Universidade de São Paulo-USP
Escola de Engenharia de São Carlos-EESC
SEL0629 - Aplicações de Microprocessadores I

Vinicius William da Silva - Número USP: 11233842
Prof. Marcelo Andrade da Costa Vieira

São Carlos, 2023

1 Introdução

Nesta prática utilizei o microcontrolador PIC18F45k22 por meio do Kit de Desenvolvimento EasyPIC v.7 da MikroEletronika. O objetivo da prática consiste em aprender a utilizar I/O, temporizador e interrupções do PIC18F45k22 ao desenvolver um sistema que, ao apertar um botão, aciona um display de 7 segmentos que conta de 0 a 9 em loop.

Mais especificamente, quando um botão na porta RB0 for pressionado, um display de 7 segmentos ligado na Porta D deve contar (de 0 a 9 em loop) com período de 1s. Quando um botão na porta RB1 for pressionado, o mesmo display de 7 segmentos deve contar com período de 0.25s. Para tanto, utilizei o temporizador TMR0 com interrupção para gerar as bases de tempo do contador e considerei a frequência do clock do PIC igual a 8 MHz.

O uso das interrupções necessárias para esta prática é feito configurando os registradores intcon, intcon2, intcon3. Em intcon é necessário habilitar as interrupções globais (GIE/GIEH), de periféricos (PEIE/GIEL) e as interrupções específicas que deseja-se utilizar (timer0 e externas). E importante também configurar a prioridade das interrupções (se houver prioridade).

O timer0 utilizado na prática deve ser configurado corretamente por meio do registrados t0con, onde define-se quantos bits terá o timer0, se funcionará como contador ou temporizador, se está parado ou não e o valor do prescaler (timer0 não possui postscaler).

A prática foi validada utilizando um osciloscópio. Foi utilizado o ponto decimal, ligado a porta RD7, deixando-o aceso em números impares e apagado em número pares, de modo que fosse possível medir o período da onda quadrada gerada pela configuração do temporizador.

2 Resultados Experimentais

2.1 Esquemático

O esquemático gerado no software **Proteus** está a seguir.

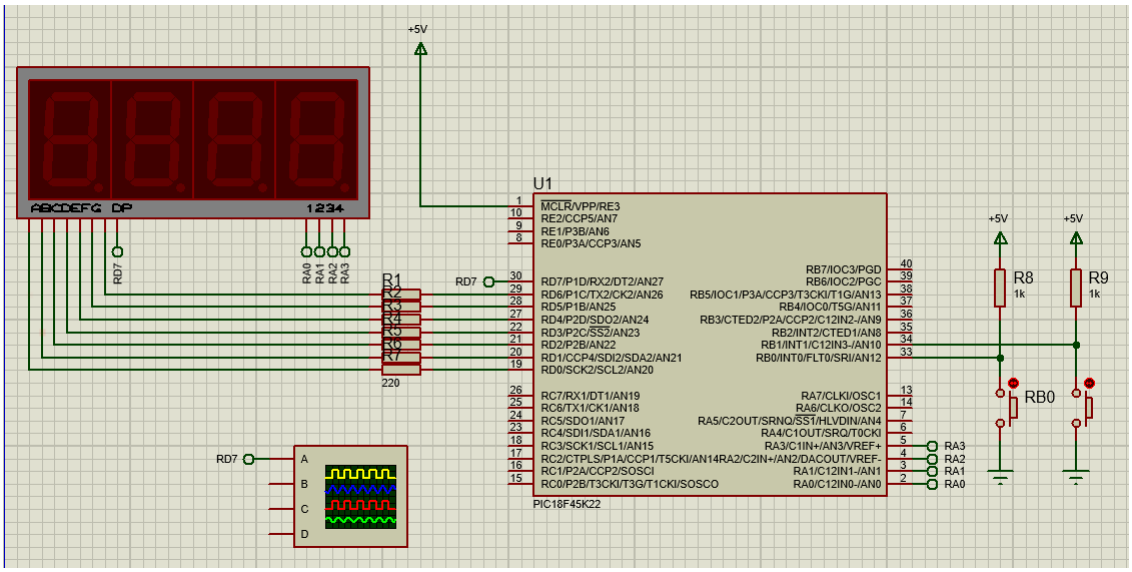

Figura 1: Esquemático do circuito.

2.2 Códigos e Simulação

As variáveis globais necessárias para serem manipuladas no código são inicialmente declaradas, bem como os defines para as portas RB0 e RB1, nas quais serão conectados os botões.

```
#define but1 RB0_bit
#define but025 RB1_bit

int counter = 0x00;
bit but1f;
unsigned char catodo, contSeg = 0;
unsigned char segmento[] = {
        0b00111111,
```

```
    0b00000110,
    0b01011011,
    0b01001111,
    0b01100110,
    0b01101101,
    0b01111101,
    0b00000111,
    0b01111111,
    0b01100111
};
```

Na função **main**, temos a configuração dos registradores necessários. Primeiro, é configurado o registrador T0CON, que habilita o *Timer0* e define o *prescaler* como 1:4. Além disso, também são configuradas as portas RB0 e RB1 como inputs digitais e iniciadas em high. Como mostrado no esquemático, estou usando resistores de *pull up* externos para que o nível lógico da porta seja 1 quando o botão não estiver ativo.

```
T0CON = 0b11001001; //prescaler 1:4
anselb = 0;
TRISB = 0x03;      // RB0 e RB1 como inputs
PORTB = 0x03;      //RB0 e RB1 iniciam em high
RBPU_bit = 1;
```

Para o *display* de 7 segmentos e 4 dígitos (utilizado o catodo comum na simulação), deve ser colocado na porta D e, além disso, deve ser feita a correta multiplexação da porta A para que apenas 1 dos 4 *displays* seja utilizado. O código abaixo mostra a habilitação das interrupções, bem como a atribuição do *display.*

```
//selecionar display
ANSELA = 0;
TRISA = 0;
 // Somente o primeiro display sera usado
```

```
  LATA =  0b0001;    //seleciona digito 0 do display

  TRISD = 0x00;          //output nos ports D
  PORTD = segmento[0];   // inicia em zero

  GIE_bit = 0x01;    //habilita interrup??o global
  PEIE_bit = 0x01;   //habilita interrup??o por perif?ricos
  T0IE_bit = 0x01;   //habilita interrup??o por estouro do
TMR0
  T0IF_bit = 0x00;

  TMR0L = 0x00;       //timer inicia em zero

  but1f = 1;
```

Para o *looping while*, a lógica é que o contador de ciclos do Timer0 seja utilizado de forma a completar o ciclo com 1 segundo e com 0.25 segundos. Para tanto, foi utilizada a fórmula, considerando apenas 8 bits no contador:

$$t = \frac{1}{8Mhz} \cdot prescaler \cdot 256$$

Como o prescaler foi selecioando como 1:4, tem-se:

$$t = 128\ \mu s$$

Portanto, para 1 segundo e 0.25, deve-se ter, respectivamente, 2000 ciclos e 500 ciclos. Para saber em qual passo estará contado, basta checar a *flag* "but1f", que dirá qual botão foi acionado por último. Uma vez estourado o counter, basta atribuir à porta D o valor correspondente do vetor de segmentos e zerar o contador, como feito no código abaixo.

```
 while(1)
  {
   if(but1f == 1)      // 1 sec
```

```
    {
       if(counter >= 8000) // 0.128 us *4 * 256 * 8000 = 1.0
sec
       {
          contSeg++;
          if(contSeg == 10) contSeg = 0;
          catodo = segmento[contSeg];          //update
display
          PORTD = catodo;
          latd.f7 = contSeg % 2;
          counter = 0;
       }
     }
     else
     {
        if(counter >= 2000) //approx 0.25sec cicle
        {
          contSeg++;
          if(contSeg == 10) contSeg = 0;
          catodo = segmento[contSeg];          //update
display
          PORTD = catodo;
          latd.f7 = contSeg % 2;
          counter = 0;
        }
     }
   } //end while
```

Por fim, basta executar a rotina de interrupção para tratar o *overflow* do Timer0 e também para checar o estado dos botões. Nesse caso, a *flag* e o *timer* precisam ser limpados.

```
    void interrupt()
```

```
{
     if(T0IF_bit)       //estouro timer0?
     {
      counter++;
      TMR0L = 0x00;        //reseta timer
      T0IF_bit = 0x00;     //clear flag
      T0IE_bit = 0x01;   //habilita interrup??o por estouro do
   TMR0

      if(!but1)
      {
       but1f = 0x01;
      }else if(!but025)
      {
        but1f = 0x00;
      }

     }
}
```

3 Resultados e Discussão

Para avaliar a precisão do temporizador, utilizei um osciloscópio para medir a frequência e o período do ponto decimal do *display* na porta RD7.

O osciloscópio apontou os seguintes resultados:

Tabela 1: Resultados obtidos por osciliscópio.

Taxa de atualização (ms)	Frequência medida (mHz)	Período medido (ms)
1000	471,65	2120,0
250	1890,0	530,0

Em ambos os casos, houveram desvios que, provavelmente, devem-se à resistência e capacitância parasitas de trilha, comuns em circuitos em geral. Ademais, há um erro inerente à precisão do osciloscópio e *delays* devido à geração de código pelo compilador, podendo gastar ciclos de máquina adicionais.

Analisando por período, tem-se a tabela abaixo com os desvios.

Tabela 2: Comparação dos resultados reais e esperados.

Taxa de atualização (ms)	Período esperado (ms)	Período medido (ms)	Desvio (%)
1000	2000,0	2120,0	6
250	500,0	530,0	6

4 Conclusão

Ao comparar os resultados, observou-se uma precisão razoável. A porcentagem de erro consideravelmente elevada pode ser pelos excessivos ciclos de máquina utilizados no código. Com o *prescaler* utilizado, precisou-se de muitos ciclos de Timer0 para alcançar o tempo requerido, com mais ciclos de máquina, mais tempo de execução é somado ao tempo final, ou seja, é possível reduzir ainda mais esse tempo utilizando um *prescaler* mais condizente e tentando reduzir o número de instruções necessárias em cada iteração.