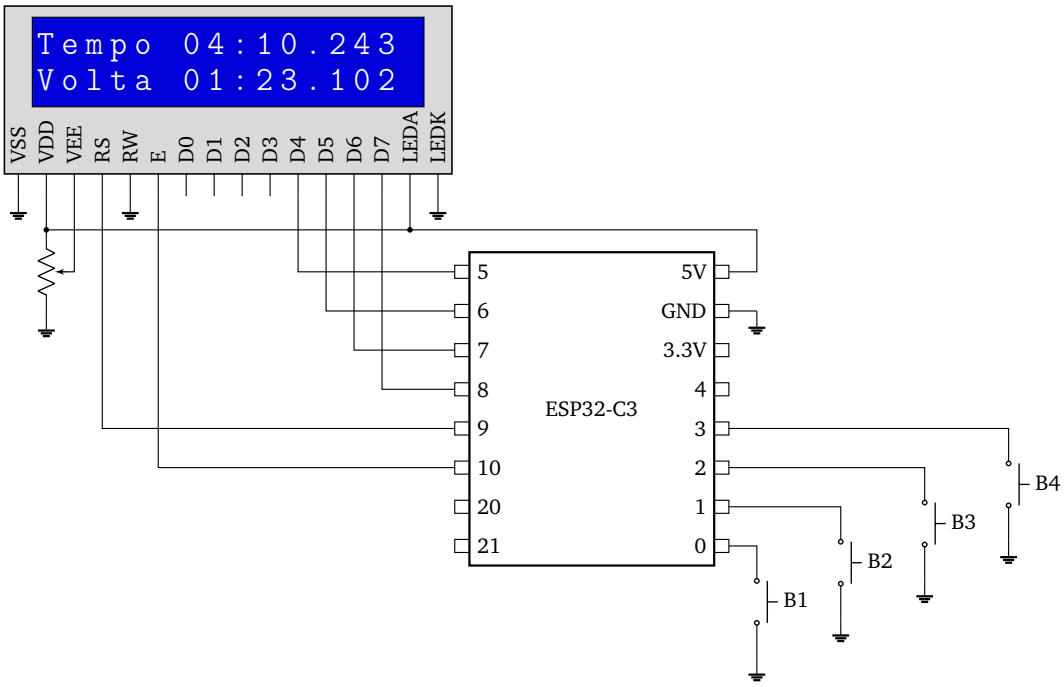
UFSC – EEL7030 Microprocessadores

Projeto 1

Cronômetro Digital

O objetivo do projeto é desenvolver um cronômetro digital que mostre o tempo decorrido (minutos, segundos e milésimos de segundo) e a última volta, com controle de início, pausa, *reset* e *lap* por botões usando interrupções. O *hardware* disponibilizado consiste em um ESP32-C3 conectado a um *display* LCD 16x2 e quatro botões táteis, conforme o esquemático abaixo:



Para alcançar o objetivo do projeto, você deve escrever um código em C para o ESP32-C3 que atenda aos seguintes critérios:

- O código deve ser implementado de forma legível, com nomes de variáveis claras, **utilizando os conceitos já abordados na disciplina**. Utilize o componente disponibilizado no Moodle para o controle do *display* LCD.
- O seu código deve estar **totalmente compreendido em um único arquivo .c** (com exceção do componente disponibilizado).
- Sempre que possível, **modularize seu código em funções auxiliares** que executam ações específicas. Por exemplo, ao invés de configurar as rotinas de interrupção dos botões direto na função principal, crie uma função auxiliar que executa os passos necessários para essa configuração e chame a função auxiliar na função principal.
- O *display* deve exibir, na primeira linha, o tempo decorrido a partir do início do cronômetro e, na segunda linha, o tempo da última volta.
  - O padrão exibido na primeira linha deve ser: “Tempo MM:SS.mmm”;

  - O padrão exibido na segunda linha deve ser: “Volta MM:SS.mmm”;
  - Para os casos anteriores, ‘MM’ corresponde aos 2 dígitos dos minutos, ‘SS’ corresponde aos dois dígitos de segundos e ‘mmm’ corresponde aos três dígitos de milésimos de segundo;
  - A contagem de tempo do cronômetro deve ser realizada utilizando um temporizador de *hardware* específico para essa função (não utilize temporizadores de *software*).
- Os controles de início, pausa, marcação de volta e reinício do cronômetro devem ser realizados através de 4 botões táteis utilizando interrupções externas. O cronômetro só deve ser reiniciado se estiver parado. Ao reiniciar, todas as informações devem ser zeradas (tanto o tempo, quanto a volta).
  - O Botão 1 (B1) inicia o cronômetro; o Botão 2 (B2) pausa o cronômetro; o Botão 3 (B3) marca a volta; o Botão 4 (B4) reinicia o cronômetro.
  - Você deve implementar o *debouncing* dos botões utilizando temporizadores de *software*. Estude a biblioteca esp_timer disponível como material extra do Roteiro 7.

Critérios de Avaliação

O projeto deve ser feito de forma **individual**. A nota do projeto será calculada de acordo com a seguinte fórmula:

$$\text{NP} = \frac{A}{10}\left(\frac{8C}{10} + \frac{2D}{10}\right)$$

Onde:

- A é a nota da arguição, podendo variar entre 0 e 10.
  - A arguição será realizada na última aula reservada ao projeto (ver cronograma), **exclusivamente no ambiente de sala de aula**, durante o horário da aula. A nota será atribuída de forma que 0 representa nenhum domínio sobre o código desenvolvido e 10 representa domínio completo sobre o código desenvolvido.
  - O aluno será arguido com perguntas sobre sua implementação do projeto, devendo respondê-las em, no máximo, 5 minutos.
  - Durante a arguição, será permitido acesso **apenas** ao código desenvolvido. Dessa forma, é vedada a pesquisa em qualquer ambiente (Google, IAs, etc). Se observado o uso de qualquer ambiente de pesquisa, a nota da arguição será considerada 0.
- C é a nota do código, podendo variar entre 0 e 10.
  - O código será avaliado quanto a sua legibilidade, adequação à estrutura padrão de códigos em C e o desenvolvimento das funcionalidades do projeto. **Serão considerados desenvolvimentos parciais das funcionalidades**. Veja a seção “Estrutura de Código e Documentação” para mais detalhes.
  - As funcionalidades avaliadas serão:

    - configuração do temporizador de *hardware* e alarme (1,00 ponto);
    - configuração de ISR para os botões táteis (1,00 ponto);
    - implementação do *debounce* para os botões táteis (1,00 ponto);
    - inicialização do *display* LCD (1,00 ponto);
    - implementação da lógica necessária para exibição do cronômetro (3,00 pontos);
    - implementação da lógica para comandar o cronômetro (iniciar, parar, marcar volta e reiniciar) (3,00 pontos);
- D é a nota da documentação, podendo variar entre 0 e 10.
  - A documentação será avaliada se foi devidamente implementada e as funções estão claramente descritas. Veja a seção "Estrutura de Código e Documentação" para mais detalhes.

Importante! Códigos que sejam copiados entre os alunos terão sua nota zerada (todos os alunos envolvidos).

Estrutura de Código e Documentação

A estrutura do código deve seguir o padrão apresentado no Roteiro 3, com pré-diretivas, variáveis globais, protótipos de funções auxiliares, função principal e declaração de funções auxiliares.

Adicionalmente, o código deve ser documentado utilizando o Doxygen. O Doxygen é uma ferramenta de geração de documentação automática para projetos escritos em linguagens como C, C++, Java, Python, entre outras. Ele extrai comentários especiais do seu código-fonte e os transforma em documentação legível em HTML, PDF, LaTeX, etc. É especialmente útil para projetos em C/C++ porque essas linguagens não têm suporte nativo a *docstrings*, como Python ou Java.

Utilizando o Doxygen

Você pode usar dois estilos principais:

```
/// Comentario de uma linha (estilo breve, por linha)

/**
* Comentario de multiplas linhas
* com tags como @param, @return, etc.
*/
```

Listagem 1: Exemplo de uso de documentação Doxygen

Os comentários de múltiplas linhas levam algumas tags para organização e geração do arquivo de documentação:

Tags do Doxygen

Tag	Descrição
@brief	Breve descrição da função ou item
@param	Descreve um parâmetro da função
@return	Descreve o valor de retorno
@file	Usada no início do arquivo para descrição
@note	Observações adicionais
@see	Referência a outra função ou arquivo
@warning	Aviso importante
@author	Autor do código

O seu arquivo de código deve começar com um cabeçalho, seguindo a seguinte estrutura:

```
/**
 * @file    arquivo.c
 * @brief   Descricao do breve do arquivo.
 *
 * Descricao mais detalhada do arquivo.
 *
 * @author  Nome do autor
 * @date    2025-05-20
 */
```

Listagem 2: Cabeçalho de início do código

Cada função deve ser documentada utilizando as tags descritas anteriormente, por exemplo:

```
/**
 * @brief Calcula o fatorial de um numero inteiro.
 *
 * Esta funcao usa recursao para calcular o fatorial de um numero n.
 *
 * @param n Numero inteiro nao negativo
 * @return O fatorial de n, ou -1 se n for negativo
 */
int fatorial(int n);

/**
 * @brief Soma dois inteiros.
 *
 * Esta funcao recebe dois inteiros e retorna sua soma.
 *
 * @param a Primeiro inteiro
 * @param b Segundo inteiro
 * @return Soma de a e b
 */
int soma(int a, int b);
```

Listagem 3: Exemplo de funções documentadas