se dotar o sistema, que implementou na parte 1, com uma funcionalidade adicional que permita a paragem temporária da conversão, ficando os *displays* a mostrar o último valor de tensão medido (*freeze)*. Para isso configure os portos **`RB1`** e **`RB0`** como entrada e faça as alterações ao código que permitam parar a conversão quando o valor lido desses dois portos tiver a combinação binária "**`01`**" (**`RB1=0`**; **`RB0=1`**). Sugestão: controle o bit de *enable*/*disable* das interrupções do *timer* T1 (*timer* que controla o ritmo de conversão da ADC).

2. Pretende-se agora integrar o controlo do duty-cycle do sinal gerado na saída OC1, no programa que escreveu no ponto anterior. Para isso, os bits **`RB1`** e **`RB0`** vão ser usados para escolher o modo de funcionamento do sistema:

   ```
   00 – funciona como voltímetro; o duty-cycle deve ser 0
   01 – congela o valor atual da tensão; o duty-cycle deve ser 100%
   1X – o duty-cycle depende do valor da tensão medido pelo sistema
   ```

   Para fazer depender o *duty-cycle* do valor da tensão medido pelo sistema (disponível na variável global "**`voltage`**") poderá fazer **`dutyCycle = 3 * voltage`**, e obterá valores entre 0 e 99.

```
volatile int voltage;

int main(void)
{
   int dutyCycle;
   configureAll();
   EnableInterrupts();  // Global Interrupt Enable
   while(1)
   {
      // Read RB1, RB0 to the variable "portVal"
      switch(portVal)
      {
         case 0:  // Measure input voltage
            // Enable T1 interrupts
            setPWM(0);
            break;
         case 1:  // Freeze
            // Disable T1 interrupts
            setPWM(100);
            break;
         default:
            // Enable T1 interrupts
            dutyCycle = voltage * 3;
            setPWM(dutyCycle);
            break;
      }
   }
   return 0;
}
```

## *Elementos de apoio*

- Slides das aulas teóricas.
- 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.

**PDF criado em 20/05/2022**