Sumário do Resumo V1

timer0

O que é o timer0?

O Timer0 é um dos timers disponíveis no microcontrolador PIC 16F877A. Ele é um **contador de 8 bits** que pode ser utilizado para várias finalidades, como **gerar atrasos temporais**, **medir intervalos de tempo**, **controlar eventos *periódicos***, entre outros. Principais características:

- **Temporizador/contador de 8 bits;**
- **Leitura e gravação;**
- **Software prescaler de 8 bits programável;**
- **Relógio interno ou externo selecionável;**
- **Interrupção no estouro de FFh para 00h;**
- **Borda de seleção para relógio externo.**

O Timer0 é controlado por **registradores específicos**, como o **TMR0** (*que **armazena o valor do contador***), o **OPTION_REG** (*que **configura as opções do Timer0***) e o **INTCON** (*que **lida com as interrupções relacionadas ao Timer0***).

O Timer0 pode operar em diferentes modos de funcionamento, dependendo das configurações dos registradores. Alguns dos modos de operação mais comuns são:

1. Modo Temporizador: *Timer0 é configurado para **contar um número específico de pulsos de clock** e, quando esse número é atingido, é gerada uma interrupção ou uma ação pode ser tomada no código. Esse modo é **útil para gerar atrasos temporais precisos.***

2. Modo Contador: *Timer0 **conta o número de eventos externos que ocorrem em seu pino de entrada**, chamado **T0CKI**. Por exemplo, pode-se utilizar o Timer0 para **contar o número de pulsos de um sinal externo ou medir a frequência** de um sinal.*

3. Modo PWM: *Timer0 pode ser configurado para gerar um sinal PWM (Pulse Width Modulation) em um pino específico, permitindo o controle da largura do pulso do sinal gerado. Isso é **útil para aplicações como controle de motores, controle de brilho de LEDs**, entre outros.*

Além disso, o **Timer0 pode ser configurado com prescaler**, *que é um divisor de frequência que reduz a taxa de contagem do timer*. Isso **permite ajustar a precisão e a faixa de tempo que o Timer0 pode contar**.

Configuração dos modos do timer0

TABLE 5-1: REGISTERS ASSOCIATED WITH TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other RESETS
01h,101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
81h,181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

T0CS - Define a **fonte de clock** do TIMER0:
1 - Externa (PINO RA4/T0CKIN) - ***Modo contador***;
0 - Interna (Cristal de quartzo) - ***Modo temporizador***;

T0SE - Seleção da **borda de transição** do pino RA4/T0CKIN
1 - transição nível alto-baixo (transição negativa);
0 - transição nível baixo-alto (transição positiva);

PSA - Habilitação do **prescaler** (no modo CONTADOR é comum desabilitar);
1 - Desabilitado;
0 - Habilitado;

PS2, PS1, PS0 - Seleção do prescaler (somente no **modo temporizador**);
bits / TMR0 rate
000 1:2
001 1:4
010 1:8
011 1:16
100 1:32
101 1:64
110 1:128
111 1:256

Modo Temporizador

Ativado quando se usa o oscilador interno do PIC ou o cristal de quartzo externo. E o prescaler é o divisor de frequência do clock do timer0, usado no modo temporizador para poder obter tempos maiores.

Fórmula do tempo para o overflow:

$$ciclo\ de\ máquina = \frac{4}{(frequência\ do\ clock\ do\ cristal\ oscilador)}$$

$$t = (ciclo\ de\ máquina) \cdot (prescaler) \cdot contagem(256 - TMR0)$$

O clock de um PIC é de 20MHz, utilizado o prescaler de 1:256 e contagem de 0 a 256, qual será o tempo?

$$ciclo\ de\ máquina = \frac{4}{20 \cdot 10^{6} s^{-1}} = 0,2 \cdot 10^{-6} s = 0,2 \mu s$$

$$t = (0,2 \mu s) \cdot (256) \cdot contagem(256 - 0) = 0,2 \cdot 256 \cdot 256$$

$$t = 13107,2 \mu s \quad ou \quad t = 13,1072 ms$$

Para ter tempos ainda maiores, como alguns segundos,um dos meios de se conseguir é utilizando a interrupção.

Códigos

Code

```
unsigned contagem;


void interrupt()
{
contagem++;       //para cada interrpção do TMR0 incrementa a variável contagem
TMR0 = 0;
INTCON.TMR0IF = 0;  //limpa o overflow.
}


void main()
{
INTCON.GIE = 1; //habilita interrupção global;
INTCON.PEIE = 1; //habilita interrupção dos perifericos;
INTCON.TMR0IE = 1; //habilita interrupção do TMR0;
TMR0 = 0; // TIMER0 inicia com o valor 0;
OPTION_REG = 0b10000001; // Modo Temporizador, prescaler 1:4;
//tempo =1us * 4 * 256 = 1ms
...
while (1)
{
   if(contagem == 1000)    //quando o contagem = 1000, tempo = 1000 * 1ms = 1s;
   {
     portb.RB0 = ~portb.RB0;  //inverte o estado do pino rb0
     contagem = 0; //reseta a variavel contagem
   }
}
}
```

```c
#include <16F877A.h>  // Biblioteca do PIC 16F877A
#use delay(clock=4000000)  // Define a frequência do clock

unsigned int counter = 0;  // Variável de contador global

#int_timer0  // Tratamento de interrupção do Timer0
void timer0_isr()
{
   counter++;  // Incrementa o contador

   // Aqui você pode adicionar outras ações a serem realizadas na interrupção do Timer0, se necessário
}

void main()
{
   setup_timer_0(RTCC_INTERNAL | RTCC_DIV_256);  // Configuração do Timer0 no modo temporizador

   enable_interrupts(INT_TIMER0);  // Habilita a interrupção do Timer0
   enable_interrupts(GLOBAL);  // Habilita as interrupções globais

   while (1)
   {
      // Resto do código principal

      // Exemplo: Aguarda até que o contador atinja um determinado valor
      if (counter >= 1000)
      {
         // Ação a ser executada quando o contador atingir 1000
         counter = 0;  // Reinicia o contador
      }
   }
}
```

Nesse exemplo, utilizamos a diretiva **#int_timer0** para definir a função de tratamento de interrupção do Timer0. Dentro dessa função, incrementamos o contador **counter**. Você pode adicionar outras ações dentro dessa função, caso necessário.

No **main()**, configuramos o Timer0 com **setup_timer_0** no modo temporizador, usando uma fonte de clock interna (**RTCC_INTERNAL**) e um prescaler de 256 (**RTCC_DIV_256**). Em seguida, habilitamos a interrupção do Timer0 com **enable_interrupts(INT_TIMER0)** e as interrupções globais com **enable_interrupts(GLOBAL)**. q

Dentro do **loop principal**, podemos realizar outras tarefas e utilizar o contador **counter** para controlar a lógica desejada. No exemplo fornecido, aguardamos até que o contador atinja um valor de 1000 e executamos uma ação específica. Após essa ação, reiniciamos o contador **counter** para recomeçar a contagem.

Modo Contador

Ativado quando se usa como oscilador os pulsos externos aplicado no pino RA4/T0CKIN.

Códigos

```
void main()
{
TRISB=0;
OPTION_REG=0b10111000; //modo contador
TMR0=0;                    //Inicializa o registro TMR0 com o valor 0.
while(1)
{
  if(INTCON.TMR0IF)
  {
     INTCON.TMR0IF=0;    //limpa a flag
     TMR0 = 0;
     portb=~portb; //quando acontecer o overflow(a cada 256 pulsos) interver as saidas
  }
delay_ms(10);
}
}
```

O registro TMR0 é o registro que armazena o estado inicial da contagem. Possui 8 bits, por isso conta de 0 a 255. Quando acontece o overflow ele retorna a contagem inicial 0. O bit do overflow é o INTCON.TMR0IF e deve ser limpo por software.

```c
#include <16F877A.h>        // Biblioteca do PIC 16F877A
#use delay(clock=4000000)   // Define a frequência do clock

unsigned int count = 0;  // Variável de contagem global

#int_ext  // Tratamento de interrupção externa
void ext_isr()
{
   count++;  // Incrementa a contagem

   // Aqui você pode adicionar outras ações a serem realizadas na interrupção externa, se necessário
}

void main()
{
   setup_timer_0(T0_EXTERNAL | T0_EDGE_RISING);  // Configuração do Timer0 no modo contador

   enable_interrupts(INT_EXT);  // Habilita a interrupção externa
   enable_interrupts(GLOBAL);  // Habilita as interrupções globais

   while (1)
   {
      // Resto do código principal

      // Exemplo: Aguarda até que a contagem atinja um determinado valor
      if (count >= 1000)
      {
         // Ação a ser executada quando a contagem atingir 1000
         count = 0;  // Reinicia a contagem
      }
   }
}

```

Nesse exemplo, utilizamos a diretiva **#int_ext** para definir a função de tratamento de interrupção externa, que é acionada quando ocorre uma transição de subida no pino **T0CKI**, utilizado como entrada do Timer0 no modo contador. Dentro dessa função, incrementamos a variável **count**. Você pode adicionar outras ações dentro dessa função, caso necessário.

No **main()**, configuramos o Timer0 com **setup_timer_0** no modo contador, utilizando uma fonte de clock externa (**T0_EXTERNAL**) e detecção na transição de subida (**T0_EDGE_RISING**) do sinal no pino **T0CKI**.

Habilitamos a interrupção externa com **enable_interrupts(INT_EXT)** e as interrupções globais com **enable_interrupts(GLOBAL)**.

Dentro do **loop principal**, podemos realizar outras tarefas e utilizar a variável **count** para controlar a lógica desejada. No exemplo fornecido, aguardamos até que a contagem atinja um valor de 1000 e executamos uma ação específica. Após essa ação, reiniciamos a variável **count** para recomeçar a contagem.

Modo PWM

Código

```
#include <16F877A.h>        // Biblioteca do PIC 16F877A
#use delay(clock=4000000)   // Define a frequência do clock

void main()
{
   setup_ccp1(CCP_PWM); // Configuração do CCP1 no modo PWM

   // Configuração do Timer0 no modo PWM
   setup_timer_0(T0_INTERNAL | T0_DIV_1);
   set_pwm1_duty(127); // Define o ciclo de trabalho inicial

   while (1)
   {
      // Resto do código principal
   }
}

```

Nesse exemplo, utilizamos a função setup_ccp1 para configurar o módulo CCP1 (Capture/Compare/PWM) no modo PWM. O parâmetro CCP_PWM indica que o CCP1 será utilizado para gerar um sinal PWM.

Em seguida, configuramos o Timer0 no modo PWM utilizando a função setup_timer_0. Os parâmetros T0_INTERNAL e T0_DIV_1 definem a fonte de clock interna do Timer0 sem prescaler.

Utilizamos a função set_pwm1_duty para definir o ciclo de trabalho inicial do sinal PWM gerado pelo CCP1. O valor passado como argumento (127 no exemplo) determina a proporção entre o tempo em nível alto e o período do sinal PWM. Valores menores resultam em um ciclo de trabalho menor, enquanto valores maiores resultam em um ciclo de trabalho maior.

Dentro do loop principal, você pode realizar outras tarefas do programa. Mais detalhes do Resumo de PWM.