**UNIVERSIDADE DE SÃO PAULO**
**ESCOLA DE ENGENHARIA DE SÃO CARLOS**
**ENGENHARIA ELÉTRICA - ÊNFASE EM ELETRÔNICA**


**SEL0433 - Aplicação de Microprocessadores (2025)**

**Docente - Prof. Pedro Oliveira C. Junior**

**Relatório da Atividade Semanal 08**

Érico Antônio C. Paes Leme
Nº USP: 14746206
Heloísa Oliveira de Carvalho
Nº USP: 13833960
Ulisses O. Basile
Nº USP: 14749255
Yuri Thadeu Oliveira Costa
Nº USP: 14754821

São Carlos – SP
31 de maio de 2025

# Sumário

# 1 Exercício 01

## 1.1 Parte 01 - Piscando um LED usando TMR3

Foi implementado um programa em C para o PIC18F4550, no qual o Timer3 é configurado com prescaler 1:8 e valor inicial zero. A cada overflow do timer (aproximadamente 262 ms), o LED conectado ao pino RD0 inverte seu estado, resultando em um piscar completo (ligar e desligar) a cada 0,52 segundos. Assim, o LED pisca aproximadamente 2 vezes por segundo. A configuração T3CON do Tmr3 foi feita de acordo com o datasheet disponibilizado e as instruções da atividade. A simulação via SimulIDE funcionou corretamente.

**Código implementado:**

```
// Exercicio 01 - TMR3 - Atividade Semanal 8 - Erico Heloisa Ulisses Yuri
void main() {
     // Configuracao do PORTD como saida
    TRISD.RD0 = 0; // Configura RD0 como saida
    PORTD.RD0 = 0;  // Garante que RD0 inicie desligado (LED apagado)
    // Configuracao do Timer3
    T3CON = 0b11110001;
    /*
       T3CON = 0b10110001 significa:
       bit7:  1 = Prescaler 1:8
       bit6 e 3: 10 = Timer3 is the capture/compare clock source for both CCP modules
       bit5-4:  11  -> Timer3 Clock source = internal clock (Fosc/4)
       bit2:   0 = Synchronize external clock input
       bit1:   0 = Internal clock (Fosc4)
       bit0:  1 - Timer3 On
    */
    // Inicializa o TMR3 com zero
    TMR3H = 0;
    TMR3L = 0;
    PIR2.TMR3IF = 0; //zera a flag de estouro do tmr3
    while(1){
     if(PIR2.TMR3IF == 1){
      PORTD.RD0 = ~LATD.RD0;
      TMR3H = 0;        // recarrega o timer3
      TMR3L = 0;        // recarrega o timer3
      PIR2.TMR3IF = 0; //zera a flag de estouro do tmr3
     }
    }
}
```

Listing 1: Código-fonte do projeto em linguagem C

**Simulação no SimulIDE:**

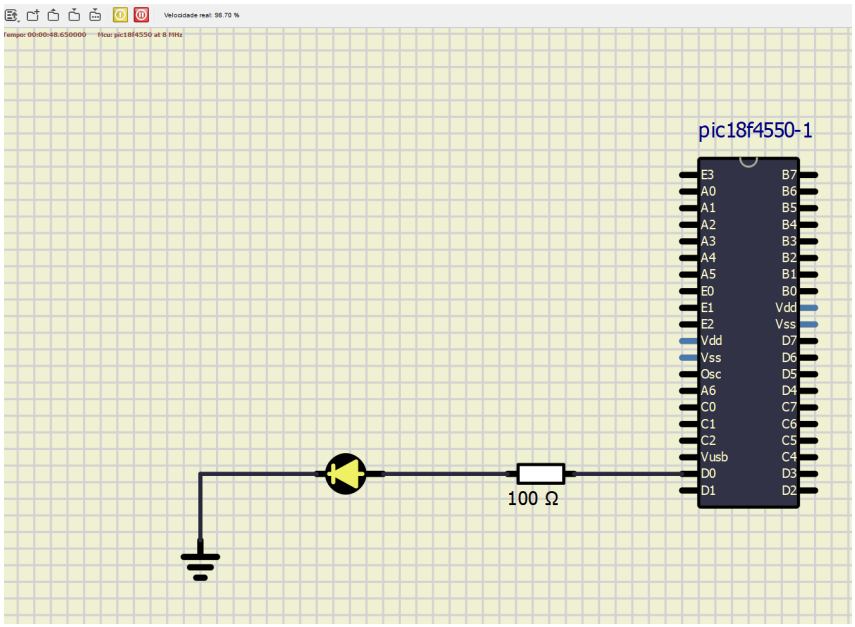


Figura 1: Circuito funcional - Exercício 01, parte 01 - Elaborado no programa SimulIDE

## 1.2 Parte 02 - Piscando um LED usando TMR2

Foi implementado um programa em C para o PIC18F4550 utilizando o Timer2 configurado com o prescaler e postscaler máximos (ambos 1:16) e o registrador PR2 definido com 255 para maximizar o tempo até o overflow do contador interno. Como o Timer2 é um contador de 8 bits que conta de zero até o valor definido em PR2, usar 255 permite a maior contagem possível (256 incrementos). Com a configuração dos escalonadores, o tempo total até a flag de overflow ser acionada é de aproximadamente 32,7 ms. A cada overflow, o LED no pino RD0 inverte seu estado, fazendo com que ele pisque com um período completo de cerca de 65,5 ms, ou seja, aproximadamente 15 vezes por segundo. A simulação via SimulIDE demonstrou o funcionamento correto do código.

**Código implementado:**

```
// Exercicio 01 - TMR2 - Atividade Semanal 8 - Erico Heloisa Ulisses
    Yuri
void main2() {
     // Configuracao do PORTD como saida
    TRISD.RD0 = 0; // Configura RD0 como saida
    PORTD.RD0 = 0;  // Garante que RD0 inicie desligado (LED apagado)
    // Configuracao do Timer2
    T2CON = 0B01111111;              // TMR2 ligado, prescaler e
       postscaler 16.
    PR2 = 255; // Valor maximo em termos de possibilidades para PR2
    PIR1.TMR2IF = 0; //zera a flag de estouro do tmr2
    while(1){
     if(PIR1.TMR2IF == 1){
      PORTD.RD0 = ~LATD.RD0;
      PIR1.TMR2IF = 0; //zera a flag de estouro do tmr2
     }
    }
}
```

Listing 2: Código-fonte do projeto em linguagem C

**Simulação no SimulIDE:**

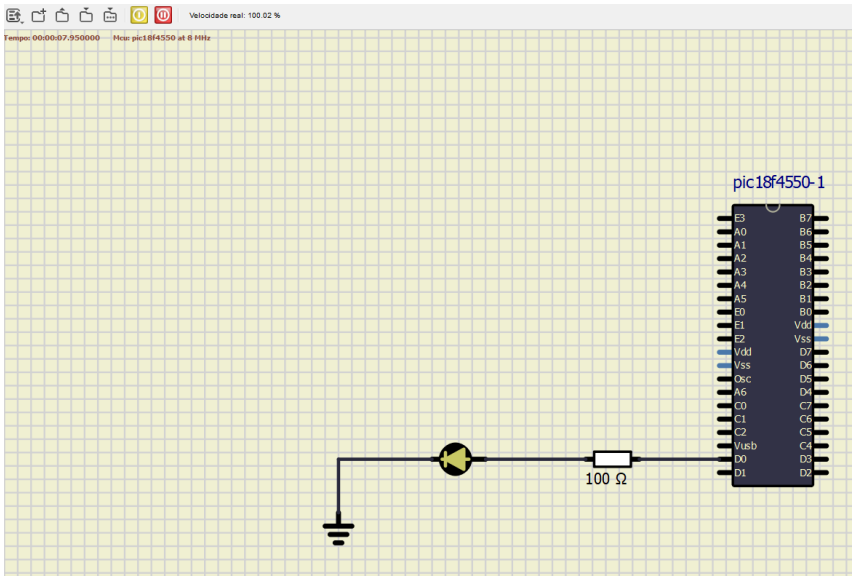


Figura 2: Circuito funcional - Exercício 01, parte 02 - Elaborado no programa SimulIDE

# 2 Exercício 02

## 2.1 Parte 01 - Interrupção via INT2

Foi implementado um programa em linguagem C no compilador MikroC PRO for PIC para o microcontrolador PIC18F4550 com objetivo de acionar um LED por meio da interrupção externa INT2. O LED está conectado ao pino RD0 (configurado como saída) e muda seu estado (liga/desliga) sempre que o botão conectado ao pino RB2 (INT2) for pressionado. A interrupção foi configurada para ocorrer na borda de descida do sinal, ou seja, quando o botão for pressionado. O botão está em configuração pull-up interna,

dispensando resistor externo. As flags e permissões globais e periféricas foram devidamente habilitadas, e o tratamento da interrupção é realizado na rotina interrupt(), onde a flag INT2IF é manualmente limpa após cada acionamento. A simulação via SimulIDE mostrou o funcionamento correto do código, em que cada vez que o botão é apertado o estado do led muda.

**Código implementado:**

```
// Exercicio 02 - Interrupcao INT2 - Atividade Semanal 8 - Erico Heloisa Ulisses Yuri
void interrupt() iv 0x0008 ics ICS_AUTO {
     // Logica de tratamento da interrupcao externa INT2
     if (INTCON3.INT2IF == 1) {
         PORTD.RD0 = ~LATD.RD0;        // Inverte o estado do LED
         INTCON3.INT2IF = 0;         // Limpa a flag da INT2
     }
}
// Configuracao das entradas/saidas, flags e interrupcoes:
void main() {
     TRISD.RD0 = 0;        // RD0 como saida
     PORTD.RD0 = 0;        // LED comeca desligado

     TRISB = 0XFF;       // RB como entrada
     INTCON2.RBPU = 0;  // Habilita pull-up interno para o PORTB

     INTCON3.INT2IE = 1;    // Habilita interrupcao externa INT2
     INTCON3.INT2IF = 0;    // Limpa flag da INT2
     INTCON3.INT2IP = 1;    // Prioridade alta para INT2

     INTCON2.INTEDG2 = 0;  // Interrupcao na borda de descida (pressionar botao)
     INTCON.GIE = 1;        // Habilita interrupcoes globais
     INTCON.PEIE = 1;       // Habilita interrupcoes perifericas

     while (1) {
         // Loop infinito, o programa depende totalmente das interrupcoes.
     }
}
```

Listing 3: Código-fonte do projeto em linguagem C

**Simulação no SimulIDE:**

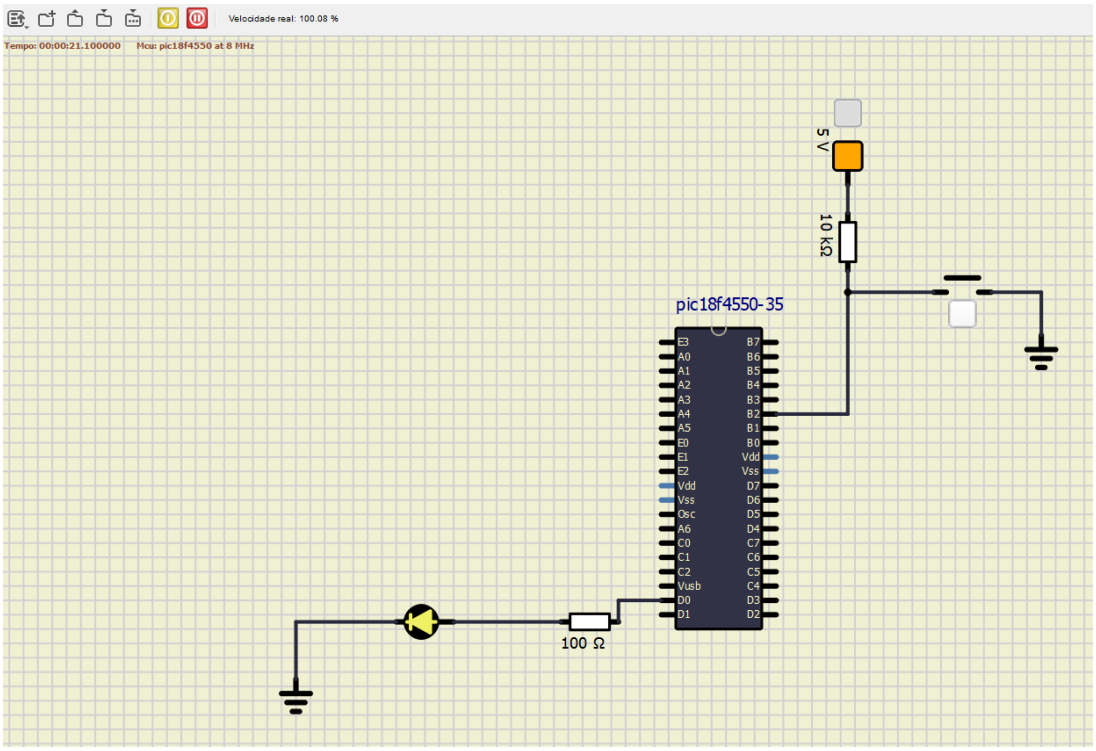


Figura 3: Circuito funcional - Exercício 02, parte 01 - Elaborado no programa SimulIDE

## 2.2 Parte 02 - Interrupção via TMR1

Foi implementado um programa que utiliza o Timer1 do PIC18F4550 no modo contador externo para contar pulsos gerados pela pressão de um botão conectado ao pino T13CKI (RC0). O Timer1, que é um contador de 16 bits, foi inicializado com o valor 65531 (0xFFFB) para que, ao receber 5 pulsos, ocorra um estouro (overflow) gerando uma interrupção. Isso acontece porque o Timer1 conta de 65531 até 65535 e, ao ultrapassar esse limite, volta a zero, acionando a flag de interrupção. Na rotina de tratamento da interrupção, o estado do LED em RD0 é invertido (ligando ou desligando), o Timer1 é recarregado para o valor inicial e a flag de interrupção é zerada, permitindo contar novamente os próximos 5 eventos. Assim, a cada 5 pulsos no botão, o LED muda seu estado. O código e a configuração do TMR1, bem como a escolha do pino T13CKI, foi feito de acordo com o datasheet disponibilizado e as instruções da atividade. A simulação via SimulIDE mostrou o funcionamento correto do código, em que a cada 5 vezes que o botão é apertado, o estado do led muda.

**Código implementado:**

```
// Exercicio 02 - Timer1 contador externo - Atividade Semanal 8 - Erico Heloisa Ulisses Yuri

void interrupt() iv 0x0008 ics ICS_AUTO {
    if (PIR1.TMR1IF == 1) {          // Se ocorreu overflow do Timer1 (5 pulsos)
        PORTD.RD0 = ~LATD.RD0;       // Acende o LED
        PIR1.TMR1IF = 0;             // Limpa flag de interrupcao do Timer1
        TMR1H = 0xFF;                // Recarrega o Timer1 para contar mais 5 eventos
        TMR1L = 0xFB;                // 0xFFFB = 65531
    }
}

void main() {
    TRISD.RD0 = 0;           // RD0 saida (LED)
    PORTD.RD0 = 0;           // LED apagado inicialmente

    TRISC.RC0 = 1;           // RC0 (T13CKI) entrada para pulsos externos (botao)

    // Configurar Timer1 como contador externo
    T1CON = 0;               // Zera T1CON
    T1CON.TMR1CS = 1;        // Fonte de clock externa no pino T13CKI (RC0)
    T1CON.TMR1ON = 1;        // Liga o Timer1

    // Inicializa Timer1 para contar ate 5 eventos
    TMR1H = 0xFF;
    TMR1L = 0xFB;            // 65531 decimal

    // Configura interrupcoes
    PIR1.TMR1IF = 0;         // Limpa flag de interrupcao do Timer1
    PIE1.TMR1IE = 1;         // Habilita interrupcao do Timer1
    INTCON.PEIE = 1;         // Habilita interrupcoes perifericas
    INTCON.GIE = 1;          // Habilita interrupcoes globais

    while(1) {
        // Loop principal vazio - operacao controlada pela interrupcao
    }
}
```

Listing 4: Código-fonte do projeto em linguagem C

**Simulação no SimulIDE:**

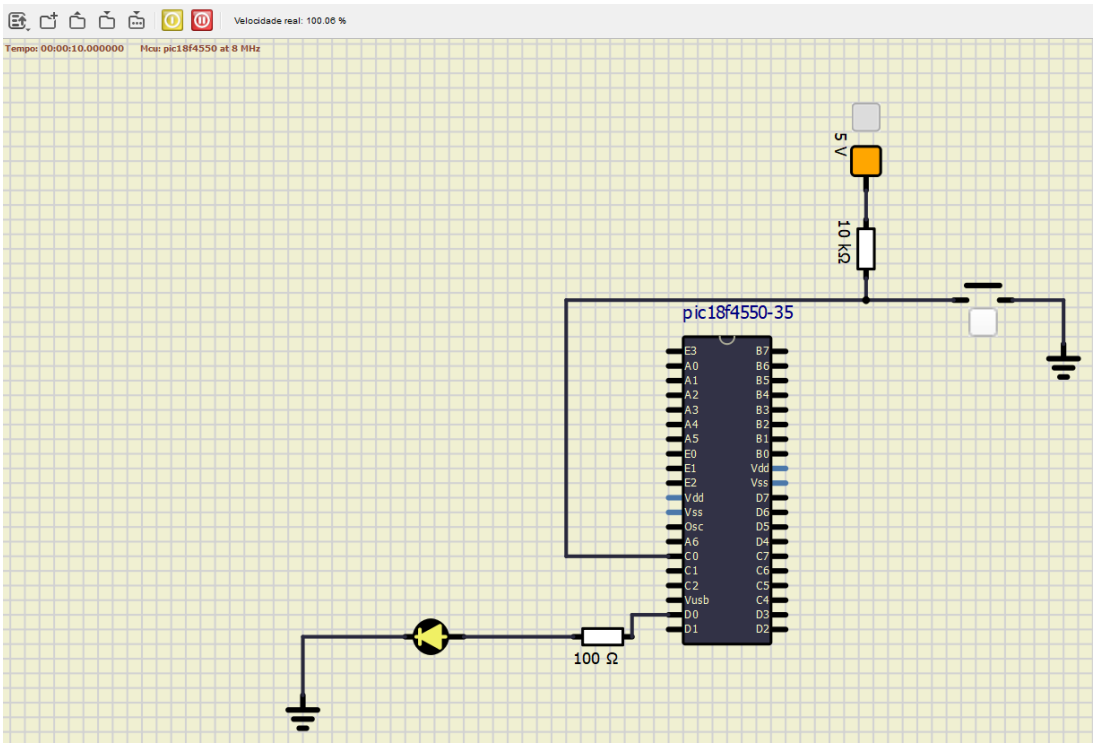


Figura 4: Circuito funcional - Exercício 02, parte 02 - Elaborado no programa SimulIDE

# 3 Exercício 03

O que acontece se o PIC18F estiver ocupado com uma interrupção de baixa prioridade e uma interrupção de alta prioridade ocorrer?

a) O microcontrolador sempre atenderá primeiro às interrupções de alta prioridade, independentemente de uma interrupção de baixa prioridade já estar em andamento.

b) O PIC18F4550 não possui um mecanismo de priorização de interrupções, e todas as interrupções são tratadas na ordem em que são geradas, sem considerar a importância ou a prioridade das interrupções.

c) O microcontrolador pode continuar processando a interrupção de baixa prioridade até que ela seja concluída, mesmo que uma interrupção de alta prioridade seja gerada.

d) As interrupções de alta prioridade podem interromper a execução de interrupções de baixa prioridade, mas após o processamento da interrupção de alta prioridade, o microcontrolador retorna automaticamente à interrupção de baixa prioridade.

**Resposta: A resposta correta é a letra d)**