Prática 1 - Semáforo de Pedestres utilizando RTOS

Átila Camurça

12 de janeiro de 2017

1 Introdução

Em sistemas embarcados não existe a figura de um sistema operacional responsável por criar e gerenciar threads, mas é possível usar o príncipio do RTOS (Real-Time Operating System).

Uma das bibliotecas possíveis de uso é a OSA, um microkernel para microcontroladores como o Microchip PIC-controllers PIC10, PIC12, PIC16, PIC18, PIC24, dsPIC; Atmel AVR 8-bit controllers, e STMicroelectronics STM8.

Um sistema RTOS permite que o programador foque em tarefas orientadas a problemas (algoritmos, fórmulas matemáticas, etc.) e não precise se preocupar com tarefas secundárias. Todas as tarefas secundárias são executadas pelo kernel do OSA, tais como:

- mudança entre processos paralelos (exemplo: leitura de teclado, saída de dados para um LCD, acionamento de relés);
- verificação de *timeout*s, contagem de *delay*s;
- troca de dados entre diferentes tarefas usando semáforos, mensagens, filas, etc.

2 Objetivo

O objetivo é refazer o mesmo projeto da disciplina de Microcontroladores que consistia em fazer um semáforo de carros e pedestres com botão de acionamento, mas agora usando um sistema RTOS.

3 Código fonte

```
#include "SanUSB48.h"
#include <osa.h>
//Evitar uso de outros laços dentro das tasks (como for, do - while, etc.!)

#define zero   0b0111111
#define um     0b0000110
#define dois   0b1011011
#define tres   0b1001111
#define quatro 0b1100110
#define cinco  0b1101101
#define seis   0b1111101
#define sete   0b0000111
```

```
#define oito   0b1111111
#define nove   0b1101111

#define pin_dezena pin_c0
#define pin_unidade pin_c1

#define verde_carro pin_a0
#define amarelo_carro pin_a1
#define vermelho_carro pin_a2

#define vermelho_pedestre pin_a3
#define verde_pedestre pin_a4

unsigned char seg[10] = {
    zero,
    um,
    dois,
    tres,
    quatro,
    cinco,
    seis,
    sete,
    oito,
    nove,
};
int i = 0, dezena = 0, unidade = 0;
int flag_pedestre = 0, ligar_displays = 0;

#pragma interrupt interrupcao

void interrupcao() {
    if (PIR1bits.TMR1IF) {
        PIR1bits.TMR1IF = 0;
        TMR1H = 0xD8;
        TMR1L = 0xF0;
        OS_Timer();
    }
}

void PIC_Init(void) {
    LATB = 0x00;
```

```
    TRISB = 0b00000000;
    TRISC = 0b000;

    T1CON = 0x80; // modo 16 bits
    TMR1H = 0xD8; // 1ms
    TMR1L = 0xF0;

    INTCON = 0;
    INTCONbits.PEIE = 1;
    PIR1bits.TMR1IF = 0; // Flag interrupcao Timer1
    PIE1bits.TMR1IE = 1; // Habilita interrupcao Timer1
    T1CONbits.TMR1ON = 1; // Liga Timer1
}

void task_multiplexacao_display_7seg(void) {
    while (1) {
        if (ligar_displays) {
            nivel_baixo(pin_dezena);
            nivel_alto(pin_unidade);
            PORTB = seg[dezena];
            OS_Delay(5);

            nivel_baixo(pin_unidade);
            nivel_alto(pin_dezena);
            PORTB = seg[unidade];
            OS_Delay(5);
        } else {
            OS_Delay(100);
        }
    }
}

void task_handle_7seg_number(void) {
    while (1) {
        if (ligar_displays) {
            dezena = i / 10;
            unidade = i % 10;
            if (i > 20) {
                i = 0;
                dezena = 0;
                unidade = 0;
```

```
                OS_Delay(500);
                ligar_displays = 0;
            } else {
                i++;
                OS_Delay(1000);
            }
        } else {
            OS_Delay(100);
        }
    }
}

void task_semaforo(void) {
    while (1) {
        if (flag_pedestre) {
            nivel_baixo(verde_carro);
            nivel_alto(amarelo_carro);
            OS_Delay(1000);
            nivel_baixo(amarelo_carro);

            nivel_alto(vermelho_carro);
            nivel_baixo(vermelho_pedestre);
            ligar_displays = 1;
            nivel_alto(verde_pedestre);
            nivel_baixo(verde_carro);
            OS_Delay(1000 * 20);
            flag_pedestre = 0;
        } else {
            nivel_alto(vermelho_pedestre);
            nivel_baixo(vermelho_carro);

            nivel_alto(verde_carro);
            nivel_baixo(verde_pedestre);
        }
        OS_Delay(100);
    }

}

void task_escuta_botao_pedestre(void) {
    while (1) {
```

```
        if (!entrada_pin_e3) {
            flag_pedestre = 1;
        }
        OS_Delay(100);
    }
}

void main(void) {
    clock_int_48MHz();

    PIC_Init();
    OS_Init();

    OS_Task_Create(0, task_multiplexacao_display_7seg);
    OS_Task_Create(1, task_escuta_botao_pedestre);
    OS_Task_Create(1, task_handle_7seg_number);
    OS_Task_Create(2, task_semaforo);

    OS_EI();
    OS_Run();
}
```

No método `main` o OSA é inicializado com a função `OS_Init`, em que ele usa o um dos *timer*s do microcontrolador com *clock*.

Depois foram necessárias as *task*s:

- `task_multiplexacao_display_7seg` - responsável pela multiplexação dos 2 *display*s de 7 segmentos.
- `task_escuta_botao_pedestre` - responsável por alternar *flag* para o botão do pedestre.
- `task_handle_7seg_number` - responsável por incrementar e resetar as variáveis de dezena e unidade.
- `task_semaforo` - responsável por acionar os LEDs.

Por fim é executado o método `OS_Run` que funciona como o `while(1){}` ou um *loop* infinito.

4 Montagem do circuito

A montagem é a mesma, e pode ser vista na Figura 1.

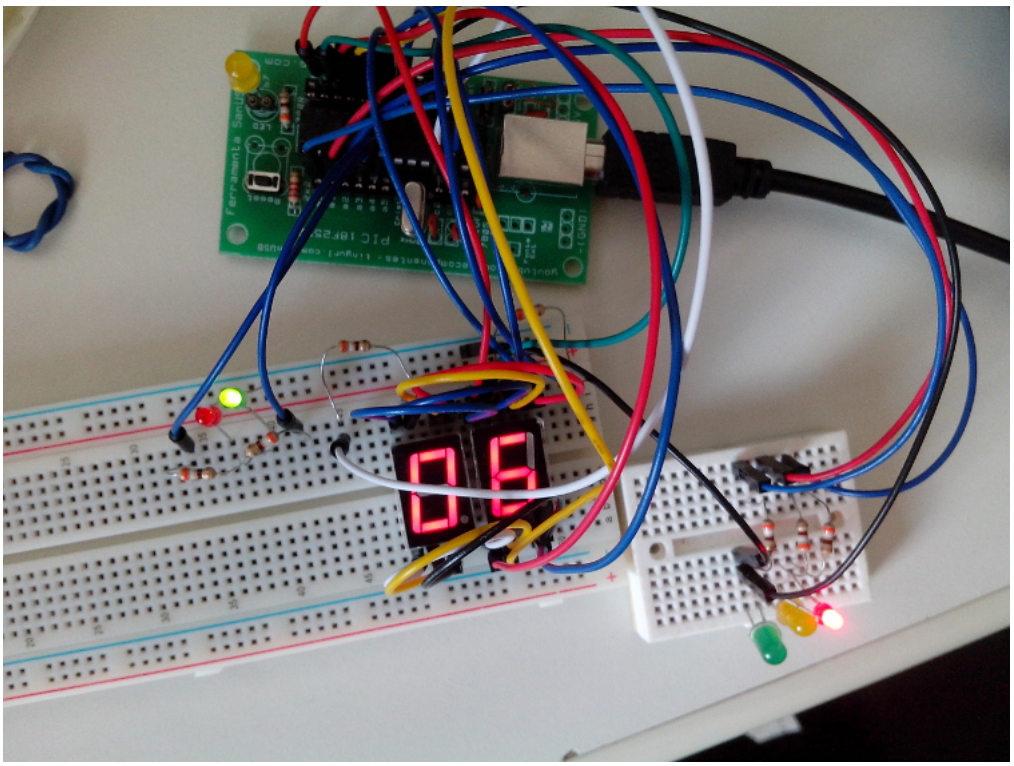

Figura 1: Circuito