embarca tech | Residência Tecnológica em Sistemas Embarcados

# Práticas de GPIO com delay

Unidade 4 | Capítulo 3

Tiago Façanha

Executores:
INSTITUTO FEDERAL Piauí
INSTITUTO FEDERAL Rio Grande do Norte
INSTITUTO FEDERAL Maranhão
INSTITUTO FEDERAL Ceará
cepedi
INSTITUTO HARDWARE BR

Coordenação: Softex

Iniciativa: MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

# Sumário

# Unidade 4
# Microcontroladores

## Capítulo 3

### 1. BOAS-VINDAS E INTRODUÇÃO

Olá, estudante do EmbarcaTech!

Seja bem-vindo a mais uma etapa do nosso curso de capacitação. Nesta unidade, vamos explorar práticas avançadas de controle de portas de entrada e saída (GPIO) utilizando temporização (delay). O foco desta aula é compreender como utilizar técnicas de atraso para controlar dispositivos eletrônicos como LEDs, displays de 7 segmentos e desenvolver sistemas mais complexos como um cofre digital.

Com base nos conhecimentos adquiridos anteriormente sobre GPIOs, estamos prontos para dar mais um passo em direção ao desenvolvimento de sistemas embarcados. Nesta aula, você terá a oportunidade de aplicar a programação em C para realizar projetos práticos e desafiadores. Vamos aprender a controlar o tempo de execução de nossos programas para sincronizar e coordenar múltiplos dispositivos e ações.
Vamos lá?

### 2. Objetivos desta Unidade

Ao final desta aula, espera-se que você tenha desenvolvido a capacidade de **utilizar técnicas de temporização** (delay) em sistemas embarcados com o Raspberry Pi Pico [2] ou com a plataforma educacional BitDogLab [1], configurando e controlando de forma eficiente as portas de entrada e saída (GPIO) para interagir com diversos dispositivos eletrônicos. Você será capaz de **implementar sistemas temporizados**, como um semáforo controlado por push-button, e controlar displays de 7 segmentos [6,8], utilizando rotinas de atraso para gerenciar a exibição de números e informações.

Você também estará preparado para aplicar a técnica de **multiplexação** em múltiplos displays de 7 segmentos, otimizando o uso dos pinos GPIO e garantindo uma **exibição eficiente e sincronizada** de dados. Além disso, terá a habilidade de desenvolver **aplicações mais complexas**, como um cofre digital com senha, empregando teclados matriciais para a entrada de dados e implementando lógica de controle e validação para garantir a segurança e a integridade do sistema.

Essas práticas permitirão que você adquira habilidades fundamentais para a resolução de problemas recorrentes em sistemas embarcados, como a **sincronização** de tarefas e a **otimização** dos recursos do microcontrolador. Você estará apto a aplicar essas técnicas em **projetos práticos**, utilizando tanto o Raspberry Pi Pico quanto a BitDogLab, expandindo sua capacidade de **criar soluções inovadoras e eficazes** para os desafios reais no desenvolvimento de sistemas embarcados.

## 3. temporização (delay) em sistemas embarcados

Nos capítulos anteriores, exploramos os conceitos básicos e intermediários de **manipulação de GPIOs**, utilizando a plataforma educacional BitDogLab e o microcontrolador Raspberry Pi Pico. Aprendemos a configurar pinos como entradas e saídas digitais, e a interagir com componentes eletrônicos como LEDs e botões. Discutimos a estrutura e as capacidades do microcontrolador RP2040, que integra múltiplos periféricos e interfaces, como GPIO, UART, I2C e SPI[4], permitindo a comunicação eficiente com dispositivos externos. Além disso, revisamos o uso das bibliotecas fornecidas no SDK do Raspberry Pi Pico, que facilitam a programação e o controle dos recursos do microcontrolador.

Nesta unidade, vamos aprofundar nosso conhecimento sobre a utilização de **temporização** (delay) em sistemas embarcados. A aplicação de delays é **fundamental** para **coordenar e sincronizar** eventos, permitindo a **criação de sistemas** que reagem de forma precisa a estímulos externos. Vamos entender como utilizar **rotinas de atraso** para controlar o tempo de execução de ações, como acender LEDs em sequência, exibir números em displays de 7 segmentos e implementar sistemas mais complexos que envolvem múltiplas interações com o usuário.

Utilizaremos a plataforma BitDogLab  e wokwi para ilustrar exemplos práticos, oferecendo um ambiente de aprendizado integrado e interativo. A BitDogLab, equipada com diversos periféricos como displays, botões e LEDs, será o suporte ideal para experimentarmos as funcionalidades de temporização e controlarmos o comportamento dos dispositivos em tempo real.

Hoje, vamos expandir esse conhecimento adquirido e explorar aplicações práticas de temporização, utilizando o microcontrolador Raspberry Pi Pico e a BitDogLab. Neste material, abordaremos novos conteúdos que incluem a **configuração de delays** para controlar LEDs de forma sequencial, a **criação de contadores** em displays de 7 segmentos e o desenvolvimento de sistemas interativos que combinam controle de tempo e entradas do usuário.

Este tópico é crucial porque possibilita o controle preciso de dispositivos eletrônicos e a coordenação de eventos em sistemas embarcados. A capacidade de utilizar temporização de maneira eficaz é essencial para o desenvolvimento de projetos que exigem **sincronização de tarefas e resposta a eventos temporais,** como semáforos, cronômetros e contadores. Dominar essas habilidades é um passo importante para a criação de soluções inovadoras e eficientes, tanto em ambientes educacionais quanto em aplicações industriais.

Nesta seção, vamos analisar **quatro exemplos práticos** que ilustram o uso dos GPIOs com temporização no desenvolvimento de projetos com o Raspberry Pi Pico, cada um explorado em um subtópico específico deste eBook:

| | | |
|---|---|---|
| **Exemplo 1** | Controle de semáforo temporizado com push-button | Demonstra como utilizar delays para controlar LEDs em um ciclo de semáforo, permitindo a interação com o usuário por meio de um botão. |
| **Exemplo 2** | Controle de display de 7 segmentos com temporização | Mostra como aplicar rotinas de atraso para exibir números de 0 a 9 em um display de 7 segmentos. |
| **Exemplo 3** | Multiplexação de displays de 7 segmentos | Discutiremos como controlar múltiplos displays de maneira eficiente, utilizando a técnica de multiplexação para alternar rapidamente entre eles e exibir contagens de 00 a 99. |
| **Exemplo 4** | Controle de cofre digital com senha | Será focado na estruturação de um sistema de segurança, no qual utilizaremos um teclado matricial para a entrada de uma senha de quatro dígitos e LEDs para indicar o status do cofre. |

**Fig. 1: quatro exemplos práticos do uso dos GPIOs com o Raspberry Pi Pico**
Fonte: Tabela elaborada pelo autor.

A imagem exibe uma tabela com quatro exemplos de projetos eletrônicos, cada um com um título e uma breve descrição. Vou descrever a tabela em detalhes:
- Exemplo 1:
- Título: Controle de Semáforo Temporizado com Push-Button.
- Descrição: Demonstra como utilizar delays para controlar LEDs em um ciclo de semáforo, permitindo a interação com o usuário por meio de um botão.
- Exemplo 2:
- Título: Controle de Display de 7 Segmentos com Temporização.
- Descrição: Mostra como aplicar rotinas de atraso para exibir números de 0 a 9 em um display de 7 segmentos.
- Exemplo 3:
- Título: Multiplexação de Displays de 7 Segmentos.
-Descrição: Discute como controlar múltiplos displays de maneira eficiente, utilizando a técnica de multiplexação para alternar rapidamente entre eles e exibir contagens de 00 a 99.
-Exemplo 4:
-Título: Controle de Cofre Digital com Senha.
- Descrição: Focado na estruturação de um sistema de segurança, utilizando um teclado matricial para entrada de senha de quatro dígitos e LEDs para indicar o status do cofre.
Cada exemplo está dividido em duas colunas: a primeira com o título do projeto e a segunda com sua descrição.

O exemplo 4, **Controle de Cofre Digital com Senha**, servirá como base para um desafio, no qual os alunos deverão implementar a lógica completa de funcionamento, explorando o uso de temporização para validar a senha digitada e gerenciar o bloqueio temporário do sistema após tentativas incorretas, simulando o comportamento de um cofre real.
Cada subtópico apresentará diagramas esquemáticos, códigos detalhados e explicações minuciosas, fornecendo uma referência

prática para o desenvolvimento de seus próprios projetos utilizando temporização em sistemas embarcados.

## 3.1. Implementar o controle de um semáforo temporizado com push-button

Neste primeiro exemplo prático, vamos explorar como utilizar os GPIOs do Raspberry Pi Pico para implementar um sistema de controle de semáforo, utilizando três LEDs para representar as cores de um semáforo real e um push-button para iniciar e pausar o ciclo de funcionamento. Essa atividade é essencial para compreender a configuração e a utilização de pinos de entrada e saída, bem como a aplicação de técnicas de temporização (delay) para o controle preciso de eventos. O controle de semáforo é um exemplo clássico em sistemas embarcados que envolve a interação com dispositivos externos e a sincronização de múltiplos sinais.

### Objetivo do Exemplo

O objetivo deste exemplo é implementar um sistema de controle de semáforo utilizando três LEDs (vermelho, amarelo e verde) e um push-button, simulando o funcionamento de um semáforo de trânsito. Vamos configurar os LEDs para acender em sequência, com tempos específicos para cada cor, e usar o push-button para iniciar e pausar o ciclo do semáforo.

### Materiais Necessários

Para realizar este experimento, você precisará dos seguintes componentes:

- 1 x Raspberry Pi Pico ou BitDogLab
- 1 x LED Vermelho
- 1 x LED Amarelo
- 1 x LED Verde
- 3 x Resistores de 330 ohms
- 1 x Push-Button

### Montagem do Circuito

1. **Conexão dos LEDs:**

- Conecte o ânodo (perna maior) do LED vermelho ao pino GPIO 11 do

Raspberry Pi Pico e o cátodo (perna menor) ao GND, através de um resistor de 330 ohms.

- Repita o procedimento para o LED amarelo no pino GPIO 12 e para o LED verde no pino GPIO 13, utilizando resistores de 330 ohms para limitar a corrente.

2. **Conexão do Push-Button:**

- Conecte um terminal do push-button ao pino GPIO 05 e o outro terminal ao GND.
- Adicione um resistor de pull-down (10k ohms) entre o pino GPIO 5 e o GND para evitar leituras instáveis.

A seguir, temos o diagrama esquemático do circuito:
Figura 1 - Diagrama do circuito do semáforo temporizado.

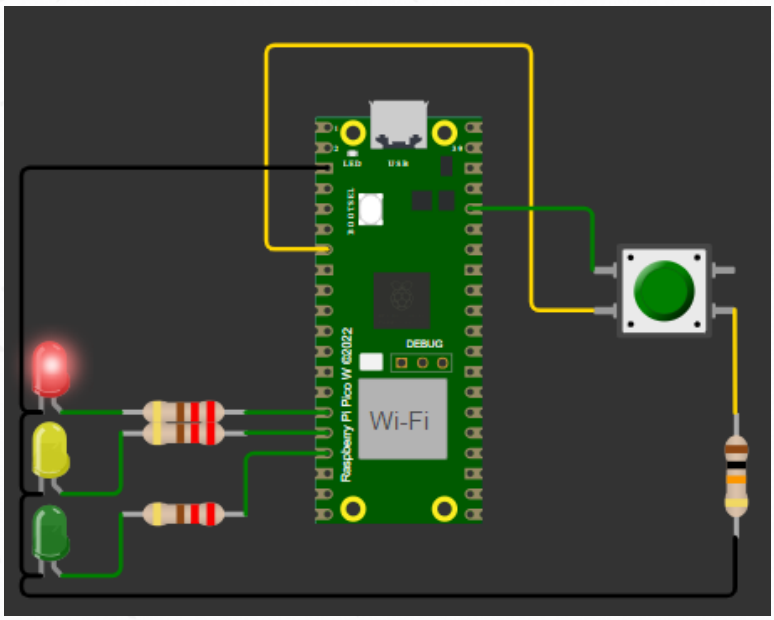


**Fig. 2: Diagrama do circuito do semáforo temporizado.**
Fonte: imagem do autor.

A imagem mostra um circuito eletrônico com um Raspberry Pi Pico W, três LEDs, um botão e dois resistores. O Raspberry Pi Pico W é um pequeno computador de placa única que se encontra no centro da imagem. A placa é verde e tem um logotipo que diz "Raspberry Pi Pico W 6522" escrito em preto. Abaixo do logotipo, há a inscrição "Wi-Fi" em amarelo. Acima do logotipo, há um chip preto com o logotipo da empresa.
Do lado direito da placa, há um botão verde que está conectado a um fio amarelo e um fio verde. O fio amarelo está conectado ao pino 33 do Raspberry Pi Pico W. O fio verde está conectado a um resistor marrom com faixa dourada que está conectado ao pino 32.
Do lado esquerdo da placa, há três LEDs: um vermelho, um amarelo e um verde. Cada LED está conectado ao Raspberry Pi Pico W por meio de um fio vermelho e um fio marrom. O LED vermelho está conectado ao pino 14, o LED amarelo ao pino 15 e o LED verde ao pino 16.
Cada um dos LEDs tem um resistor marrom com faixa dourada conectado a seu fio marrom. Os resistores estão conectados aos pinos 13, 12 e 11, respectivamente.

| | Componentes | GPIO |
|---|---|---|
| 1 | Semáforo | Vermelho - GPIO11<br>Amarelo - GPIO12<br>Verde - GPIO13 |
| 2 | Botões | GPIO5 |

**Fig. 3: lista de componentes e seus respectivos pinos GPIO**
Fonte: Tabela elaborada pelo autor.

A tabela apresenta uma lista de componentes e seus respectivos pinos GPIO correspondentes. A tabela está organizada em três colunas:
A primeira coluna é um número de identificação, a segunda coluna indica o componente e a terceira coluna indica o pino GPIO correspondente.
Aqui estão os elementos da tabela:
Número Componente GPIO
1 Semáforo Vermelho - GPIO11<br>Amarelo - GPIO12<br>Verde - GPIO13
2 Botão GPIO5
Em resumo, a tabela informa que o semáforo está conectado aos pinos GPIO11, GPIO12 e GPIO13, representando as cores vermelho, amarelo e verde, respectivamente. O botão está conectado ao pino GPIO5.

## Código Exemplo

Segue abaixo, o código de implementação do controle do semáforo temporizado

com o push-button. O código utiliza a biblioteca padrão do Raspberry Pi Pico para configurar os pinos GPIO e controlar o ciclo dos LEDs.

## Figura 2: Código de exemplo

```c
#include "pico/stdlib.h"

#define LED_VERDE_PIN 13
#define LED_AMARELO_PIN 12
#define LED_VERMELHO_PIN 11
#define BUTTON_PIN 5

bool ciclo_ativo = false;

void iniciar_ciclo() {
    while (ciclo_ativo) {
        // LED Verde aceso por 5 segundos
        gpio_put(LED_VERDE_PIN, 1);
        sleep_ms(5000);
        gpio_put(LED_VERDE_PIN, 0);

        // LED Amarelo aceso por 2 segundos
        gpio_put(LED_AMARELO_PIN, 1);
        sleep_ms(2000);
        gpio_put(LED_AMARELO_PIN, 0);

        // LED Vermelho aceso por 5 segundos
        gpio_put(LED_VERMELHO_PIN, 1);
        sleep_ms(5000);
        gpio_put(LED_VERMELHO_PIN, 0);
    }
}
```

**Fig. 4: código de exemplo.**
Fonte: imagem do autor.

A imagem contém um trecho de código em C que é utilizado para controlar um ciclo de LEDs em um microcontrolador, provavelmente na plataforma Raspberry Pi Pico, dado o cabeçalho `#include "pico/stdlib.h"`. Aqui está a descrição dos principais elementos do código:
1. Inclusão de biblioteca: O código inclui a biblioteca padrão da Raspberry Pi Pico, que contém funções necessárias para trabalhar com a GPIO (General Purpose Input/Output).
2. Definição de constantes:
- `#define LED_VERDE_PIN 13`: Define o pino 13 como o pino para o LED verde.
- `#define LED_AMARELO_PIN 12`: Define o pino 12 como o pino para o LED amarelo.
- `#define LED_VERMELHO_PIN 11`: Define o pino 11 como o pino para o LED vermelho.
- `#define BUTTON_PIN 5`: Define o pino 5 como o pino para um botão.
3. Declaração de variável:
- `bool ciclo_ativo = false;`: Declara uma variável booleana chamada `ciclo_ativo`, inicializada como falsa. Essa variável controla se o ciclo de LEDs deve ser executado.
4. Função para iniciar o ciclo:
- `void iniciar_ciclo()`: Declara uma função que não retorna valor e inicia o ciclo de LEDs.
- O corpo da função contém um loop `while (ciclo_ativo)`, que continuará a executar enquanto a variável `ciclo_ativo` for verdadeira.
5. Sequência de LEDs:
- LED Verde: O LED verde é aceso (`gpio_put(LED_VERDE_PIN, 1)`) por 5 segundos, seguido de um atraso de 500 milissegundos (`sleep_ms(500)`), e então é apagado (`gpio_put(LED_VERDE_PIN, 0)`).
- LED Amarelo: O LED amarelo é aceso por 2 segundos (`gpio_put(LED_AMARELO_PIN, 1)`), seguido por um atraso de 200 milissegundos (`sleep_ms(200)`), e depois apagado (`gpio_put(LED_AMARELO_PIN, 0)`).
- LED Vermelho: O LED vermelho é aceso por 5 segundos (`gpio_put(LED_VERMELHO_PIN, 1)`), e depois apagado após um atraso de 5000 milissegundos (`sleep_ms(5000)`).
Este código implementa uma sequência simples em que os LEDs verde, amarelo e vermelho são acionados em uma ordem específica, com tempos de ativação e desativação definidos. A execução do ciclo depende da variável `ciclo_ativo`, que deve ser definida como verdadeira para que o ciclo inicie

```c
int main() {
    stdio_init_all();

    // Inicializa e configura os pinos dos LEDs como saída
    gpio_init(LED_VERDE_PIN);
    gpio_set_dir(LED_VERDE_PIN, GPIO_OUT);

    gpio_init(LED_AMARELO_PIN);
    gpio_set_dir(LED_AMARELO_PIN, GPIO_OUT);

    gpio_init(LED_VERMELHO_PIN);
    gpio_set_dir(LED_VERMELHO_PIN, GPIO_OUT);

    // Inicializa e configura o pino do push-button como entrada com pull-down
    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_down(BUTTON_PIN);

    while (true) {
        // Verifica se o botão foi pressionado
        if (gpio_get(BUTTON_PIN)) {
            // Alterna o estado do ciclo do semáforo
            ciclo_ativo = !ciclo_ativo;

            // Se o ciclo estiver ativo, inicia o ciclo do semáforo
            if (ciclo_ativo) {
                iniciar_ciclo();
            }

            // Adiciona um pequeno delay para evitar múltiplas leituras do botão
            sleep_ms(200);
        }
    }
    return 0;
}
```

**Fig. 5: código de exemplo.**
Fonte: imagem do autor.

A imagem mostra mais um trecho de código em C, que contém a função principal (`main()`) para controlar o comportamento de LEDs e um botão (push-button) utilizando GPIOs em um microcontrolador. Aqui está a descrição detalhada do código:
1. Início da função `main()`:
- O código começa com a inicialização da entrada e saída padrão usando `stdio_init_all()`.
2. Configuração dos pinos dos LEDs:
- Para cada LED (verde, amarelo e vermelho), o pino correspondente é inicializado com a função `gpio_init()`.
- O modo dos pinos dos LEDs é configurado como saída com a função `gpio_set_dir(pino, GPIO_OUT)`. Os pinos dos LEDs configurados são:
- `LED_VERDE_PIN` (pino 13)
- `LED_AMARELO_PIN` (pino 12)
- `LED_VERMELHO_PIN` (pino 11)
3. Configuração do pino do botão:
- O pino do botão é inicializado usando `gpio_init(BUTTON_PIN)`.
- O modo do pino do botão é configurado como entrada com `gpio_set_dir(BUTTON_PIN, GPIO_IN)`.
- O pull-down interno do pino é habilitado com `gpio_pull_down(BUTTON_PIN)`, o que significa que, quando o botão não estiver sendo pressionado, o pino será mantido em nível baixo (0).
4. Loop principal (`while(true)`):
- Dentro do loop principal, o código verifica se o botão foi pressionado usando `gpio_get(BUTTON_PIN)`. Se o botão for pressionado:
- O estado do ciclo de LEDs é alternado: se o ciclo estava ativo, ele é desativado, e vice-versa (`ciclo_ativo = !ciclo_ativo`).
- Se o ciclo estiver ativo (`if (ciclo_ativo)`), a função `iniciar_ciclo()` é chamada para iniciar a sequência de LEDs.
- Há um pequeno atraso de 200 milissegundos (`sleep_ms(200)`) para evitar que múltiplas leituras do botão sejam registradas rapidamente.
5. Finalização:
- O código termina com `return 0;`, embora, como o loop `while` é infinito, essa linha não seja executada.
Este código é responsável por controlar um ciclo de LEDs (verde, amarelo e vermelho) com base na interação com um botão. Quando o botão é pressionado, o ciclo de LEDs é ativado ou desativado, e o estado dos LEDs é alternado em uma sequência determinada pela função `iniciar_ciclo()`.

**Análise do Código**

O código apresentado acima implementa a **lógica de controle** do semáforo utilizando funções da biblioteca pico/stdlib.h para manipular os pinos GPIO. Na função main(), os LEDs são configurados como **saídas digitais** e o **push-button** como uma entrada com resistor de pull-down, garantindo que a leitura seja estável quando o botão não estiver pressionado.

A função iniciar_ciclo() contém o ciclo de funcionamento do semáforo, no qual cada LED é aceso por um tempo específico: o LED verde por 5 segundos, seguido do LED amarelo por 2 segundos e, finalmente, o LED vermelho por 5 segundos. O estado ciclo_ativo controla se o ciclo do semáforo está em execução. Quando o push-button é pressionado, o valor de ciclo_ativo é alternado entre verdadeiro e falso, permitindo iniciar ou pausar o ciclo.

O funcionamento do semáforo é controlado pelo estado do push-button. Quando pressionado, o botão inicia o ciclo de temporização dos LEDs, alternando entre verde, amarelo e vermelho com os respectivos tempos definidos. A cada novo ciclo, o LED correspondente acende e apaga conforme a sequência pré-determinada. Se o botão for pressionado novamente, o ciclo é interrompido e o semáforo permanece no estado atual até que o botão seja pressionado novamente para retomar. A partir desse ponto, você pode experimentar diferentes combinações de LEDs e adicionar mais botões para ampliar as opções de controle, criando sequências de iluminação e interações mais complexas.

No próximo subtópico, exploraremos a utilização de um display de 7 segmentos para criar um temporizador que exibe números de 0 a 9. Nesse exemplo, vamos aplicar técnicas de temporização para controlar a mudança dos dígitos de forma precisa, oferecendo uma base prática para a criação de contadores e indicadores visuais em projetos de sistemas embarcados.

## 3.2 Implementar o controle de display de 7 segmentos com temporização por rotina de atraso

Neste segundo exemplo prático, vamos explorar como utilizar os GPIOs

do Raspberry Pi Pico para controlar um **display de 7 segmentos[6]**. Este exercício tem como objetivo desenvolver um temporizador simples que exibe os números de 0 a 9, utilizando rotinas de atraso para gerenciar a mudança de dígitos. A prática com displays de 7 segmentos é essencial para a criação de contadores, temporizadores e outros sistemas que exigem uma interface visual para exibir informações numéricas de maneira clara e precisa.

**Objetivo do Exemplo**

O objetivo deste exemplo é implementar um **contador de 0 a 9** utilizando um **display de 7 segmentos** controlado pelo Raspberry Pi Pico. A cada segundo, o número exibido no display será incrementado, e ao atingir 9, o contador reiniciará em 0. Este exemplo ajudará a consolidar os conceitos de temporização e controle de GPIOs, além de mostrar como manipular um display de 7 segmentos para criar interfaces visuais simples, mas eficazes, em sistemas embarcados.

**Materiais Necessários**

- 1 x Raspberry Pi Pico  no wokwi[7]
- 1 x Display de 7 segmentos (catodo comum)[6]
- 8 x Resistores de 220 ohms
- 1 x Protoboard

**Montagem do Circuito**

1. **Conexão dos Segmentos do Display:**
- Conecte os pinos dos segmentos do display (a, b, c, d, e, f, g) aos pinos GPIO correspondentes do Raspberry Pi Pico.
- Utilize resistores de 330 ohms em série para proteger os LEDs do display.

2. **Conexão do Ponto Decimal:**
- Se necessário, o pino do ponto decimal (dp) pode ser conectado a um GPIO adicional ou deixado desconectado se não for utilizado.

A seguir, temos o diagrama esquemático do circuito:

Figura 3 - Diagrama de conexão do display de 7 segmentos ao Raspberry Pi Pico.

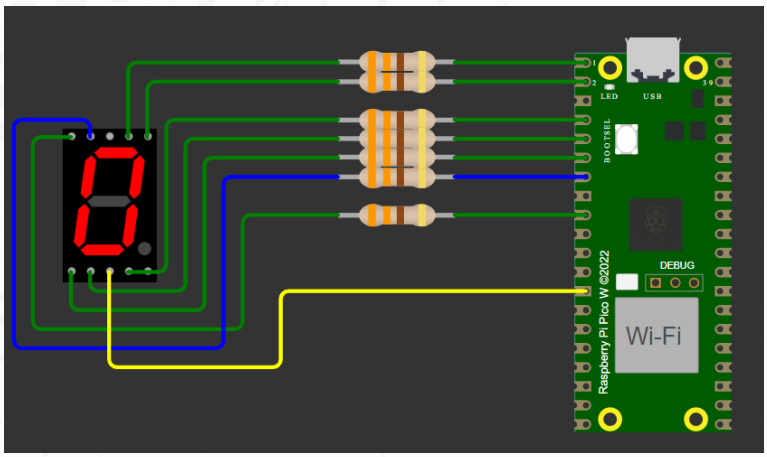


**Fig. 6: Diagrama de conexão do display de 7 segmentos ao Raspberry Pi Pico.**
Fonte: Tabela elaborada pelo autor.

Esta é uma imagem de um diagrama de circuito mostrando uma tela de 7 segmentos conectada a um Raspberry Pi. A tela de 7 segmentos está na esquerda e o Raspberry Pi está na direita. A tela de 7 segmentos é um tipo comum de exibição que é usado para exibir números. O Raspberry Pi é um pequeno computador que pode ser usado para uma variedade de fins, como aprender programação, automatizar tarefas e controlar projetos de eletrônicos.
A imagem mostra as conexões entre o Raspberry Pi e a tela de 7 segmentos. Existem vários fios conectando os dois dispositivos. Os fios são coloridos para indicar seus propósitos: verde, amarelo e azul. A tela de 7 segmentos está mostrando o número 0.
O diagrama de circuito também inclui resistores que servem para limitar a corrente que flui para a tela de 7 segmentos. Os resistores são importantes porque a tela de 7 segmentos precisa de uma certa quantidade de corrente para funcionar corretamente. Se muito pouca corrente fluir, a tela de 7 segmentos não acenderá. Se muita corrente fluir, a tela de 7 segmentos pode ser danificada.

| | Componentes | GPIO |
|---|---|---|
| 1 | Display 7 | a – GPIO 0<br>b – GPIO 1<br>c – GPIO 2<br>d – GPIO 3<br>e – GPIO 4<br>f – GPIO 5<br>g – GPIO 6<br>Com – GND4 |

**Fig .7.**
Fonte: Tabela elaborada pelo autor.

Esta é uma tabela com duas colunas. A primeira coluna é intitulada "Componentes" e a segunda coluna é intitulada "GPIO". Na primeira linha da tabela, a coluna "Componentes" lista "Display 7 segmentos" e a coluna "GPIO" lista "a - GPIO 0, b - GPIO 1, c - GPIO 2, d - GPIO 3, e - GPIO 4, f - GPIO 5, g - GPIO 6, Com - GND4"

**Código Exemplo: https://wokwi.com/projects/410112644471688193**

Abaixo está o código em C que implementa o controle do display de 7 segmentos, incrementando o número exibido a cada segundo. O código utiliza a biblioteca padrão do Raspberry Pi Pico para configurar os pinos GPIO e controlar cada segmento do display.

```c
#include "pico/stdlib.h"

// Definição dos pinos dos segmentos do display
const uint8_t segment_pins[] = {0, 1, 2, 3, 4, 5, 6};

// Mapeamento dos dígitos para os segmentos do display
const uint8_t digits[10][7] = {
    {1, 1, 1, 1, 1, 1, 0}, // 0
    {0, 1, 1, 0, 0, 0, 0}, // 1
    {1, 1, 0, 1, 1, 0, 1}, // 2
    {1, 1, 1, 1, 0, 0, 1}, // 3
    {0, 1, 1, 0, 0, 1, 1}, // 4
    {1, 0, 1, 1, 0, 1, 1}, // 5
    {1, 0, 1, 1, 1, 1, 1}, // 6
    {1, 1, 1, 0, 0, 0, 0}, // 7
    {1, 1, 1, 1, 1, 1, 1}, // 8
    {1, 1, 1, 0, 0, 1, 1}  // 9
};

// Função para exibir um dígito no display
void display_digit(uint8_t digit) {
    for (int i = 0; i < 7; i++) {
        gpio_put(segment_pins[i], digits[digit][i]);
    }
}

int main() {
    stdio_init_all();

    // Inicializa e configura os pinos dos segmentos como saída
    for (int i = 0; i < 7; i++) {
        gpio_init(segment_pins[i]);
        gpio_set_dir(segment_pins[i], GPIO_OUT);
    }
```

**Fig. 8: Exemplo de código.**
Fonte: imagem do autor.

A imagem mostra um código escrito em linguagem C para um microcontrolador (provavelmente um Raspberry Pi Pico), utilizado para controlar um display de 7 segmentos. Vou descrever os trechos principais do código:
1. Inclusão de biblioteca: A primeira linha inclui a biblioteca `pico/stdlib.h`, essencial para a funcionalidade do Raspberry Pi Pico.
2. Definição dos pinos:
- O array `segment_pins[]` define quais pinos GPIO do microcontrolador estão conectados a cada um dos 7 segmentos do display. São sete pinos numerados de 0 a 6.
3.Mapeamento dos dígitos:
- O array bidimensional `digits[10][7]` define como acionar os segmentos para formar os dígitos de 0 a 9. Cada linha contém uma sequência de 0s e 1s, onde 1 significa que o segmento correspondente deve acender, e 0 significa que ele deve permanecer apagado.
4.Função `display_digit(uint8_t digit)`:
- Esta função exibe um dígito no display. Ela recebe como parâmetro o dígito (de 0 a 9) e ativa ou desativa os segmentos corretos de acordo com o mapeamento.
5. Função principal `main()`:
- Inicializa o sistema com `stdio_init_all()` e configura os pinos GPIO como saídas. Cada pino do array `segment_pins` é inicializado e configurado como saída através de um loop.
Este código é responsável por exibir números de 0 a 9 em um display de 7 segmentos, controlando diretamente os pinos GPIO do microcontrolador.

```c
    uint8_t contador = 0;
    while (true) {
        display_digit(contador);  // Exibe o dígito atual no display
        sleep_ms(1000);  // Espera 1 segundo
        contador++;  // Incrementa o contador
        if (contador > 9) {
            contador = 0;  // Reinicia o contador após 9
        }
    }
    return 0;
}
```

**Fig. 9: exemplo de código.**
Fonte: imagem do autor.

A imagem exibe uma continuação do código anterior, também escrito em C, utilizado para controlar um display de 7 segmentos. Aqui está a descrição das linhas de código:
1. Inicialização do contador:
- A variável `contador` do tipo `uint8_t` (um inteiro de 8 bits sem sinal) é inicializada com o valor `0`.
2. Loop infinito `while(true)`:
- Este loop faz o programa rodar indefinidamente.
3. Função `display_digit(contador)`:
- A função `display_digit()` exibe o valor atual do contador no display de 7 segmentos.
4. Função `sleep_ms(1000)`:
- O programa pausa por 1000 milissegundos (1 segundo), fazendo com que o número exibido permaneça visível por esse tempo.
5. Incremento do contador:
- O contador é incrementado em 1 a cada ciclo do loop.
6. Verificação e reinicialização do contador:
- Se o valor do contador ultrapassar 9, ele é reiniciado para 0. Isso faz com que os dígitos no display sejam exibidos em um ciclo contínuo de 0 a 9.
7. Retorno:
- Embora a função `return 0;` esteja presente no final do código, ela nunca é executada, pois o loop `while(true)` impede que o programa saia.
Este código faz com que o display de 7 segmentos exiba números de 0 a 9 em sequência, mudando a cada segundo e repetindo o ciclo indefinidamente.

## Análise do Código

O código apresentado acima utiliza a função **display_digit()** para controlar a exibição dos números no display de 7 segmentos. A **matriz digits** armazena o mapeamento dos segmentos para cada dígito de 0 a 9. Na função **main()**, os pinos dos segmentos são configurados como saídas digitais, e um **loop infinito** é utilizado para atualizar o display a cada segundo.

A cada iteração, o dígito exibido é incrementado, e a função **sleep_ms(1000)** adiciona um atraso de 1 segundo entre as atualizações. Quando o contador atinge o **valor 10**, ele é reiniciado para 0, criando um **ciclo contínuo de contagem de 0 a 9.**

A partir de agora, você pode experimentar **modificando** o tempo de **atraso** para alterar a **velocidade da contagem**, ou adicionar mais displays para criar um contador de dois dígitos. No próximo subtópico, exploraremos a **multiplexação** de displays de 7 segmentos, que nos permitirá controlar múltiplos displays utilizando um número reduzido de pinos GPIO, tornando possível a exibição de **contagens mais complexas** e a criação de **interfaces mais elaboradas.**

## 3.3. Implementar uma rotina de controle de multiplexação de displays de 7 segmentos

Neste terceiro exemplo prático, vamos explorar como utilizar a técnica de **multiplexação**[8] para **controlar** dois displays de 7 segmentos com o Raspberry Pi Pico. A multiplexação permite alternar rapidamente entre os displays, criando a **ilusão** de que ambos estão **acesos simultaneamente**, enquanto utilizamos um **número reduzido** de pinos GPIO. Esse exercício é fundamental para o desenvolvimento de sistemas que exigem a exibição de números ou informações em múltiplos displays, otimizando o uso dos recursos do microcontrolador.

### Objetivo do Exemplo

O objetivo deste exemplo é implementar um **contador de dois dígitos** utilizando **multiplexação** para controlar **dois displays de 7 segmentos**. A cada segundo, o número exibido nos displays será incrementado, contando de 00 a 99, e então reiniciará para 00. Essa prática permitirá **consolidar** os conceitos de multiplexação e **controle** de GPIOs, técnicas fundamentais para desenvolver sistemas eficientes que exigem a utilização de múltiplos displays ou indicadores visuais, utilizando um número reduzido de pinos. Isso possibilita a **criação** de interfaces visuais **complexas e otimizadas**, mesmo em microcontroladores com **recursos limitados**.

### Materiais Necessários

- 1 x Raspberry Pi Pico no wokwi
- 2 x Displays de 7 segmentos (catodo comum)
- 16 x Resistores de 220 ohms
- 1 x Protoboard

## Montagem do Circuito

1. **Conexão dos Segmentos dos Displays:**
- Conecte os pinos dos segmentos (a, b, c, d, e, f, g) dos dois displays aos mesmos pinos GPIO do Raspberry Pi Pico, usando resistores de 330 ohms em série para proteger os LEDs dos displays.

2. **Conexão dos Pinos de Controle dos Displays:**
- Utilize dois GPIOs adicionais (GPIO 11 e GPIO 12, por exemplo) para controlar a ativação dos displays. Conecte os pinos comuns de cada display a esses GPIOs, permitindo ligar e desligar cada display individualmente.

A seguir, temos o diagrama esquemático do circuito:

Figura 5- Diagrama de conexão dos displays de 7 segmentos ao Raspberry Pi Pico.

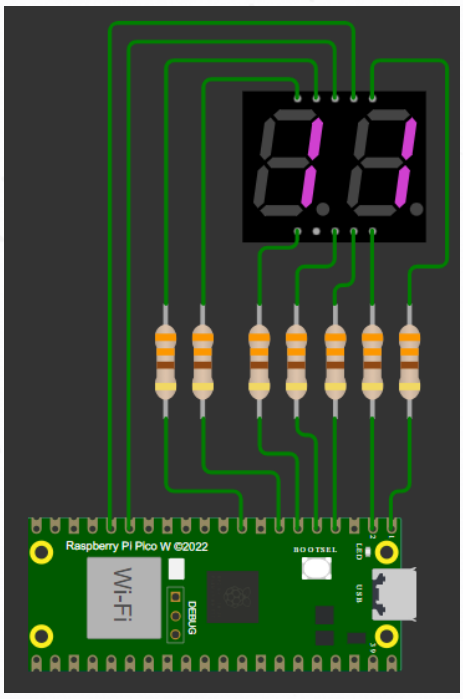


**Fig. 10: Diagrama de conexão dos displays de 7 segmentos ao Raspberry Pi Pico.**
Fonte: imagem do autor.

A imagem mostra um diagrama de um circuito eletrônico, com um Raspberry Pi Pico W conectado a um display de sete segmentos. O display de sete segmentos está na parte superior da imagem, exibindo o número 8.8 em um tom roxo brilhante. O Raspberry Pi Pico W está na parte inferior da imagem, com o seu conector USB visível à direita.
O circuito é composto por várias conexões representadas por fios verdes. Os fios conectam os pinos do Raspberry Pi Pico W aos pinos do display de sete segmentos, passando por resistores. Há sete resistores no total, simbolizados por linhas marrom-laranja-marrom.
O diagrama ilustra como o Raspberry Pi Pico W pode ser usado para controlar um display de sete segmentos, permitindo que ele exiba números e caracteres. Essa configuração é frequentemente usada em projetos de eletrônica para exibir dados ou informações

| | Componentes | GPIO |
|---|---|---|
| 1 | Display 7 segmentos | a – GPIO 5<br>b – GPIO 6<br>c – GPIO 4<br>d – GPIO 2<br>e – GPIO 3<br>f – GPIO 0<br>g – GPIO 1<br>Dig1 – GP11<br>Dig2 - GP12 |

**Fig .11.**
Fonte: Tabela elaborada pelo autor.

Esta é uma tabela com duas colunas. A primeira coluna é intitulada "Componentes" e a segunda coluna é intitulada "GPIO". Na primeira linha da tabela, a coluna "Componentes" lista "Display 7 segmentos" e a coluna "GPIO" lista "a - GPIO 5, b - GPIO 6, c - GPIO 4, d - GPIO 2, e - GPIO 3, f - GPIO 0, g - GPIO 1, Dig1 - GP11, Dig2 - GP12".

## Código Exemplo

Abaixo está o **código em C** que implementa o controle de **dois displays de 7 segmentos** utilizando **multiplexação** (Dig1 e Dig2). O código alterna **rapidamente** entre os displays para exibir um número de **dois dígitos**, incrementando o valor a cada segundo.

```c
#include "pico/stdlib.h"

// Definição dos pinos dos segmentos do display e dos controles dos displays (Dig1 e Dig2)
const uint8_t segment_pins[] = {5, 6, 4, 2, 3, 0, 1}; // Pinos dos segmentos a, b, c, d, e, f, g
const uint8_t display_pins[] = {12, 11}; // Pinos de controle dos displays (Dig1 e Dig2)

// Mapeamento dos dígitos para os segmentos do display (0 = desligado, 1 = ligado)
const uint8_t digits[10][7] = {
    {1, 1, 1, 1, 1, 1, 0}, // 0
    {0, 1, 1, 0, 0, 0, 0}, // 1
    {1, 1, 0, 1, 1, 0, 1}, // 2
    {1, 1, 1, 1, 0, 0, 1}, // 3
    {0, 1, 1, 0, 0, 1, 1}, // 4
    {1, 0, 1, 1, 0, 1, 1}, // 5
    {1, 0, 1, 1, 1, 1, 1}, // 6
    {1, 1, 1, 0, 0, 0, 0}, // 7
    {1, 1, 1, 1, 1, 1, 1}, // 8
    {1, 1, 1, 0, 0, 1, 1}  // 9
};

// Função para configurar todos os pinos como saída
void setup() {
    for (int i = 0; i < 7; i++) {
        gpio_init(segment_pins[i]);
        gpio_set_dir(segment_pins[i], GPIO_OUT);
    }
    for (int i = 0; i < 2; i++) {
        gpio_init(display_pins[i]);
        gpio_set_dir(display_pins[i], GPIO_OUT);
        gpio_put(display_pins[i], 0); // Desativa inicialmente os displays
    }
}
```

**Fig. 12: exemplo de código.**
Fonte: imagem do autor.

A imagem mostra um código fonte escrito em C para um microcontrolador, provavelmente para um Raspberry Pi Pico. O código define a configuração de um circuito que inclui dois displays de sete segmentos e alguns pinos GPIO.
O código inicia incluindo a biblioteca "pico/stdlib.h", essencial para o funcionamento do Raspberry Pi Pico. Em seguida, são definidas duas variáveis do tipo "const uint8_t" para armazenar os pinos conectados aos displays.
A variável "segment_pins" armazena os pinos conectados aos segmentos do display de 7 segmentos (a, b, c, d, e, f, g). A variável "display_pins" armazena os pinos conectados aos controles dos displays, responsáveis por ativar ou desativar cada display.
Após a definição das variáveis de pinos, o código define uma matriz "const uint8_t digits" que mapeia cada dígito de 0 a 9 para uma combinação de bits que representa o estado de cada segmento do display. Um valor 1 significa que o segmento está ligado e 0 significa que está desligado.
A seguir, o código define uma função "void setup()" que inicializa os pinos do display de sete segmentos como saídas e configura os displays para estarem inicialmente desligados. Essa função é provavelmente chamada no início do programa para preparar o circuito.
O código é organizado em blocos com comentários em português, explicando o propósito de cada linha. As variáveis e funções são nomeadas de forma clara, o que facilita a compreensão do código.
Em resumo, o código configura um circuito com dois displays de sete segmentos conectados a um microcontrolador. As variáveis definidas no código permitem controlar os segmentos dos displays para exibir dígitos, e a função "setup" inicializa o circuito e define os displays como desligados.

```c
// Função para exibir um dígito em um display específico
void show_digit(uint8_t display, uint8_t digit) {
    // Desativa ambos displays
    gpio_put(display_pins[0], 0);
    gpio_put(display_pins[1], 0);

    // Ativa os segmentos correspondentes ao dígito
    for (int i = 0; i < 7; i++) {
        gpio_put(segment_pins[i], digits[digit][i]);
    }

    // Ativa o display desejado
    gpio_put(display_pins[display], 1);
}
```

**Fig. 13: exemplo de código.**
Fonte: imagem do autor.

A imagem mostra um trecho de código fonte escrito na linguagem C, provavelmente para controlar um circuito que inclui dois displays de 7 segmentos. O código define uma função chamada "show_digit" que recebe dois argumentos: "display" e "digit". O argumento "display" indica qual dos dois displays deve ser usado (0 ou 1), e o argumento "digit" indica qual dígito deve ser exibido no display (de 0 a 9).
A função "show_digit" começa desativando ambos os displays definindo os pinos "display_pins[0]" e "display_pins[1]" como 0.
Em seguida, a função ativa os segmentos correspondentes ao dígito desejado. Para isso, utiliza um loop que itera pelos segmentos do display (a, b, c, d, e, f, g) e configura o pino correspondente de acordo com o valor armazenado na matriz "digits".
Por fim, a função ativa o display desejado definindo o pino "display_pins[display]" como 1.
A função "show_digit" permite que o código controle individualmente cada display e exiba o dígito desejado em cada um

```c
int main() {
    stdio_init_all();
    setup(); // Configura todos os pinos como saída

    while (true) {
        for (uint8_t numero = 0; numero < 100; numero++) { // Loop para contagem de 0 a 99
            uint8_t dezenas = numero / 10;  // Calcula o dígito das dezenas
            uint8_t unidades = numero % 10; // Calcula o dígito das unidades

            // Multiplexa rapidamente entre os displays
            for (int j = 0; j < 50; j++) { // Ajuste o valor de j para controlar a velocidade de atualização
                show_digit(0, dezenas);  // Exibe o dígito das dezenas no primeiro display
                sleep_ms(5);  // Tempo de permanência no primeiro display
                show_digit(1, unidades); // Exibe o dígito das unidades no segundo display
                sleep_ms(5);  // Tempo de permanência no segundo display
            }

            sleep_ms(500); // Pausa de meio segundo entre incrementos
        }
    }
    return 0;
}
```

**Fig. 14: exemplo de código.**
Fonte: imagem do autor.

A imagem mostra um código fonte escrito na linguagem C para um microcontrolador, provavelmente para um Raspberry Pi Pico. O código implementa uma contagem de 0 a 99 usando dois displays de 7 segmentos.
O código começa definindo a função principal "int main()", que contém o código principal do programa. Dentro da função "main", o código chama a função "stdio_init_all()" para inicializar a biblioteca padrão de entrada e saída, e a função "setup()" para configurar os pinos do microcontrolador como saídas.
Em seguida, o código entra em um loop infinito "while (true)". O loop usa um for para contar de 0 a 99. A cada iteração do loop, o código calcula o dígito das dezenas e o dígito das unidades do número atual, armazenando-os nas variáveis "dezenas" e "unidades", respectivamente.
Dentro do loop "while", o código usa outro loop "for" para multiplexar os displays. A cada iteração deste loop, o código chama a função "show_digit" para exibir o dígito das dezenas no primeiro display e o dígito das unidades no segundo display.
O código também inclui comandos "sleep_ms" para atrasar a execução, permitindo que os displays exibem os números por um tempo determinado.
Em resumo, o código controla dois displays de 7 segmentos para mostrar uma contagem de 0 a 99. O código usa uma técnica de multiplexação para alternar rapidamente entre os dois displays e criar a ilusão de que ambos estão sendo atualizados simultaneamente.
O código está bem organizado e comentado em português. O comentário "Ajuste o valor de j para controlar a velocidade de atualização" indica que o loop de multiplexação pode ser ajustado para controlar a velocidade da contagem.

**Análise do Código**

O código apresentado acima utiliza a função display_digit() para alternar rapidamente entre os dois displays, exibindo um dígito de cada vez em cada display. A matriz digits armazena o mapeamento dos segmentos para cada número de 0 a 9. Na função main(), os pinos dos segmentos e dos displays são configurados como saídas digitais, e o loop infinito é utilizado para atualizar os displays com base na contagem de 0 a 99. Vamos explorar o funcionamento de cada parte do código para entender como a multiplexação é utilizada para controlar múltiplos displays com um número limitado de pinos GPIO:

- **segment_pins[ ]:** Cada elemento do array corresponde a um pino do Raspberry Pi Pico[3] que está conectado aos segmentos a, b, c, d, e, f, g dos displays de 7 segmentos. Esses pinos controlam quais segmentos de cada display serão acesos para formar o dígito.

- **display_pins[ ]:** Define os pinos que controlam qual display está ativo. O primeiro display é controlado pelo GPIO 12 e o segundo pelo GPIO 11.

- **Mapeamento dos dígitos:** Cada linha do array digits corresponde a um número de 0 a 9, e cada coluna representa um segmento do display (a, b, c, d, e, f, g). O valor 1 significa que o segmento correspondente estará aceso, enquanto 0 indica que estará apagado. Por exemplo, para exibir o número 2, os segmentos a, b, d, e, g serão ativados (1), e os segmentos c e f serão desativados (0).

- **Função setup ( ):** Na inicialização, os pinos de segmento (segment_pins) e de controle de display (display_pins) são configurados como saídas. Na desativação inicial, todos os displays são desativados inicialmente (gpio_put(display_pins[i], 0)), garantindo que nenhum display esteja ativo ao iniciar.

- **Função show_digit( ):** A desativação dos displays ocorre antes de configurar os segmentos, pois ambos os displays são desativados para evitar exibição incorreta de dígitos. Na configuração dos segmentos, a função percorre cada segmento e configura o estado (ligado ou desligado) com base na matriz digits para o dígito desejado.

A técnica de **multiplexação** permite que o **mesmo conjunto** de pinos de segmentos seja **compartilhado** entre os displays, com apenas **dois pinos adicionais** utilizados para controlar qual display está ativo em um dado momento. A **rápida alternância** entre os displays cria a **ilusão** de que ambos estão **acesos simultaneamente**, sem causar interferências visuais.

Você pode experimentar adicionando mais displays e ajustando o tempo de multiplexação para melhorar a estabilidade visual dos dígitos exibidos.

No próximo subtópico, vamos estruturar um desafio mais avançado: o desenvolvimento de um **cofre digital com senha**, utilizando um **teclado matricial** para a entrada de dados e **temporização** para validação das entradas e controle de segurança. Esse exemplo permitirá explorar a **criação de sistemas interativos mais sofisticados**, aplicando tudo o que aprendemos até agora.

## 3.4. Projetar uma aplicação embarcada de controle de um cofre de 4 dígitos

Neste subtópico, o **objetivo** é guiá-lo na **estruturação** de um **projeto** completo de controle de um **cofre digital** utilizando um teclado matricial[5] e dois displays de 7 segmentos. A intenção é mostrar a **lógica** e o **raciocínio** necessários para desenvolver a aplicação sem fornecer o código completo, incentivando você a aplicar os conceitos aprendidos e a desenvolver suas próprias soluções. Essa prática será valiosa para consolidar o conhecimento adquirido e para que você possa enfrentar desafios reais em projetos de sistemas embarcados.

### Objetivo do Projeto

O objetivo deste projeto é **criar** um sistema embarcado que permita o **controle de um cofre digital** por meio de uma **senha** de 4 dígitos. A senha deve ser inserida utilizando um teclado matricial, e o estado do cofre (aberto ou fechado) será indicado por **dois LEDs: um LED verde**, que acenderá quando o cofre estiver aberto, e um **LED vermelho**, que piscará em caso de senha incorreta. Além disso, após **três tentativas incorretas consecutivas**, o sistema deve entrar em um **modo de bloqueio temporário**, desativando a entrada de novas tentativas por um **período determinado**.

## Planejamento e Estruturação do Projeto

a) Definição dos Requisitos Funcionais
Antes de iniciar a implementação, é essencial definir claramente o que o sistema deve fazer. Os requisitos funcionais básicos para este projeto são:

| | |
|---|---|
| **Leitura de Senha** | Capturar a senha de 4 dígitos utilizando um teclado matricial. |
| **Validação da Senha** | Comparar a senha inserida com a senha predefinida do cofre. |
| **Controle de Acesso** | Se a senha for correta, acionar o LED verde indicando que o cofre está aberto |
| | Se a senha for incorreta, piscar o LED vermelho para indicar erro. |
| **Bloqueio Temporário** | Após três tentativas incorretas consecutivas, entrar em um modo de bloqueio temporário, desativando o sistema por alguns segundos. |

b) Diagrama de Blocos do Sistema
Para facilitar a compreensão do funcionamento do sistema, crie um diagrama de blocos. O diagrama deve incluir os seguintes componentes:

| | |
|---|---|
| **Teclado Matricial** | Dispositivo de entrada para capturar a senha. |
| **Microcontrolador** | Responsável por processar a entrada do teclado, validar a senha e controlar os displays e LEDs. |
| **Displays de 7 Segmentos** | Utilizados para exibir o status do sistema e a senha digitada. |
| **LEDs** | LED verde para indicar que o cofre está aberto. |
| | LED vermelho para indicar tentativas incorretas. |
| **Sistema de Bloqueio** | Lógica para impedir novas tentativas após três erros consecutivos. |

c) Design do Circuito

Para implementar o sistema, você precisará conectar adequadamente todos os componentes ao microcontrolador. Aqui estão as principais conexões que devem ser feitas:

| | |
|---|---|
| **Conexão do Teclado Matricial** | Conectar as 4 linhas do teclado aos pinos GPIO do microcontrolador e as 4 colunas a outros pinos GPIO, utilizando resistores de pull-down para evitar leituras incorretas. |
| **Conexão dos Displays de 7 Segmentos** | Conectar os segmentos dos displays aos pinos GPIO específicos para exibir os dígitos correspondentes. |
| | Controlar quais displays estão ativos utilizando os pinos GPIO de controle (Dig1 e Dig2). |
| **Conexão dos LEDs** | Conectar o LED verde a um pino GPIO do microcontrolador através de um resistor de 330 ohms. |
| | Conectar o LED vermelho a outro pino GPIO com o resistor de 330 ohms. |

d) Algoritmo de Funcionamento

Descreva o algoritmo básico que o sistema deve seguir para atingir os objetivos propostos:

| | |
|---|---|
| **Inicialização** | Configurar todos os pinos como entrada ou saída, conforme necessário. |
| | Desativar inicialmente os displays e LEDs. |
| **Captura da Senha** | Esperar a entrada dos 4 dígitos do teclado matricial. |
| | Exibir cada dígito no display conforme ele é inserido. |
| **Validação da Senha** | Comparar a senha digitada com a senha predefinida. |
| | Se a senha for correta, acender o LED verde e manter o cofre aberto. |
| | Se a senha for incorreta, incrementar o contador de tentativas e piscar o LED vermelho. |
| **Controle de Bloqueio** | Se três tentativas incorretas forem feitas, entrar no modo de bloqueio temporário. |
| | Durante o bloqueio, desativar os displays e não aceitar novas entradas do teclado. |
| | Após o tempo de bloqueio, permitir novas tentativas de entrada de senha. |

e) Funções Auxiliares

Para simplificar a implementação, é importante dividir o código em funções menores que realizem tarefas específicas:

| | |
|---|---|
| **Função de Leitura do Teclado:** | Captura a tecla pressionada e retorna o número correspondente. |
| **Função de Exibição no Display** | Mostra os dígitos no display correspondente, utilizando multiplexação para alternar entre os displays. |
| **Função de Validação de Senha** | Compara a senha digitada com a senha correta. |
| **Função de Bloqueio Temporário** | Impede novas entradas por um determinado tempo após três tentativas incorretas. |

**Desafio para Implementação**

**NA PRÁTICA**

Com base nas orientações fornecidas, implemente a lógica para capturar a senha, validar a entrada e controlar os LEDs e displays conforme descrito. Não fornecemos o código completo, pois este exercício é um desafio prático para consolidar seu aprendizado. Lembre-se de aplicar os conceitos de temporização e controle de GPIOs aprendidos nos exemplos anteriores para completar o projeto.

1. Desafio Adicional:

- Modifique o sistema para permitir a configuração da senha através do próprio teclado matricial.
- Adicione um buzzer para emitir um alerta sonoro em caso de tentativas incorretas.

Ao finalizar o projeto, você terá uma aplicação funcional de controle de cofre digital, uma excelente demonstração prática dos conceitos de sistemas embarcados com microcontroladores.

## SINTESE

Nesta unidade, exploramos como utilizar **técnicas de temporização** (delay) em sistemas embarcados com o Raspberry Pi Pico ou com a plataforma educacional BitDogLab, aplicando-as em diversos exemplos práticos. Iniciamos com o controle de um semáforo temporizado, no qual aprendemos a usar delays para alternar entre **três LEDs** representando as fases do semáforo, e a utilizar um **push-button** para iniciar e pausar o ciclo de operação. Em seguida, passamos para o controle de um **display de 7 segmentos**, exibindo números de 0 a 9 com **intervalos de tempo definidos**, reforçando o conceito de mapeamento correto dos segmentos e o uso de rotinas de atraso para temporização.

A seguir, abordamos a **multiplexação** de **displays de 7 segmentos**, criando um contador de **dois dígitos** que **alterna rapidamente** entre os displays, **economizando pinos** de GPIO e exibindo números de 00 a 99. Com esse exemplo, consolidamos a compreensão da multiplexação e sua aplicação em projetos com múltiplos displays.

Por fim, estruturamos o **projeto** de um **cofre digital com senha**, utilizando um **teclado matricial** para a entrada de dados e LEDs para indicar o status do cofre. Este projeto destacou a importância do uso de **temporização** na validação de entradas e na gestão de tentativas incorretas.

De forma geral, desenvolvemos **habilidades essenciais** para o controle de GPIO em sistemas embarcados, aplicando temporização para coordenar dispositivos e criar sistemas interativos. A **prática** com esses exemplos proporcionou uma **base sólida** para **projetos mais complexos**, ampliando nossa capacidade de **criar soluções eficientes e robustas** em aplicações embarcadas.

Agora que você concluiu esta unidade sobre práticas de GPIO com delay, é hora de colocar todo o conhecimento adquirido em prática! Relembre cada exemplo, revise o código e experimente implementar suas próprias variações. Modifique os tempos de delay, adicione novos componentes e crie suas próprias aplicações utilizando os conceitos de temporização.

Não se esqueça de revisar todo o material e concluir as atividades práticas disponíveis na plataforma. Esses exercícios são essenciais para consolidar o seu aprendizado e garantir que você compreendeu todos os conceitos abordados.

Lembre-se, a prática constante é o caminho para a excelência. Desafie-se a desenvolver projetos novos e criativos. Que tal implementar um temporizador de contagem regressiva ou um sistema de controle de acesso mais avançado?

**Compartilhe suas criações e dúvidas com a comunidade EmbarcaTech! Participar ativamente das discussões e colaborar com outros alunos é uma excelente maneira de aprimorar suas habilidades e aprender novas soluções.**

Estamos ansiosos para ver o que você é capaz de construir.

## 6. CONCLUSÃO

Nesta unidade, exploramos a aplicação **prática de técnicas** de temporização com GPIOs, utilizando o Raspberry Pi Pico para desenvolver projetos que exigem **controle preciso de tempo**. Implementamos desde sistemas simples, como **o controle de semáforo com push-button**, até projetos mais complexos, como a **multiplexação** de displays de 7 segmentos e o **desenvolvimento** de um cofre digital com senha.
Ao dominar o uso de **delays**, você aprendeu a **coordenar ações** e **sincronizar dispositivos** de maneira eficiente, **habilidades fundamentais** para qualquer desenvolvedor de sistemas embarcados. A prática com temporização, aliada ao conhecimento de manipulação de GPIOs, abre portas para a criação de sistemas **interativos** e **inteligentes**, capazes de responder a estímulos externos com precisão.

Agora, você está preparado para enfrentar novos desafios, aplicando os conceitos aprendidos em projetos ainda mais avançados. Não se esqueça de continuar praticando, explorando novas possibilidades e buscando soluções criativas para os problemas encontrados. A programação em C, aliada à manipulação de hardware, permite uma infinidade de aplicações. Utilize este conhecimento para criar, inovar e transformar suas ideias em realidade.

Na próxima aula, vamos discutir interrupções, abordando sua configuração, rotinas de tratamento e atividades práticas. Com as interrupções, você aprenderá a reagir a eventos externos de forma instantânea, sem depender de verificações constantes no código. Isso permitirá a criação de sistemas mais eficientes e responsivos, expandindo ainda mais suas habilidades em sistemas embarcados.

O aprendizado contínuo é essencial. Continue se aprofundando nos temas abordados, participando de comunidades de desenvolvedores e experimentando novas técnicas e tecnologias. Estamos confiantes de que você está no caminho certo para se tornar um especialista em sistemas embarcados.

**Parabéns por completar mais esta etapa!**

## REFERÊNCIAS


[1] BITDOGLAB. BitDogLab. Disponível em: https://github.com/BitDogLab/BitDogLab. Acesso em: 26 set. 2024.
[2] RASPBERRY PI FOUNDATION. RP2040 Datasheet. Disponível em: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. Acesso em: 25 set. 2024.
[3] GAY, Warren. Debouncing. In: GAY, Warren. Exploring the Raspberry Pi 2 with C++. 1. ed. New York: Apress, 2015. p. 105-112.
[4] IGINO, Wellington Passos et al. Ensino de sistemas embarcados baseado em projeto: exemplo aplicado à robótica. 2023.
[5] CIRCUIT BASICS. How to set up a keypad on an Arduino. Disponível em: https://www.circuitbasics.com/how-to-set-up-a-keypad-on-an-arduino/. Acesso em: 26 set. 2024.
[6] BARROS, ANDRÉ; OLIVEIRA, MELLO. Fundamentos para o Ensino Técnico de MECATRÔNICA–TEORIA. 2017.
[7] WOKWI. Wokwi 7 Segment Display. Disponível em: https://docs.wokwi.com/pt-BR/parts/wokwi-7segment. Acesso em: 25 set. 2024.
[8] TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. Sistemas digitais. Pearson Educación, 2010.


embarca
tech
Residência
Tecnológica
em Sistemas
Embarcados