Anexo 9 – Temporizador de Hardware 2

# Descrição

Este experimento demonstra o uso de um Timer de Hardware. São utilizadas funções da biblioteca “hardware/timer.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.

# Objetivo

- Demonstrar o uso de Timer de Hardware;  
- 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 para controle de GPIO, temporização e comunicação serial.

#include "hardware/timer.h" // Biblioteca para gerenciamento de temporizadores de hardware.

// Definição dos pinos para o LED e o botão

const uint LED\_PIN = 12; // Define o pino GPIO 12 para o LED.

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

// Variáveis de controle

bool led\_on = false; // Armazena o estado atual do LED (ligado/desligado).

absolute\_time\_t turn\_off\_time; // Armazena o tempo em que o LED deve ser desligado.

bool led\_active = false; // Indica se o LED está ativo, para evitar reativação durante o tempo de espera.

// Função de callback que será chamada pelo temporizador repetidamente a cada 1 segundo.

bool repeating\_timer\_callback(struct repeating\_timer \*t) {

// Verifica se o LED deve ser desligado:

// Se o LED está ativo e o tempo atual é maior ou igual ao tempo de desligamento, desliga o LED.

if (led\_active && absolute\_time\_diff\_us(get\_absolute\_time(), turn\_off\_time) <= 0) {

// Define o estado do LED como desligado e atualiza a variável de controle.

led\_on = false;

gpio\_put(LED\_PIN, false); // Desliga o LED.

led\_active = false; // Atualiza a variável para indicar que o LED foi desligado.

}

// Retorna true para manter o temporizador repetindo esta função de callback.

return true;

}

int main() {

// Inicializa a comunicação padrão, permitindo o uso de printf para depuração.

// Útil para enviar mensagens pela interface serial (USB ou UART).

stdio\_init\_all();

// Inicializa e configura o pino do LED (GPIO 12) como saída.

gpio\_init(LED\_PIN);

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

// Inicializa e configura o pino do botão (GPIO 5) como entrada.

gpio\_init(BUTTON\_PIN);

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

// Habilita o resistor de pull-up interno no 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);

// Configura um temporizador repetitivo que chama a função de callback a cada 1 segundo (1000 ms).

struct repeating\_timer timer;

add\_repeating\_timer\_ms(1000, repeating\_timer\_callback, NULL, &timer);

// Loop principal do programa para monitorar o estado do botão e controlar o LED.

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 (aguarda 50 ms para evitar leituras errôneas).

sleep\_ms(50);

// Verifica novamente o estado do botão após o debounce para confirmar a pressão.

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

// Acende o LED e atualiza o estado e as variáveis de controle.

led\_on = true;

gpio\_put(LED\_PIN, true); // Liga o LED.

led\_active = true;

// Define o tempo para desligar o LED após 2 segundos (2000 ms).

// `make\_timeout\_time\_ms` calcula um tempo futuro a partir do tempo atual.

turn\_off\_time = make\_timeout\_time\_ms(2000);

}

}

// Pequena pausa de 10 ms para reduzir o uso da CPU e evitar execução excessivamente rápida do loop.

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 “hardware/timer.h” no CMakeList.txt, como indicado abaixo e, em seguida, salvar o arquivo CMakeList.txt:

