



Residência
Tecnológica
em Sistemas
Embarcados

Clock e Temporizadores

Unidade 4 | Capítulo 5

Executores:



Coordenação:



Iniciativa:



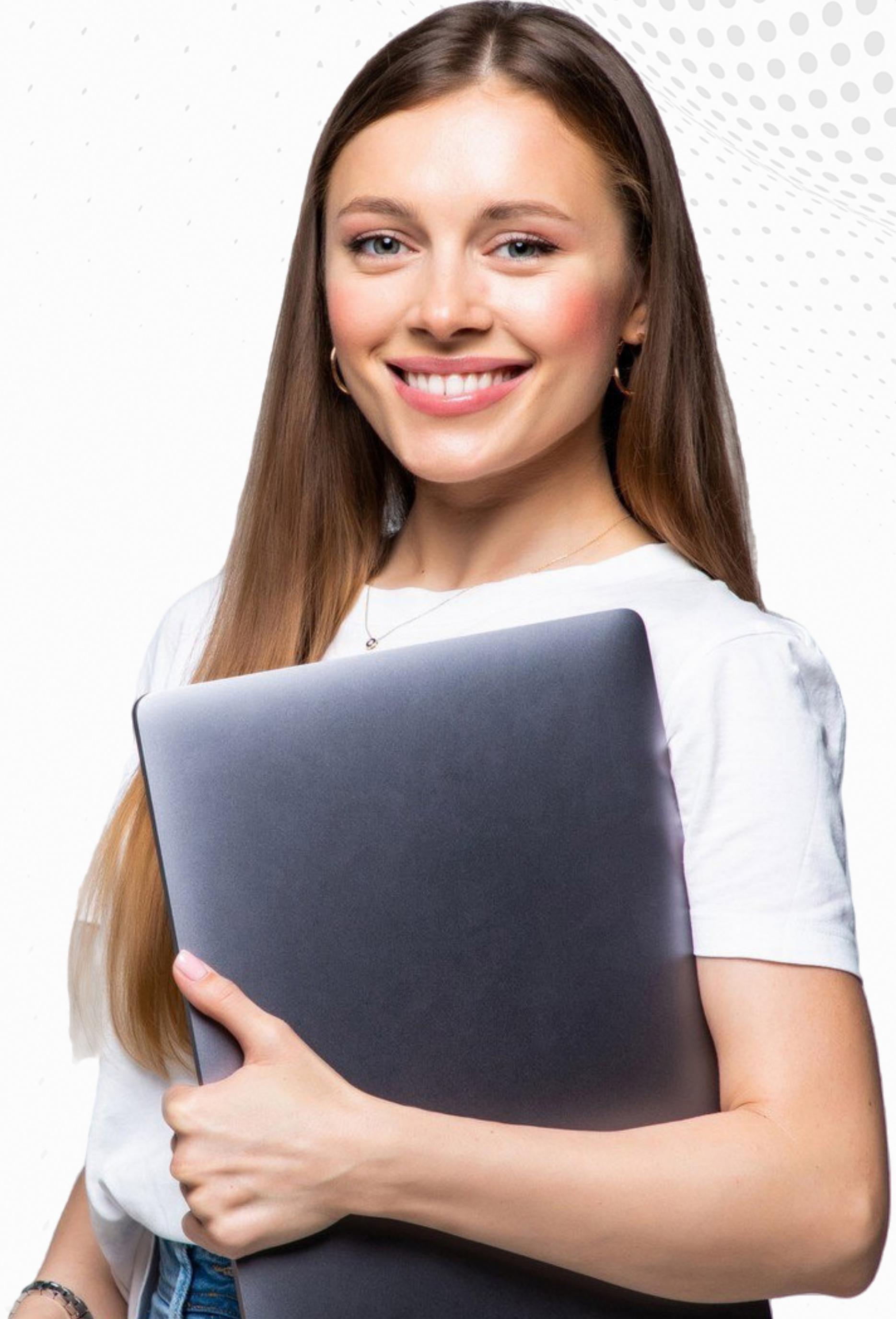
Sumário

- Objetivos
- Revisão
- Visão Geral
- Clockno RP2040
- Exemplos de Código
- Temporizadores no RP2040
- Exemplos de Código
- Principais Pontos
- Conclusão



Objetivos

- Compreender o funcionamento do Clocke Temporizadores
- Configurar o módulo de Clock
- Configurar os módulos de Temporizadores
- Aplicar os conhecimentos de interrupções e temporizadores



Revisão

Nas últimas aulas você aprendeu:

- Usar e configurar as interrupções no RP2040
- Controlar os pinos de entrada e saída (GPIO)
- Desenvolvimento de software em microcontroladores



Visão Geral

O que é o clock?

- O clock em um microcontrolador é um sinal elétrico periódico que serve como referência de tempo para a execução de operações internas. Ele sincroniza a sequência de instruções, permitindo que o microcontrolador realize tarefas de forma coordenada e precisa.
- O clock é gerado por um oscilador (interno ou externo) e sua frequência, medida em hertz (Hz), determina quantas operações o microcontrolador pode executar por segundo. Uma frequência mais alta significa maior velocidade de processamento, mas também pode resultar em maior consumo de energia. Assim, a escolha da frequência do clock é crucial para otimizar o desempenho e a eficiência do sistema.

Configurações do clock

- No microcontrolador RP2040, utilizado no Raspberry Pi Pico e no Raspberry Pi Pico W, o sistema de clock é altamente configurável. Ele utiliza um oscilador de cristal externo (ou uma fonte de clock externa), um oscilador interno (clocks RC) e um sistema de Phase-Locked Loops (PLLs) para gerar os diversos clocks necessários para o funcionamento dos periféricos.
- A configuração dos clocks no RP2040 é feita por meio de registradores específicos, que permitem:
 - » Selecionar a fonte de clock para cada um dos módulos.
 - » Configurar os divisores de clock para ajustar a frequência de saída.
 - » Ativar/desativar clocks para economia de energia.

Configurações do clock

Aqui estão as principais configurações disponíveis:

- Fontes de Clock:
 - » XOSC (External Crystal Oscillator)
 - » ROSC (Ring Oscillator)
 - » CLK_SYS (System Clock)
 - » CLK_PERI (Peripheral Clock)
 - » CLK_USB
 - » CLK_ADC
 - » CLK_RTC
- PLLs (Phase-Locked Loops)
 - » PLL_SYS
 - » PLL_USB
- Divisores de Clock:
 - » CLK_REF (Clock de Referência)
 - » CLK_SYS
 - » CLK_PERI
 - » CLK_USB
 - » CLK_ADC
 - » CLK_RTC
- Módulo Wi-Fi
 - » Chip adicional de Wi-Fi (CYW43439).
 - » Configuração de clock específica que não afeta diretamente o sistema de clocks do RP2040

Visão geral do clock

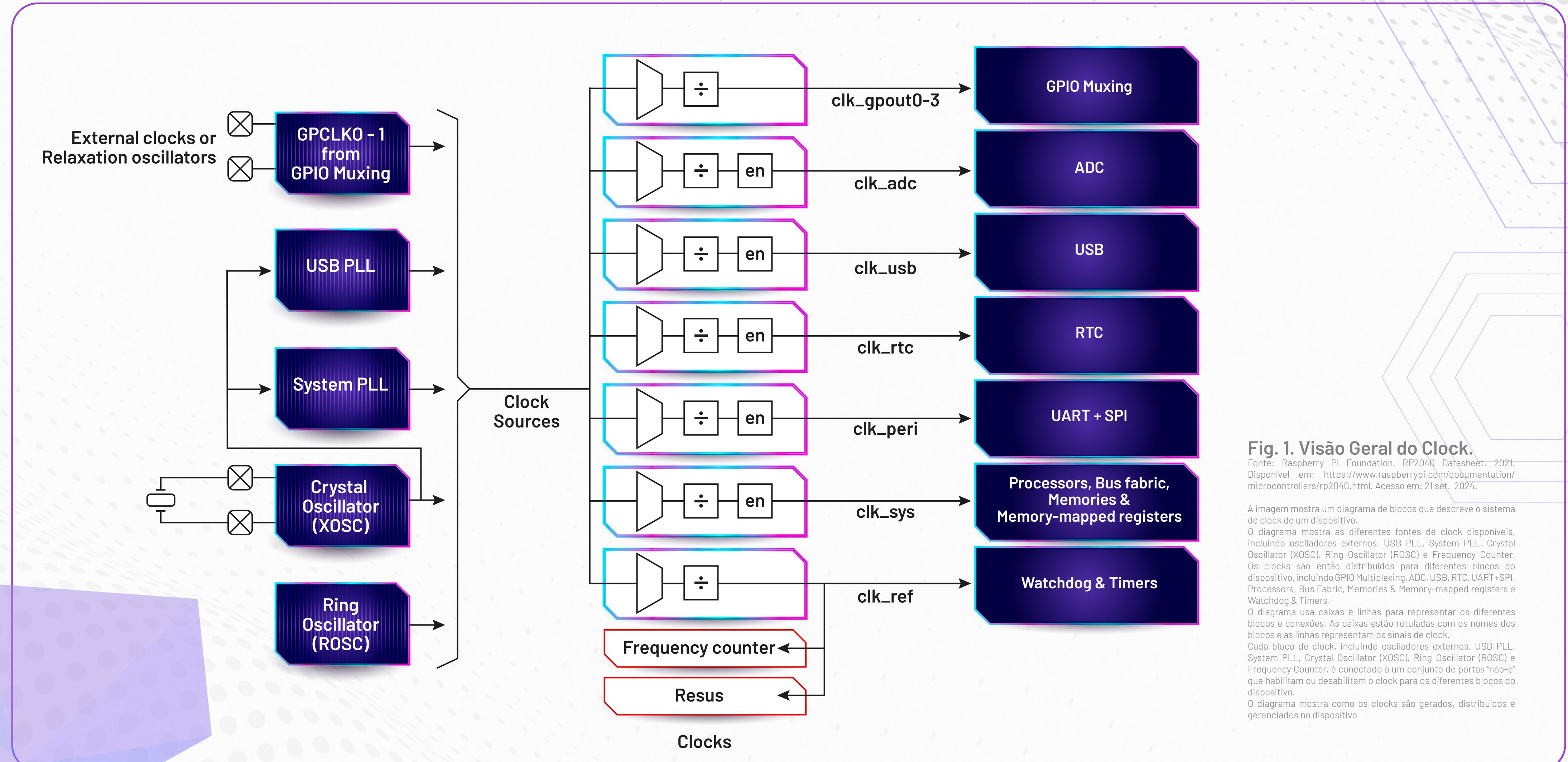


Fig. 1. Visão Geral do Clock.

Fonte: Raspberry Pi Foundation. RP2040 Datasheet. 2021. Disponível em: <https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html>. Acesso em: 21 set. 2024.

A imagem mostra um diagrama de blocos que descreve o sistema de clock de um dispositivo.

O diagrama mostra as diferentes fontes de clock disponíveis, incluindo osciladores externos, USB PLL, System PLL, Crystal Oscillator (XOSC), Ring Oscillator (ROSC) e Frequency Counter. Os clocks são então distribuídos para diferentes blocos do dispositivo, incluindo GPIO Multiplexing, ADC, USB, RTC, UART+SPI, Processors, Bus Fabric, Memories & Memory-mapped registers e Watchdog & Timers.

O diagrama usa caixas e linhas para representar os diferentes blocos e conexões. As caixas estão rotuladas com os nomes dos blocos e as linhas representam os sinais de clock.

Cada bloco de clock, incluindo osciladores externos, USB PLL, System PLL, Crystal Oscillator (XOSC), Ring Oscillator (ROSC) e Frequency Counter, é conectado a um conjunto de portas "não-e" que habilitam ou desabilitam o clock para os diferentes blocos do dispositivo.

O diagrama mostra como os clocks são gerados, distribuídos e gerenciados no dispositivo.

Exemplo Prático 01

```
#include "pico/stdlib.h"
#include "hardware/clocks.h"
#include "hardware/pll.h"

int main() {
    // Inicializa os clocks
    clocks_init(); // Alterado para clocks_init()

    // Configura o PLL_SYS para 125 MHz
    pll_init(pll_sys, 1, 1500 * MHZ, 6 * MHZ, 1); // Multiplicador de 125 (750/6 MHz)

    // Configura o CLK_SYS para usar o PLL_SYS como fonte de clock
    clock_configure(clk_sys,
                    CLOCKSYS_CLK_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX,
                    CLOCKSYS_CLK_CTRL_AUXSRC_VALUE_CLKSRC_PLL_SYS,
                    125 * MHZ, // Frequência desejada
                    125 * MHZ); // Frequência de origem

    // A partir daqui, o CLK_SYS opera a 125 MHz
    printf("CLK_SYS está operando a 125 MHz\n"); // Imprime a mensagem

    while (true) {
        // Loop principal
    }
}
```

Fig. 2. Exemplo de código.

Fonte: imagem do autor

A imagem mostra um código em C voltado para configuração de clock em um sistema embarcado. O objetivo principal do código é alterar a frequência de operação do sistema. Aqui está uma descrição detalhada:

1. Inclusão de bibliotecas:
 - O código inclui três bibliotecas com os seguintes cabeçalhos:
 - `#include <stdio.h>`: para entrada e saída padrão.
 - `#include "pico/stdlib.h"`, para funções e definições padrão da placa Raspberry Pi Pico.
 - `#include "hardware/clocks.h"`, para manipulação e controle dos clocks do hardware.
2. Função 'main()':
 - O código principal do programa começa dentro da função `main`.
3. Inicialização de hardware:
 - `clocks_init()`: inicializa todas as interfaces de entrada e saída padrão.
 - `clock_init()`: inicializa o sistema de clocks (relógios) do hardware.
4. Configuração dos clocks:
 - O código contém dois trechos de configuração para definir o sistema de PLLs (Phase-Locked Loops) que controlam a frequência de clock.
 - A função `set_sys_clock_pll` define a frequência de operação do sistema para 125 MHz.
 - Outra função `set_pll` também é utilizada, provavelmente para configurar uma PLL com valores apropriados para manter a precisão do clock.
5. Mensagem de saída:
 - `printf("CLK_SYS está operando a 125 MHz\n")` imprimi uma mensagem indicando que o sistema está operando a 125 MHz.
6. Loop infinito:
 - `while(true)` mantém o programa em execução indefinidamente.

Resumindo, o código configura o clock do sistema para operar a 125 MHz e imprime uma mensagem informando isso.

Exemplo Prático 01

The screenshot shows a software interface for developing and simulating embedded systems. On the left, there is a code editor with tabs for "main.c" and "diagram.json". The "main.c" tab contains the following C code:

```
1 #include "pico/stdlib.h"
2 #include "hardware/clocks.h"
3 #include "hardware/pll.h"
4
5 int main() {
6     // Inicializa a comunicação serial
7     stdio_init_all();
8
9     // Inicializa os clocks
10    clocks_init();
11
12    // Configura o PLL_SYS para 125 MHz
13    pll_init(pll_sys, 1, 1500 * MHZ, 6 * MHZ, 1); // Multiplicador de 125 (750/6 MHz)
14
15    // Configura o CLK_SYS para usar o PLL_SYS como fonte de clock
16    clock_configure(clk_sys,
17                    CLOCKS_CLK_SYS_CTRL_SRC_VALUE_CLKSRC_CLK_SYS_AUX,
18                    CLOCKS_CLK_SYS_CTRL_AUXSRC_VALUE_CLKSRC_PLL_SYS,
19                    125 * MHZ, // Frequência desejada
20                    125 * MHZ); // Frequência de origem
21
22    // A partir daqui, o CLK_SYS opera a 125 MHz
23    printf("CLK_SYS está operando a 125 MHz\n"); // Imprime a mensagem
24
25    while (true) {
26        // Loop principal - você pode adicionar mais lógica aqui, se necessário
27    }
}
```

On the right, there is a "Simulation" window. It features a top bar with a green circular button, a square button, and a pause button, along with the text "00:33.803" and "99%". Below this is a schematic diagram of a Raspberry Pi Pico W board. At the bottom of the simulation window, the text "CLK_SYS está operando a 125 MHz" is displayed.

Fig. 3. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409672647153452033>

A imagem mostra um trecho de código em um editor de texto, provavelmente programado em C ou C++. No topo da janela, há uma aba com o nome do arquivo "main.c", o que sugere que é um arquivo principal de código-fonte. O código está estruturado da seguinte forma:

1. Inclusões de bibliotecas:
 - #include "pico/stdlib.h"
 - #include "hardware/clocks.h"
 - #include "hardware/pll.h"
2. Função principal(main):
 - Dentro da função 'int main()', há comentários em português que orientam sobre as etapas do código.
 - Primeira etapa: Inicializa a comunicação serial e a função 'stdio_init_all()'.
 - Inicialização dos clocks com 'Clocks_init()'.
3. Configuração da PLL:
 - Um comentário explica que a PLL é configurada para 125 MHz.
 - A função 'pll_init()' é chamada com parâmetros específicos para configurar o clock.
4. Configuração de clock para 'CLK_SYS':
 - A função 'clock_configure()' configura o sistema de clocks para operar em 125 MHz usando a fonte PLL_SYS.
5. Saída no console:
 - Um comando 'printf' imprime a mensagem: "CLK_SYS está operando a 125 MHz".

6. Loop principal(while):

- O loop 'while(true)' está presente, sugerindo que o programa continua executando indefinidamente.

O código está bem comentado, com orientações em português sobre o que cada trecho faz. É um exemplo típico de inicialização de sistema e configuração de clocks para uma aplicação embarcada. A interface de usuário mostra vários botões de controle, incluindo botões de reproduzir, pausar e parar, bem como um contador que mostra o tempo de execução atual da simulação. A parte inferior da interface de usuário mostra uma mensagem em português que diz "CLK_SYS está operando a 125 MHz", indicando a frequência de clock do dispositivo simulado.

A imagem também mostra um gráfico de barras que representa a atividade do dispositivo, bem como botões para aumentar, diminuir ou ajustar o zoom do gráfico. A interface de usuário é mostrada contra um fundo cinza escuro e é organizada em duas seções: a seção superior mostra a imagem do dispositivo e os botões de controle da simulação, enquanto a seção inferior mostra a mensagem de estado do dispositivo e o gráfico de barras.

A imagem sugere que o dispositivo simulado é um microcontrolador e que o usuário está trabalhando com o simulador para analisar o funcionamento do dispositivo.

O link que você forneceu leva para um projeto de eletrônica chamado "Wokwi". Neste projeto, você pode interagir com um circuito eletrônico, que inclui um sensor de temperatura e um LED que pisca de acordo com a temperatura.

Visão Geral

O que é o temporizador?

- Um temporizador em um microcontrolador é um componente que permite medir intervalos de tempo ou gerar eventos em tempos específicos. Ele pode ser usado para diversas funções, como:
 - » **Contagem de Intervalos:** O temporizador pode contar pulsos de um clock interno, permitindo medir intervalos de tempo com precisão.
 - » **Geração de Interrupções:** Pode gerar interrupções após um certo tempo, permitindo que o microcontrolador execute outras tarefas enquanto aguarda.
 - » **Controle de Frequência:** Usado em aplicações que requerem sinais de PWM(modulação por largura de pulso) para controle de motores, LEDs, etc.
 - » **Divisão de Frequência:** Permite dividir a frequência de um sinal, útil em diversas aplicações de controle.

Visão Geral

O que é o temporizador?

- RP2040 oferece uma variedade de temporizadores que podem ser usados para diferentes propósitos:
 - » **Temporizadores de Hardware (Timer Hardware):** O RP2040 tem vários temporizadores de hardware que podem ser usados para gerar interrupções temporizadas ou medir o tempo com precisão. Esses temporizadores são configuráveis e oferecem uma ampla gama de possibilidades para gerar eventos baseados em tempo.
 - » **Temporizador de Sistema (SysTick):** Este temporizador é útil para gerar interrupções regulares. É frequentemente usado para implementar sistemas de tempo real ou para criar intervalos regulares para execução de tarefas.
 - » **Temporizadores de Pulse Width Modulation (PWM):** O RP2040 possui blocos de PWM que permitem gerar sinais PWM com precisão. Isso é útil para controlar servomecanismos, motores e outros dispositivos baseados em PWM. Esse assunto será visto no capítulo 7.
 - » **Temporizadores de Contagem (Counters):** O RP2040 pode ser configurado para contar eventos ou pulsos, o que é útil para aplicações que requerem a medição precisa de frequência ou eventos.

Exemplo Prático 02

```
1 #include "pico/stdc.h"
2 #include "pico/time.h"
3
4 int main() {
5     stdio_init_all();
6
7     // Configurar o intervalo para 1 segundo (1000 milissegundos)
8     uint32_t interval = 1000; // Intervalo em milissegundos
9
10    // Configurar o próximo tempo de acordar
11    absolute_time_t next_wake_time = delayed_by_us(get_absolute_time(), interval * 1000);
12
13    while (true) {
14        // Verificar se o tempo atual atingiu o tempo definido
15        if (time_reached(next_wake_time)) {
16            // Imprimir mensagem
17            printf("1 segundo passou\n");
18
19            // Atualizar o próximo tempo de acordar
20            next_wake_time = delayed_by_us(next_wake_time, interval * 1000);
21        }
22
23        // Pequena pausa para reduzir o uso da CPU
24        sleep_ms(1);
25    }
26}
27
```

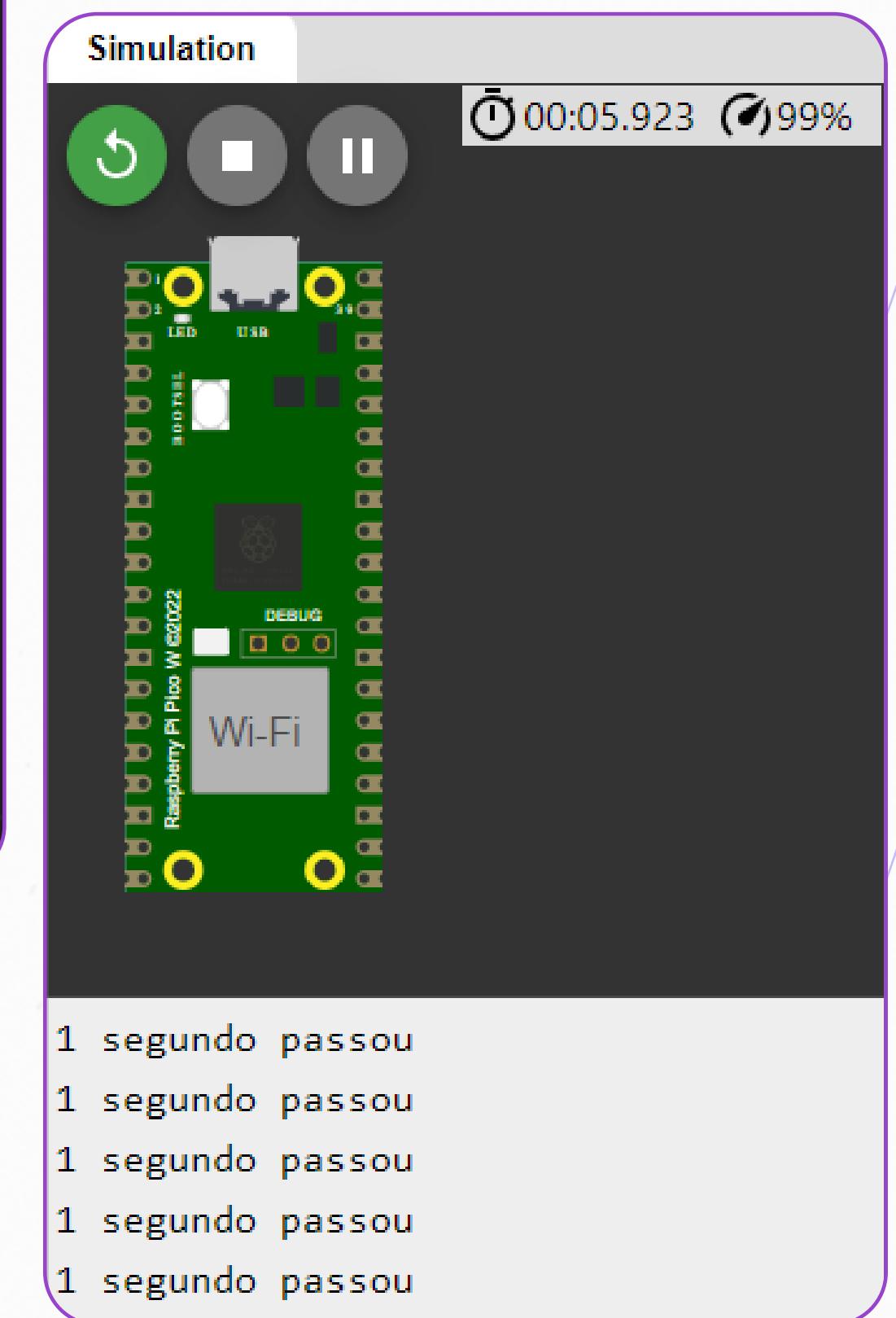


Fig. 4. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409690557858797569>

A imagem mostra um trecho de código em C usado para controlar o tempo e imprimir uma mensagem repetidamente a cada segundo em um microcontrolador Raspberry Pi Pico. O código está formatado com cores para facilitar a leitura, destacando palavras-chave, comentários e funções.

1. Inclusão de bibliotecas:
 - "#include "pico/stdc.h"": Inclui a biblioteca padrão para funções gerais no Raspberry Pi Pico.
 - "#include "pico/time.h"": Inclui a biblioteca que fornece funções relacionadas ao tempo no Pico.
2. Função principal ('main'):
 - Dentro da função principal 'int main() {}', a primeira linha chama 'stdio_init_all()', que inicializa o sistema de entrada/saída padrão.
3. Configuração de Intervalo:
 - 'uint32_t interval = 1000;': Define uma variável chamada 'interval' do tipo 'uint32_t' (inteiro sem sinal de 32 bits) e atribui a ela o valor '1000', que corresponde a um intervalo de 1000 milissegundos (1 segundo).
 - 'absolute_time_t next_wake_time = delayed_by_us(get_absolute_time(), interval * 1000);': Configura o próximo tempo de "acordar" do sistema, utilizando a função 'get_absolute_time()' para obter o tempo atual e a função 'delayed_by_us' para definir o próximo tempo de espera, multiplicando o intervalo por 1000 para converter de milissegundos para microsegundos.
4. Laço infinito ('while true'):
 - O código entra em um laço infinito 'while(true){}', garantindo que as instruções dentro dele sejam executadas repetidamente.
 - Dentro do laço:
 - 'if (time_reached(next_wake_time)) {}': Verifica se o tempo atual atingiu o próximo tempo de "acordar", usando a função 'time_reached'.
 - 'printf("1 segundo passou\n");': Se o tempo definido foi alcançado, imprime a mensagem "1 segundo passou" na saída padrão.
 - 'next_wake_time = delayed_by_us(next_wake_time, interval * 1000);': Atualiza o valor de 'next_wake_time', somando o intervalo definido para continuar a repetição a cada segundo.
5. Pausa para reduzir o uso da CPU:
 - 'sleep_ms(1);': Insere uma pausa de 1 milissegundo para evitar o uso excessivo da CPU, permitindo que o processador "descanse" brevemente antes de continuar o ciclo.
6. Encerramento:
 - O laço e a função principal são encerrados com chaves '}'.

A imagem mostra uma simulação de um circuito eletrônico, com um pequeno computador chamado "Raspberry Pi Pico W" no centro. Na parte superior da imagem, você pode ver três botões: O primeiro é uma seta circular que representa "reiniciar". O segundo é um quadrado que significa "pausar". O terceiro é um símbolo de pausa dupla, indicando "parar". Do lado direito da imagem, há dois indicadores: O primeiro é um cronômetro que mostra "00:05.923", indicando o tempo de execução da simulação. O segundo é um ícone de uma bateria com a porcentagem "99%", mostrando que a simulação está quase completa.

Na parte inferior da imagem, há um texto escrito em português, informando que um "segundo passou" cinco vezes seguidas. Isso indica que o circuito está funcionando e a simulação está em andamento.

O "Raspberry Pi Pico W" é uma placa de circuito verde com vários pinos, alguns dos quais são marcados com um círculo amarelo. No centro da placa, há um retângulo cinza com o texto "Wi-Fi", indicando que o computador pode se conectar à internet. A imagem como um todo representa um circuito eletrônico em funcionamento, com a simulação mostrando o estado atual do circuito e o tempo que já se passou desde o início.

Exemplo Prático 03

```
1 #include "pico/stdlib.h"
2 #include "hardware/timer.h"
3
4 // Função que será chamada na interrupção do temporizador
5 bool repeating_timer_callback(struct repeating_timer *t) {
6     printf("1 segundo passou\n");
7     return true; // Retorna true para repetir a interrupção
8 }
9
10 int main() {
11     stdio_init_all();
12
13     // Configurar um temporizador de repetição
14     struct repeating_timer timer;
15     // Configurar o temporizador para chamar a função a cada 1 segundo
16     add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer);
17
18     while (true) {
19         // No loop principal, pode realizar outras tarefas
20         sleep_ms(1000);
21     }
22 }
```

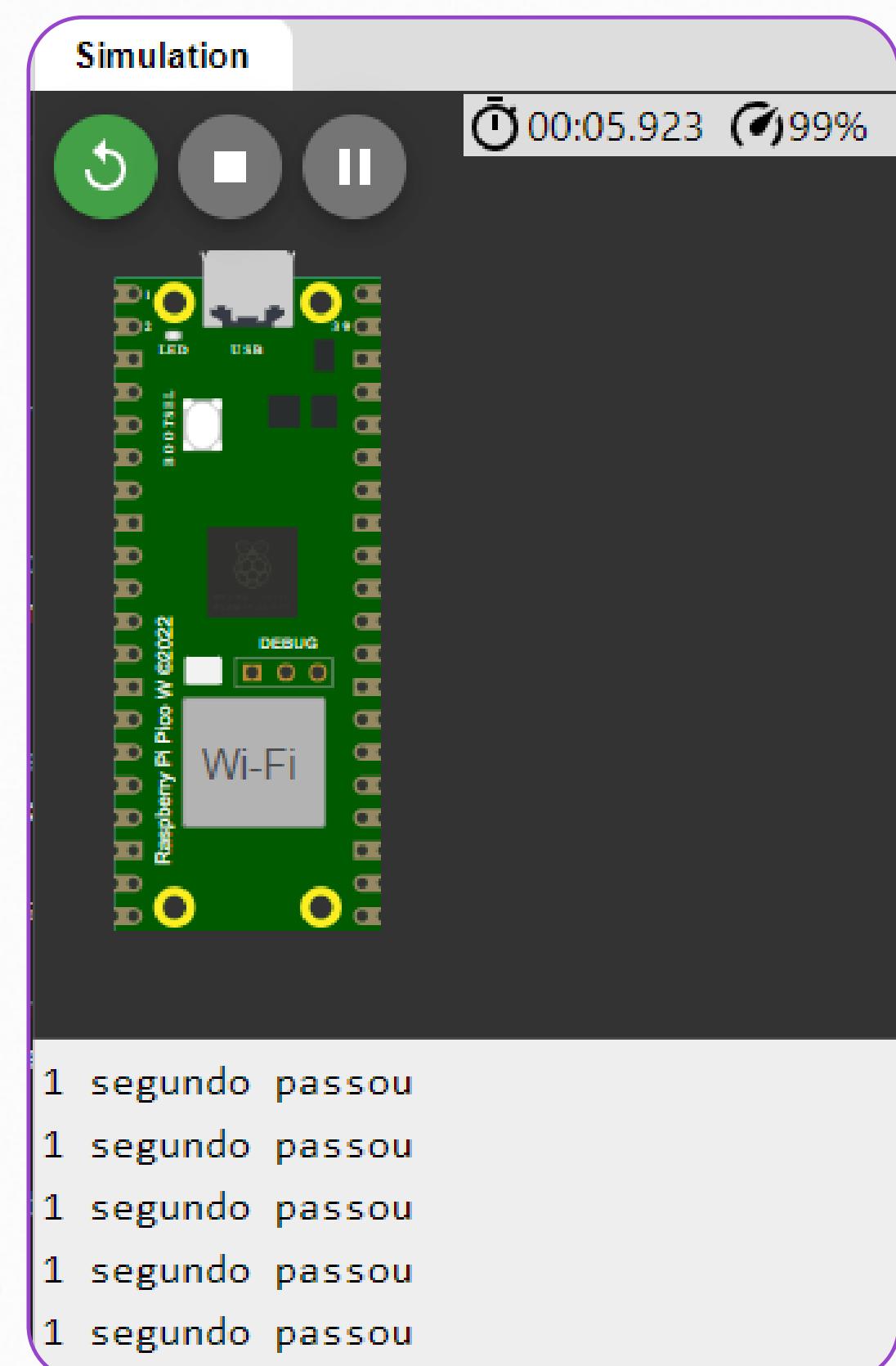


Fig. 5. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409690905220586497>

A imagem mostra um trecho de código em C que configura um temporizador repetitivo no microcontrolador Raspberry Pi Pico, usado para executar uma ação a cada intervalo de tempo definido. O código está formatado com cores, com palavras-chave, funções e comentários destacados.

1. Inclusão de bibliotecas:
 - '#include "pico/stdcib.h": Inclui a biblioteca padrão do Raspberry PI Pico para funções básicas.
 - '#include "hardware/timer.h": Inclui a biblioteca de hardware que fornece funções relacionadas ao uso de temporizadores.
 2. Função de callback('repeating_timer_callback'):
 - Esta função será chamada cada vez que o temporizador atingir o intervalo definido.
 - 'bool repeating_timer_callback(struct repeating_timer *t) ': A função é declarada para aceitar um parâmetro do tipo 'struct repeating_timer'.
 - Dentro da função:
 - 'printf("1 segundo passou\n");': Imprime a mensagem "1 segundo passou" na saída padrão cada vez que a função é chamada.
 - 'return true;': Retorna 'true' para garantir que o temporizador continue chamando a função repetidamente.
 3. Função principal('main'):
 - A função 'int main(){' inicia o programa.
 - 'stdio_init_all();': Inicializa o sistema de entrada/saída padrão.
 4. Configuração do temporizador:
 - 'struct repeating_timer timer;': Declara uma variável 'timer' do tipo 'struct repeating_timer' para armazenar o temporizador.
 - 'add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer);': Configura o temporizador para chamar a função 'repeating_timer_callback' a cada 1000 milissegundos (1 segundo). A função 'add_repeating_timer_ms' define que o temporizador chamará essa função repetidamente, passando o valor '1000' para definir o intervalo de 1 segundo.
 5. Laço principal('while(true)'):
 - O código entra em um laço infinito 'while (true) {', que roda continuamente.
 - Dentro do laço, há um comando 'sleep_ms(1000);', que faz o microcontrolador "dormir" por 1000 milissegundos (1 segundo), permitindo que outras tarefas sejam realizadas enquanto o temporizador continua funcionando em segundo plano.
 6. Encerramento:
 - O laço e a função principal são encerrados com chaves '}'.
- Resumo: O código configura um temporizador repetitivo que chama a função 'repeating_timer_callback' a cada 1 segundo. A função imprime a mensagem "1 segundo passou" e continua sendo chamada repetidamente. Enquanto isso, o laço principal pode executar outras tarefas, com o processador aguardando por 1 segundo antes de repetir o ciclo.

A imagem mostra uma simulação de um circuito eletrônico, com um pequeno computador chamado "Raspberry Pi Pico W" no centro. Na parte superior da imagem, você pode ver três botões: O primeiro é uma seta circular que representa "reiniciar". O segundo é um quadrado que significa "pausar". O terceiro é um símbolo de pausa dupla, indicando "parar". Do lado direito da imagem, há dois indicadores: O primeiro é um cronômetro que mostra "00:05.923", indicando o tempo de execução da simulação.

O segundo é um ícone de uma bateria com a porcentagem "99%", mostrando que a simulação está quase completa.

Na parte inferior da imagem, há um texto escrito em português,

informando que um "segundo passou" cinco vezes seguidas. Isso

indica que o circuito está funcionando e a simulação está em

andamento.

O "Raspberry Pi Pico W" é uma placa de circuito verde com vários

pinos, alguns dos quais são marcados com um círculo amarelo.

No centro da placa, há um retângulo cinza com o texto "Wi-Fi",

indicando que o computador pode se conectar à internet.

A imagem como um todo representa um circuito eletrônico em

funcionamento, com a simulação mostrando o estado atual do

círcuito e o tempo que já se passou desde o início.

Comparação entre os exemplos 2 e 3

- **Abordagem:** O primeiro código, utilizando temporizador de sistema, utiliza uma abordagem manual de controle de tempo, enquanto o segundo, utilizando temporizador de hardware, usa um temporizador automático.
- **Complexidade:**
 - » O primeiro código é mais complexo devido ao gerenciamento manual do tempo.
 - » O segundo código é mais simples e modular, com separação clara entre a lógica de temporização e o loop principal.
- **Conclusão:** Ambos os códigos têm a mesma funcionalidade de imprimir uma mensagem a cada segundo, mas a implementação do segundo código é mais limpa e eficiente. Utilizando uma abordagem de temporização que evita a necessidade de controle manual de tempo. O primeiro código oferece mais flexibilidade, mas à um custo de complexidade. A escolha entre os dois depende das necessidades específicas do seu projeto.

Exemplo Prático 04

The screenshot shows the Wokwi platform interface. On the left, the code editor displays `main.c` with the following C code:

```
1 #include "pico/stdlib.h"
2 #include "pico/time.h"
3 int main() {
4     // Inicializar comunicação padrão (para printf)
5     stdio_init_all();
6     // Configurar o pino 12 como saída
7     const uint LED_PIN = 12;
8     gpio_init(LED_PIN);
9     gpio_set_dir(LED_PIN, GPIO_OUT);
10    // Definir o intervalo para 1 segundo (1000 milissegundos)
11    uint32_t interval = 1000; // Intervalo em milissegundos
12    // Definir o próximo tempo para acordar
13    absolute_time_t next_wake_time = delayed_by_us(get_absolute_time(), interval * 1000);
14    // Estado inicial do LED (desligado)
15    bool led_on = false;
16    while (true) {
17        // Verificar se o tempo atual atingiu o próximo tempo definido
18        if (time_reached(next_wake_time)) {
19            // Alternar o estado do LED manualmente
20            led_on = !led_on;
21            gpio_put(LED_PIN, led_on);
22            // Atualizar o próximo tempo de acordar
23            next_wake_time = delayed_by_us(next_wake_time, interval * 1000);
24        }
25        // Pequena pausa para reduzir o uso da CPU
26        sleep_ms(1);
27    }
28 }
```

On the right, the simulation window shows a Raspberry Pi Pico W board with a breadboard setup. A blue LED is connected to pin 12 through a resistor. The board also has a Wi-Fi module and a USB port. The simulation controls include a play button, a plus sign for zoom, and a three-dot menu.

Fig. 6. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409692282956192769>

A imagem exibe um código em C que implementa um temporizador repetitivo no microcontrolador Raspberry Pi Pico. O código está formatado com cores que destacam palavras-chave, funções e comentários. Aqui está a descrição detalhada do código:

1. Inclusão de bibliotecas:
 - '#include "pico/stdlib.h": Importa a biblioteca padrão do Raspberry Pi Pico para funções básicas.
 - '#include "hardware/timer.h": Importa a biblioteca para lidar com temporizadores no hardware.
 2. Função de callback('repeating_timer_callback'):
 - 'bool repeating_timer_callback(struct repeating_timer *t)': Declara uma função chamada 'repeating_timer_callback' que será chamada cada vez que o temporizador atingir o tempo definido.
 - Dentro da função:
 - 'printf("1 segundo passou\n")': Exibe a mensagem "1 segundo passou" na saída padrão (como um terminal ou console).
 - 'return true;': Retorna 'true' para garantir que o temporizador continue chamando essa função repetidamente.
 3. **Função principal('main'):
 - A função 'int main(){}' inicia o código principal.
 - 'stdio_init_all();': Inicializa o sistema de entrada/saída padrão para permitir o uso de funções como 'printf'.
 4. Configuração do temporizador:
 - 'struct repeating_timer timer;': Declara uma variável 'timer' do tipo 'struct repeating_timer', usada para armazenar o temporizador.
 - 'add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer)': Configura o temporizador para chamar a função 'repeating_timer_callback' a cada 1000 milissegundos (1 segundo), repetidamente. O 'NULL' significa que nenhum dado extra está sendo passado para a função de callback.
 5. **Laço principal('while(true)'):
 - O código entra em um laço infinito com 'while(true){}', que garante a execução contínua.
 - Dentro do laço, o comando 'sleep_ms(1000);' faz o microcontrolador "dormir" por 1000 milissegundos (1 segundo) em cada ciclo. Durante esse tempo, o temporizador continua funcionando em segundo plano, permitindo que outras tarefas sejam executadas entre as pausas.
 6. Encerramento:
 - O laço e a função principal são encerrados com as chaves de fechamento '}'.
- **Resumo**: Este código configura um temporizador que chama a função 'repeating_timer_callback' a cada 1 segundo. A função imprime a mensagem "1 segundo passou" e é executada repetidamente. Enquanto isso, o código principal fica em um loop infinito, com o processador aguardando 1 segundo em cada iteração.

Exemplo Prático 05

The screenshot shows the Wokwi development environment. On the left, the code editor displays `main.c` with the following C code:

```
1 #include "pico/stdlib.h"
2 #include "hardware/timer.h"
3 // Definir o pino do LED
4 const uint LED_PIN = 12;
5 // Função que será chamada na interrupção do temporizador
6 bool repeating_timer_callback(struct repeating_timer *t) {
7     // Alternar o estado do LED
8     static bool led_on = false;
9     led_on = !led_on;
10    gpio_put(LED_PIN, led_on);
11    // Imprimir mensagem opcional (para fins de depuração)
12    printf("LED %s\n", led_on ? "ligado" : "desligado");
13    return true; // Retorna true para continuar repetindo a interrupção
14 }
15 int main() {
16     // Inicializar comunicação padrão (para printf)
17     stdio_init_all();
18     // Configurar o pino 12 como saída
19     gpio_init(LED_PIN);
20     gpio_set_dir(LED_PIN, GPIO_OUT);
21     // Configurar um temporizador de repetição
22     struct repeating_timer timer;
23     // Configurar o temporizador para chamar a função a cada 1 segundo
24     add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer);
25     while (true) {
26         // Loop principal está livre para outras tarefas
27     }
28 }
```

On the right, the simulation window shows a Raspberry Pi Pico W board with a breadboard. A blue LED is connected to pin 12 (labeled `LED`) through a resistor. The board also has a Wi-Fi module and an OLED screen. The simulation interface includes a toolbar with play, stop, and settings buttons.

Fig. 7. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409325289916859393>

A imagem contém uma interface de simulação do Wokwi, com código C e um diagrama de hardware representando uma placa microcontroladora, possivelmente um Raspberry Pi Pico. Descrição do Código: O código na esquerda utiliza uma biblioteca do Pico para configurar um LED que alterna o estado (liga/desliga) em intervalos regulares usando um temporizador.

Bibliotecas Incluídas: `pico/stdcib.h` e `hardware/timer.h` são importadas para controlar o hardware do microcontrolador e configurar o temporizador.

Função de Callback para o Temporizador: `bool repeating_timer_callback(struct repeating_timer *t)`: Esta função é chamada a cada interrupção do temporizador, alternando o estado do LED. Usa uma variável estática `led_on` para alternar o estado do LED, ligando e desligando com `gpio_put()`.

Também imprime o estado do LED (ligado ou desligado) usando `printf()`. Função Main: Inicializa a comunicação padrão (`stdio_init_all()`), configura o LED como saída e inicia o temporizador para chamar a função `repeating_timer_callback()` a cada 1 segundo (1000 ms) com `add_repeating_timer_ms()`.

Loop Principal: O loop `while (true)` está vazio, sugerindo que o microcontrolador está livre para realizar outras tarefas enquanto o temporizador gerencia o LED.

Diagrama à Direita: O diagrama mostra uma placa de microcontrolador com um LED conectado. O LED piscá, alternando entre ligado e desligado. Ao lado, há um painel que indica o estado do LED:

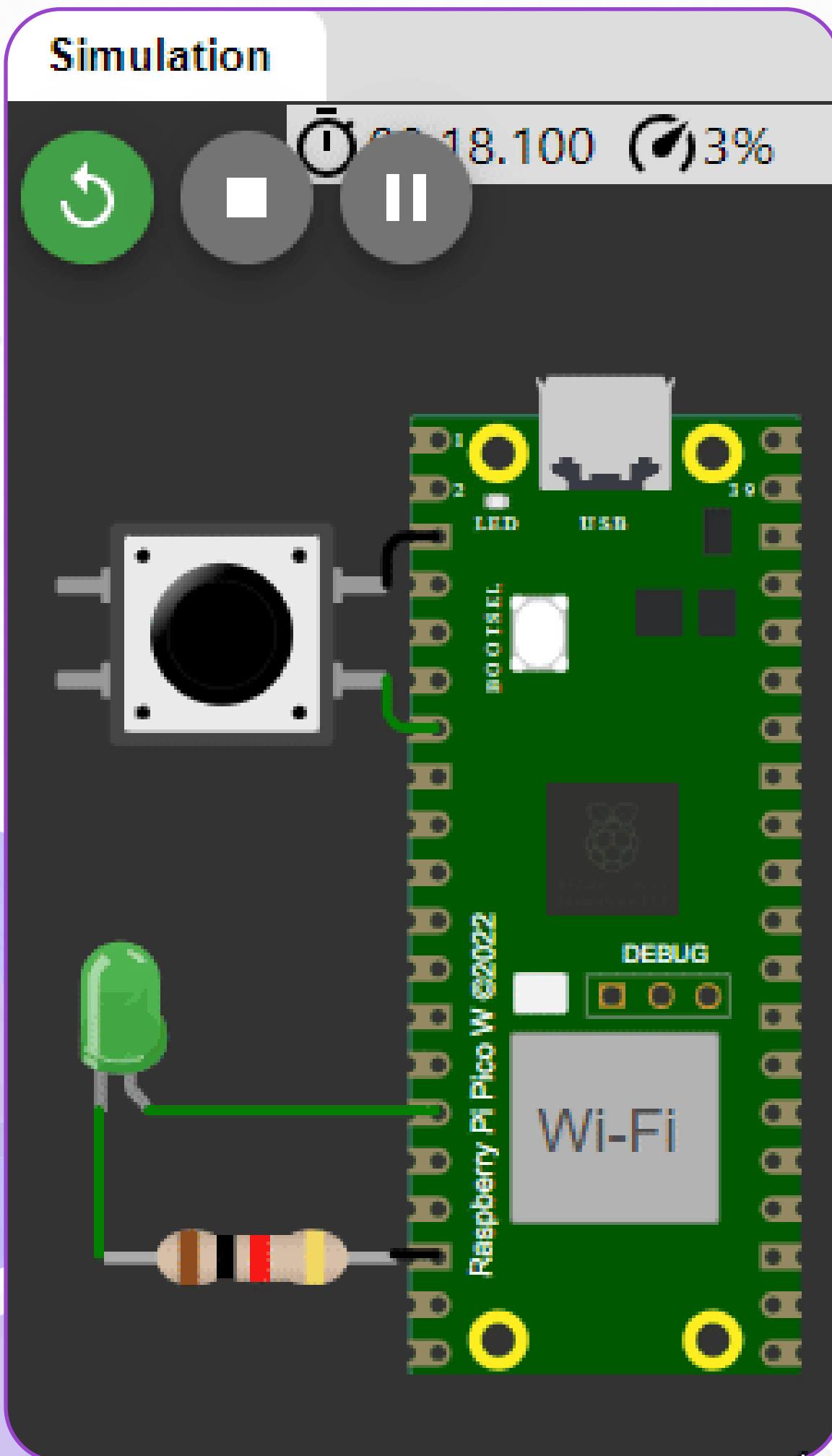
LED ligado
LED desligado
LED ligado (aparecem essas mensagens repetidas).
A simulação sugere que o LED está piscando a cada segundo, conforme o código.

Pisca LED com temporizador

Placa BitDogLab

Usar vídeo do documento
no lists

Exemplo Prático 06



```
1 // #include "pico/stdlib.h"
2 #include "pico/time.h"
3 const uint LED_PIN = 11; // Pino do LED
4 const uint BUTTON_PIN = 5; // Pino do botão
5 bool led_on = false; // Estado do LED
6 bool led_active = false; // Indica se o LED está aceso
7 absolute_time_t turn_off_time; // Tempo para desligar o LED
8 int64_t turn_off_callback(alarm_id_t id, void *user_data) {
9     // Desligar o LED
10    gpio_put(LED_PIN, false);
11    led_active = false;
12    return 0; // Não repetir o alarme
13 }
14 int main() {
15     // Inicializar comunicação padrão
16     stdio_init_all();
17     // Configurar o pino 11 como saída (LED) e o pino 5 como entrada (botão)
18     gpio_init(LED_PIN);
19     gpio_set_dir(LED_PIN, GPIO_OUT);
20     gpio_init(BUTTON_PIN);
21     gpio_set_dir(BUTTON_PIN, GPIO_IN);
22     gpio_pull_up(BUTTON_PIN); // Habilitar o pull-up interno
23     while (true) {
24         // Verificar se o botão foi pressionado
25         if (gpio_get(BUTTON_PIN) == 0 && !led_active) {
26             // Adicionar um pequeno debounce
27             sleep_ms(50);
28             if (gpio_get(BUTTON_PIN) == 0) {
29                 // Acender o LED
30                 gpio_put(LED_PIN, true);
31                 led_active = true;
32                 // Agendar para desligar o LED em 3 segundos (3000 ms)
33                 add_alarm_in_ms(3000, turn_off_callback, NULL, false);
34             }
35         }
36         // Pequena pausa para reduzir o uso da CPU
37         sleep_ms(10);
38     }
39 }
```

Fig. 8. Exemplo Prático.

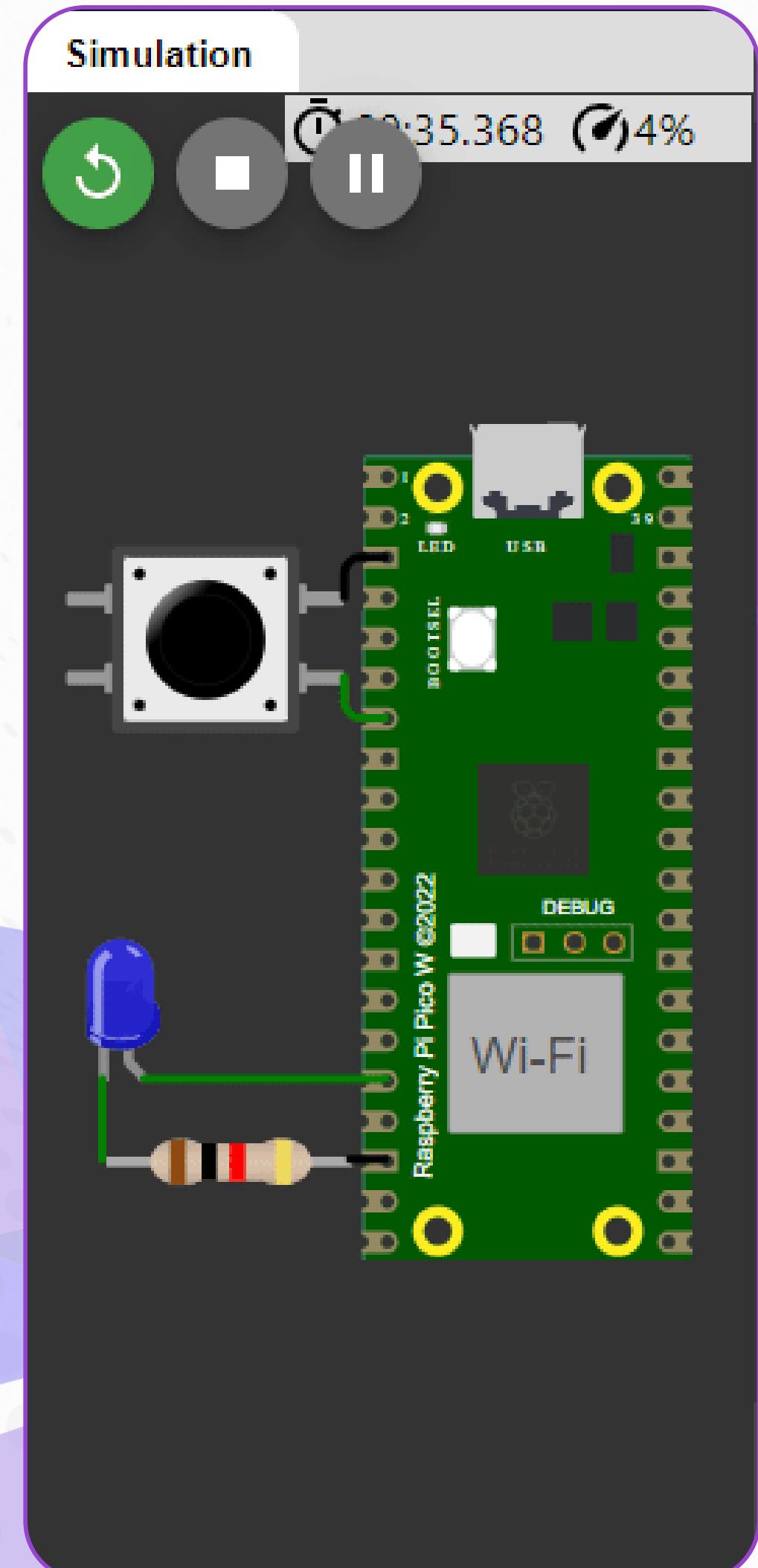
Fonte: <https://wokwi.com/projects/409696442353327105>

A imagem mostra um exemplo prático de um circuito e código relacionado a um microcontrolador, com o título "Exemplo Prático 06". Há duas seções principais: Diagrama à Esquerda (Simulação de Hardware): O diagrama mostra uma placa de microcontrolador, possivelmente um Raspberry Pi Pico, conectado a um LED verde e a um botão. O botão está conectado a um dos pinos do microcontrolador e o LED está ligado a outro pino, com um resistor em série. Há controles de simulação no topo, como um botão de "play" (para iniciar a simulação), "pause" e um contador de tempo de simulação em andamento. Código à Direita (Em C): O código usa a biblioteca pico/stdlib.h para controle do microcontrolador e pico/time.h para trabalhar com temporizadores. Definições de Pins: Define LED_PIN para o pino do LED e BUTTON_PIN para o pino do botão. Variáveis booleanas para armazenar o estado do LED (led_on) e se o LED foi ativado (led_active). Função de Callback para Temporizador: turn_off_callback() é uma função que desliga o LED após um tempo determinado, usando gpio_put() para definir o pino do LED como desligado. Função Main: Inicialização: Configura a comunicação padrão, inicializa o pino do LED como saída e o do botão como entrada com pull-up. Loop Principal: Verifica se o botão foi pressionado (gpio_get()), adiciona um pequeno atraso (debounce) e se o botão estiver pressionado, acende o LED e agenda o desligamento automático em 3 segundos com add_alarm_in_ms(). Um pequeno atraso (sleep_ms(10)) é adicionado para evitar sobrecarga de CPU. Abaixo do diagrama, há um link de simulação rotulado como "Cap5.Ex6", sugerindo que o código e o circuito fazem parte de um exercício prático em algum capítulo de um material didático ou livro.

Exemplo Prático 06 aplicado

Usar vídeo do documento
no lists

Exemplo Prático 07



```
#include "pico/stdlib.h"
#include "hardware/timer.h"
const uint LED_PIN = 12; // Definir o pino do LED
const uint BUTTON_PIN = 5; // Definir o pino do botão
// Variáveis de controle
bool led_on = false; // Estado do LED
absolute_time_t turn_off_time; // Tempo de desligar o LED
bool led_active = false; // Indica se o LED deve permanecer aceso
// Função chamada pelo temporizador a cada 1 segundo
bool repeating_timer_callback(struct repeating_timer *t) {
    // Verificar se o LED deve ser desligado
    if (led_active && absolute_time_diff_us(get_absolute_time(), turn_off_time) <= 0) {
        led_on = false;
        gpio_put(LED_PIN, false); // Desligar o LED
        led_active = false; // Marcar que o LED foi desligado
    }
    return true; // Continuar chamando o temporizador
}
int main() {
    // Inicializar comunicação padrão (para printf)
    stdio_init_all();
    // Configurar o pino 12 como saída (LED) e o pino 5 como entrada (botão)
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);
    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_up(BUTTON_PIN); // Habilitar o pull-up interno
    // Configurar um temporizador de repetição para chamar a função a cada 1 segundo
    struct repeating_timer timer;
    add_repeating_timer_ms(1000, repeating_timer_callback, NULL, &timer);
    while (true) {
        // Verificar se o botão foi pressionado (nível baixo indica pressão)
        if (gpio_get(BUTTON_PIN) == 0 && !led_active) {
            // Adicionar um pequeno debounce (aguardar para evitar falsos acionamentos)
            sleep_ms(50);
            if (gpio_get(BUTTON_PIN) == 0) {
                // Acender o LED
                led_on = true;
                gpio_put(LED_PIN, true); // Ligar o LED
                led_active = true;
                // Definir o tempo para desligar o LED após 2 segundos
                turn_off_time = make_timeout_time_ms(2000);
            }
        }
        sleep_ms(10); // Pequena pausa para reduzir o uso da CPU
    }
}
```

Fig. 9. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409325572648671233>

A imagem mostra um "Exemplo Prático 07" com uma simulação de hardware e código C. A estrutura é dividida em duas partes principais:

Diagrama à Esquerda (Simulação de Hardware): O diagrama apresenta um microcontrolador (provavelmente um Raspberry Pi Pico) conectado a um LED azul e a um botão. O LED está ligado a um pino com um resistor em série, e o botão está conectado a outro pino do microcontrolador. No topo do diagrama, há botões de controle da simulação: "play", "pause", e um contador indicando que a simulação está em andamento por mais de 38 segundos.

Abaixo do diagrama, há um link para a simulação rotulado como "Cap5_Ex7", sugerindo que este é o exercício 7 de um capítulo 5 de algum material didático.

Código à Direita (Em C):

O código controla um LED e um botão usando temporizadores e GPIO.

Definições de Pinos: Define LED_PIN para o LED e BUTTON_PIN para o botão.

Variáveis para controlar o estado do LED (led_on), se o LED foi ativado (led_active), e a hora absoluta para desligar o LED (absolute_time).

Função de Callback para Temporizador: repeating_timer_callback() é chamada periodicamente e verifica se o tempo atual ultrapassou o tempo limite para desligar o LED. Se sim, o LED é desligado e a variável led_active é definida como false.

Função Main: Inicialização: Configura a comunicação padrão (stdio_init_all()), configura o LED como saída e o botão como entrada com pull-up.

Um temporizador de repetição é adicionado para chamar repeating_timer_callback() a cada 1 segundo, gerenciado por add_repeating_timer_ms().

Loop Principal: Verifica se o botão foi pressionado e se o LED já estava desligado. Em caso afirmativo, liga o LED e define um tempo limite de 2 segundos para desligá-lo automaticamente. Adiciona um pequeno atraso (sleep_ms(10)) para debounce e evitar sobre carga de CPU.

O exemplo mostra como usar temporizadores para controlar o LED e realizar tarefas repetitivas em um microcontrolador.

Exemplo Prático 07 aplicado

Usar vídeo do documento
no lists

Exemplo Prático 08

Temporizador como contador

Simulation

The simulation interface shows a Raspberry Pi Pico W board. A red LED is connected to pin 13 through a resistor, and its other end is connected to ground. A pushbutton is connected between pin 5 and ground. The board also has a USB port, a DEBUG port, and a Wi-Fi module. The simulation window includes a header with a refresh icon, a square icon, a pause icon, a timer showing 09:46.288, and a battery icon at 1%.

```
#include "pico/stdlib.h"
#include "hardware/timer.h"
#include <stdio.h>
const uint LED_PIN = 13; // Pino do LED
const uint BUTTON_PIN = 5; // Pino do botão
volatile int button_press_count = 0; // Contador de pressões do botão
volatile bool led_on = false; // Estado do LED
volatile bool led_active = false; // Indica se o LED está aceso
int64_t turn_off_callback(alarm_id_t id, void *user_data) {
    gpio_put(LED_PIN, false); // Desligar o LED
    led_active = false;
    printf("LED desligado\n"); // Exibir mensagem quando o LED for desligado
    return 0; // Não repetir o alarme
}
bool repeating_timer_callback(struct repeating_timer *t) {
    static absolute_time_t last_press_time = 0;
    static bool button_last_state = false;
    bool button_pressed = !gpio_get(BUTTON_PIN); // Pressionado = LOW
    if (button_pressed && !button_last_state &&
        absolute_time_diff_us(last_press_time, get_absolute_time()) > 200000) { // 200 ms
        last_press_time = get_absolute_time();
        button_last_state = true;
        button_press_count++;
        if (button_press_count == 2) {
            gpio_put(LED_PIN, true);
            led_active = true;
            printf("LED ligado\n"); // Exibir mensagem quando o LED for ligado
            add_alarm_in_ms(4000, turn_off_callback, NULL, false);
            button_press_count = 0; // Resetar contador
        } else if (!button_pressed) {
            button_last_state = false; // Resetar estado quando o botão é liberado
        }
        return true; // Continuar o temporizador de repetição
    }
    int main() {
        stdio_init_all();
        // Configurar o pino do LED como saída
        gpio_init(LED_PIN);
        gpio_set_dir(LED_PIN, GPIO_OUT);
        gpio_put(LED_PIN, 0); // LED começa apagado
        // Configurar o pino do botão como entrada com pull-up interno
        gpio_init(BUTTON_PIN);
        gpio_set_dir(BUTTON_PIN, GPIO_IN);
        gpio_pull_up(BUTTON_PIN);
        // Configurar o temporizador para verificar o estado do botão a cada 100 ms
        struct repeating_timer timer;
        add_repeating_timer_ms(100, repeating_timer_callback, NULL, &timer);
        while (true) { // O loop principal fica livre para outras tarefas
            // O temporizador cuida do controle do botão e do LED
            tight_loop_contents();
        }
    }
}
```

Fig. 10. Exemplo Prático.

Fonte: <https://wokwi.com/projects/409698053693899777>

Descrição Geral:

Este código, escrito em linguagem de programação, controla um dispositivo eletrônico, possivelmente um microcontrolador, para funcionar como um temporizador. Imagine um relógio de cozinha que você usa para marcar o tempo enquanto cozinha. Este código faz algo parecido, mas em uma escala muito menor e com mais funcionalidades.

Elementos Principais e Funcionalidades:

Botão: O código detecta quando um botão físico é pressionado.

LED: Um LED (Diodo Emissor de Luz) é usado como indicador visual. Ele pode ser ligado ou desligado para mostrar diferentes estados do temporizador.

Temporizador: O coração do código. Ele conta o tempo em intervalos regulares e executa ações específicas quando um determinado tempo passa.

Contador: O temporizador funciona como um contador, aumentando seu valor a cada intervalo de tempo.
Simulação: O código pode ser simulado em um computador para visualizar o seu funcionamento sem precisar de hardware real.

Como Funciona (em termos gerais):

Inicialização: O código configura o botão, o LED e o temporizador para funcionarem corretamente.

Loop Principal: O código entra em um loop infinito, verificando constantemente se o botão foi pressionado.

Ação do Botão: Quando o botão é pressionado, o temporizador é iniciado ou reiniciado, e o LED muda de estado para indicar o início da contagem.

Contagem: O temporizador conta o tempo em intervalos regulares, e o valor do contador aumenta.

Condições de Parada: O código pode ser configurado para parar a contagem após um determinado tempo ou quando o botão é pressionado novamente.

Visualização: O LED pisca ou permanece aceso para indicar o estado do temporizador e o valor do contador.

Elementos Visuais na Simulação:

Placa: Uma representação visual da placa eletrônica com o botão, o LED e a conexão Wi-Fi.

Botão: Um botão que pode ser clicado na simulação para simular ação de pressionar o botão físico.

LED: Um LED que muda de cor ou estado para indicar o estado do temporizador.

Contador: Um display que mostra o valor atual do contador.

Exemplo Prático 08 aplicado

Usar vídeo do documento
no lists

Principais aspectos

- Clockesua importância
- Temporizador esua importância
- Plataforma BitDogLab
- Plataforma wokwi
- Exercícios práticos

Clock e Temporizadores

Unidade 4 | Capítulo 5