Anexo 8 - Temporizador de Sistema 2

# Descrição

Este experimento demonstra o uso de um Timer de Sistema. São utilizadas funções da biblioteca “pico/time.h”. O código irá controlar um LED, ligando-o, a partir de um evento externo (acionamento do botão) e, em seguida, adicionando um alarme a fim de temporizar o LED para desligá-lo automaticamente após o tempo definido.

# Objetivos:

- Demonstrar o uso de Timer de Sistema;  
- Utilizar verificações periódicas de tempo;  
- Utilizar alarmes;  
- Executar uma tarefa quando o tempo é alcançado.

# Código-fonte (main.c)

#include "pico/stdlib.h" // Biblioteca padrão do Raspberry Pi Pico para controle de GPIO, temporização e comunicação serial.

#include "pico/time.h" // Biblioteca para gerenciamento de temporizadores e alarmes.

const uint LED\_PIN = 11; // Define o pino GPIO 11 para controlar o LED.

const uint BUTTON\_PIN = 5; // Define o pino GPIO 5 para ler o estado do botão.

bool led\_on = false; // Variável global para armazenar o estado do LED (não utilizada neste código).

bool led\_active = false; // Indica se o LED está atualmente aceso (para evitar múltiplas ativações).

// Função de callback para desligar o LED após o tempo programado.

int64\_t turn\_off\_callback(alarm\_id\_t id, void \*user\_data) {

// Desliga o LED configurando o pino LED\_PIN para nível baixo.

gpio\_put(LED\_PIN, false);

// Atualiza o estado de 'led\_active' para falso, indicando que o LED está desligado.

led\_active = false;

// Retorna 0 para indicar que o alarme não deve se repetir.

return 0;

}

int main() {

// Inicializa a comunicação serial para permitir o uso de printf.

// Isso é útil para depuração, embora não seja usado neste exemplo.

stdio\_init\_all();

// Configura o pino LED\_PIN (11) como saída digital.

gpio\_init(LED\_PIN);

gpio\_set\_dir(LED\_PIN, GPIO\_OUT);

// Configura o pino BUTTON\_PIN (5) como entrada digital.

gpio\_init(BUTTON\_PIN);

gpio\_set\_dir(BUTTON\_PIN, GPIO\_IN);

// Habilita o resistor pull-up interno para o pino do botão.

// Isso garante que o pino seja lido como alto (3,3 V) quando o botão não está pressionado.

gpio\_pull\_up(BUTTON\_PIN);

// Loop principal do programa que verifica continuamente o estado do botão.

while (true) {

// Verifica se o botão foi pressionado (nível baixo no pino) e se o LED não está ativo.

if (gpio\_get(BUTTON\_PIN) == 0 && !led\_active) {

// Adiciona um pequeno atraso para debounce, evitando leituras errôneas.

sleep\_ms(50);

// Verifica novamente o estado do botão após o debounce.

if (gpio\_get(BUTTON\_PIN) == 0) {

// Liga o LED configurando o pino LED\_PIN para nível alto.

gpio\_put(LED\_PIN, true);

// Define 'led\_active' como true para indicar que o LED está aceso.

led\_active = true;

// Agenda um alarme para desligar o LED após 3 segundos (3000 ms).

// A função 'turn\_off\_callback' será chamada após esse tempo.

add\_alarm\_in\_ms(3000, turn\_off\_callback, NULL, false);

}

}

// Introduz uma pequena pausa de 10 ms para reduzir o uso da CPU.

// Isso evita que o loop seja executado muito rapidamente e consuma recursos desnecessários.

sleep\_ms(10);

}

// Retorno de 0, que nunca será alcançado devido ao loop infinito.

// Isso é apenas uma boa prática em programas com um ponto de entrada main().

return 0;

}

# Observações Práticas

- Incluir a Biblioteca “pico/time.h” no CMakeList.txt, como indicado abaixo e, em seguida, salvar o arquivo CMakeList.txt:

