

Arquitetura de Computadores II

Exercícios adicionais da componente prática (guiões 7 a 11)

Exercício 1

Escreva um programa, em linguagem C, que execute em ciclo infinito, e que, sempre que o utilizador prime uma tecla no PC, faça as seguintes tarefas:

- tecla 0: deve acender apenas o LED ligado ao porto RE0, e deve aparecer nos *displays* o valor 00;
- tecla 1: deve acender apenas o LED ligado ao porto RE1, e deve aparecer nos *displays* o valor 01;
- tecla 2: deve acender apenas o LED ligado ao porto RE2, e deve aparecer nos *displays* o valor 02;
- tecla 3: deve acender apenas o LED ligado ao porto RE3, e deve aparecer nos *displays* o valor 03;
- outra tecla qualquer: os 4 LED devem acender e permanecer ligados durante 1s; no final desse tempo os 4 LED devem ser apagados (este deve ser o seu estado inicial). Simultaneamente, deve aparecer nos *displays* o valor FF durante 1s e a seguir os dois *displays* devem apagar-se (i.e., todos os segmentos devem ser desligados). Qualquer tecla premida dentro desse intervalo de tempo deve ser ignorada.

A frequência de refrescamento dos *displays* deve ser 100 Hz, obtida por interrupção do Timer 2. Para controlar a temporização correspondente ao intervalo de tempo de 1s, deve utilizar o *core timer* do PIC32.

Exercício 2

- Escreva um programa em linguagem C que implemente um contador crescente módulo 100, atualizado com uma frequência de 10 Hz. O valor do contador deve ser mostrado em decimal nos dois *displays* da placa e deve ser impresso no ecrã, em hexadecimal, formatado com 2 dígitos. A atualização do contador deve ser feita por interrupção do Timer 1. A frequência de refrescamento dos *displays* deve ser 50Hz, controlada pelo Timer 2.
- Altere o programa que escreveu em a) de modo a que, adicionalmente, quando o utilizador prime uma das teclas 0 a 4 a frequência de incremento do contador seja modificada de acordo com a seguinte expressão: **freq = 2 * (1 + tecla_premida)**. Quando for detetado que o utilizador premiu uma tecla diferente (dentro da gama) deve ser impressa no ecrã a mensagem "**Nova frequência:**", seguida do valor da tecla (0 a 4).
- Altere o programa que escreveu em b) de modo a que a frequência de incremento do contador seja dada pela seguinte expressão (recorde que a ADC tem uma resolução de 10 bits):

$$freq = 1 + \frac{VAL_{ADC}}{127} [Hz]$$

A frequência de amostragem da ADC deve ser 4 Hz, obtida através da utilização do Core Timer, e o fim de conversão da ADC deve ser processado por interrupção.

Exercício 1

PR2 ≤ 65535

$$PR2 = \frac{20000000}{100} - 1 = 199999$$

→ Não cabe num registro de 16 bits

1) Prescaler = 4

$$PR2 = \frac{20000000}{4 \cdot 100} - 1 = 49999$$

Prescaler

1 000 0

2 001 1

4 010 2 → # T2CONbits: TCKPS = 2;

PR2 = 49999;

TMR2 = 0;

T2CONbits.TON = 1;

Exercício 3

- Escreva um programa para gerar na saída OC3 um sinal PWM, com uma frequência de 1kHz, cujo *duty-cycle* seja controlado pelo potenciômetro que está ligado à entrada AN4 da ADC do PIC32: ao valor máximo da conversão deve corresponder o *duty-cycle* 100% e ao valor mínimo deve corresponder o *duty-cycle* 0%. A frequência de amostragem da ADC deve ser 10 Hz, obtida através da utilização do Core Timer. O fim de conversão da ADC deve ser processado por *polling*. Utilize o Timer 3 como referência para a geração do sinal PWM.
- Mostre nos 2 *displays* de 7 segmentos o valor atual do *duty-cycle*, entre 00 e 99. Quando o *duty-cycle* atingir 100%, deve acender o LED ligado ao porto RC14 (LED D11 na placa) e, nos *displays*, deve ser apresentado o valor 00 (em conjunto com o LED indica um *duty-cycle* de 100%).
- Acrescente à resolução que fez em b) a geração de um segundo sinal PWM, na saída OC5, usando como referência o mesmo timer. Para este sinal, ao valor máximo da conversão A/D deve corresponder o *duty-cycle* 0% e ao valor mínimo deve corresponder o *duty-cycle* 100%.

Se possível, verifique os resultados com o auxílio de um osciloscópio (com duas pontas de prova para o exercício b).

Nota: Em ambos os exercícios deve configurar o timer de referência de modo a maximizar a resolução do sinal PWM gerado.

Exercício 4

Retome o exercício anterior. Faça as alterações que permitam processar o fim de conversão da ADC por interrupção e acrescente o módulo de visualização de modo a mostrar nos dois *displays* de 7 segmentos o valor do *duty-cycle*. Quando o *duty-cycle* for 100%, deve ser mostrado o valor 00 nos dois *displays*, e deve ser ativado o LED que está ligado ao porto **RC14**. A frequência de atualização dos *displays* deve ser 100 Hz, obtida por interrupção do Timer 5.

Exercício 5

Escreva um programa que imprima no ecrã, à frequência de 2 Hz, o valor de cada um dos portos RB3 a RB0, sem utilizar qualquer um dos *system calls* disponíveis (por exemplo, se os portos RB_{3:0} tiverem a combinação binária "1100", no ecrã deverá aparecer "RB30=1100").

Para isso, configure a UART2 com os parâmetros de comunicação 1200 bps, 8 data bits, sem paridade, 1 stop bit, e escreva as funções de transmissão de um carater e de transmissão de uma *string* que lhe permitam enviar a informação para o PC. A transmissão do carater deve ser feita por *polling*.

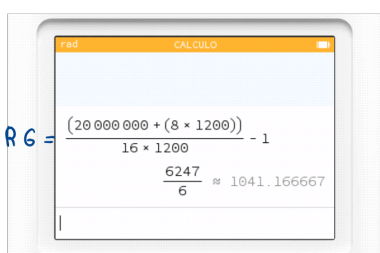
Para o teste deste exercício não se esqueça de executar o "pterm" com os mesmos parâmetros de comunicação com que configurou a UART.

Configuração:

• U2MODEbits.BRGH = 0;

• Ver datasheet - Paridade, STOP-0.1s, ...

• $U \times BRG =$


$$U \times BRG = \frac{(20000000 + (8 \times 1200))}{16 \times 1200} - 1$$
$$\frac{6247}{6} \approx 1041.166667$$

• U2STAbits.UTXEN =

• 1;

• U2STAbits.URXEN =

• U2MODEbits.ON = 1;

Exercicio 3

$$1 \text{ kHz} = 1000 \text{ Hz}$$

$$0 \leq \text{duty-cycle} \leq 100$$

$$\text{ADC} \rightarrow 10 \text{ Hz}$$

Timer 3 - PWM

$$\text{Display} \rightarrow \text{duty-cycle} \quad (\text{if } \text{d_cycle} = 100) \quad \{$$

$$\text{RC14} = 1;$$

Exercício 6

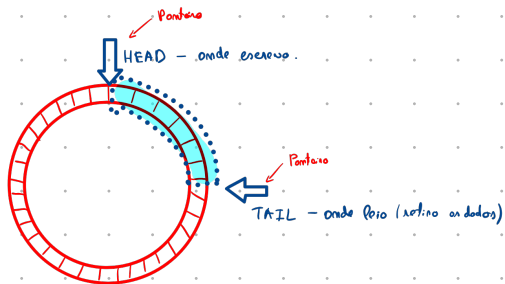
Escreva um programa que leia, à frequência de 1.5 Hz, o valor dos 4 bits dos portos RB3 a RB0 e que guarde esse valor num *buffer* circular com 16 posições (quando o buffer ficar cheio deve ser descartado o valor mais antigo). Para gerar esta temporização deve usar o Core Timer.

Configure a UART2 com os parâmetros de comunicação 57600 bps, 8 data bits, paridade ímpar, 1 stop bit, e escreva as funções de transmissão de um carácter e de transmissão de uma *string*, por *polling*. Acrescente ainda o código necessário para processar a receção de um carácter por interrupção.

Quando for recebido o carácter 'D' deve ser enviado para o PC o número de valores armazenados no *buffer* circular, seguido desses valores, enviados por ordem cronológica de ocorrência. Adicionalmente, o número de valores do buffer deve ser repostado a zero.

Sugestão: transforme o valor armazenado numa dada posição do *buffer* circular numa *string* binária e use a função de envio de uma *string* para fazer a transmissão do valor para o PC (por exemplo: #elem: 5 – 1001, 1001, 0000, 0011, 1010).

Buffer circular



Suponhamos que queres um buffer de 8 elementos:

```
c
#define BUF_SIZE 8

char buffer[BUF_SIZE];
int head = 0; // escreve
int tail = 0; // lê
int count = 0; // quantos elementos há no buffer
```

Inserir :

```
void buffer_put(char data) {
    if(count < BUF_SIZE) {
        buffer[head] = data;
        head = (head + 1) % BUF_SIZE;
        count++;
    }
    // Se o buffer estiver cheio, podes ignorar, sobrescrever ou dar erro
}
```

Remover :

```
char buffer_get(void) {
    char data = 0;

    if(count > 0) {
        data = buffer[tail];
        tail = (tail + 1) % BUF_SIZE;
        count--;
    }

    return data;
}
```