Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade – Referente ao capítulo 04 da unidade 03.

Tema do Capítulo – Sistemas de Tempo Real: Conceitos Básicos.

Prazo dia 29/06/2025 as 23:59

Enunciado: Semáforo Didático com FreeRTOS: LED RGB Controlado por Tarefas.

Este projeto tem como objetivo o desenvolvimento de uma aplicação embarcada utilizando a placa BitDogLab, em conjunto com o sistema operacional de tempo real FreeRTOS. A proposta consiste em simular o funcionamento de um semáforo, empregando um LED RGB para representar as cores indicativas, controladas por meio de tarefas executadas de forma sequencial e com temporização definida. A implementação deverá respeitar a lógica convencional de um semáforo, A alternando entre as cores vermelho, verde e amarelo, com os seguintes tempos de exibição: vermelho 5 s, verde: 5 s e amarelo 3 segundos. A cor amarela deverá ser simulada por meio da fusão dos canais vermelho e verde do LED RGB, ou seja, acionando simultaneamente os dois pinos correspondentes. O aluno é responsável por definir a melhor estratégia para essa fusão, de modo a garantir uma representação visual clara e eficiente da cor amarela.

Descrição do Sistema:

1) Configure os pinos GPIO11 (verde), GPIO12 (azul) e GPIO13 (vermelho) como saídas digitais.
2) Certifique-se de desligar os três canais no início da execução para evitar estados indesejados.
3) Implemente uma tarefa no FreeRTOS que controle a alternância das cores do LED RGB com base nos tempos definidos.
4) A sequência de execução deve ser: vermelho → verde → amarelo → repetir.

Links das simulações

- Nome do arquivo principal: “main.c”.
- Link do Projeto completo com a pasta build: Atividade_Uni_03_Cap_01.zip
- Link do Vídeo: https://youtu.be/V2jRphEfOw8
- GitHub: https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_03/Cap_01/Atividade_01

Como entender o projeto:

1. Comece pela função “int main()” em main.c para ver a ordem em que tudo acontece.
2. Vá abrindo os headers.h para saber o que cada módulo oferece.
3. Leia os .c se quiser entender os detalhes internos.
4. Todos os arquivos estão comentados com Doxygen em português para facilitar.

- Código: main.c

```
/**
 * @file main.c
 * @author Manoel Felipe Costa Furtado
 * @brief Implementação de um semáforo de trânsito didático utilizando FreeRTOS na plataforma Raspberry Pi Pico.
 * @version 1.0
 * @date 2025-06-22
 * @copyright 2025 Manoel Furtado (MIT License) (veja LICENSE.md)
 * =======================================================
 * Projeto: Semáforo Didático com FreeRTOS e BitDogLab
 * =======================================================
 *
 * ☑ Objetivo:
 * Desenvolver uma aplicação embarcada em tempo real utilizando a plataforma BitDogLab e o sistema operacional
FreeRTOS.
 * O objetivo é controlar um LED RGB para simular o comportamento de um semáforo, com transições de cores e
temporizações específicas.
 * Este projeto demonstra o uso de múltiplas tarefas e semáforos para sincronização no FreeRTOS. Três tarefas
concorrentes, cada uma
 * controlando uma cor do LED RGB (vermelho, verde, amarelo), são sincronizadas para executar em uma sequência
específica, simulando um semáforo de trânsito.
 * A medição do tempo de execução de trechos críticos de cada tarefa também é implementada para análise de
performance.
 *
 * ☑ O que o código faz:
```

```
 * - Cria 3 tarefas concorrentes, uma para cada cor do semáforo.
 * - Todas as tarefas são criadas com a mesma prioridade.
 * - Utiliza 3 semáforos binários para garantir a ordem de execução:
 *   Vermelho -> Verde -> Amarelo -> (repete).
 * - Uma tarefa só é executada após ser sinalizada (liberada) pela tarefa anterior.
 *
 * ☑ Estrutura de Sincronização:
 * 1. tarefa_vermelho: Espera seu sinal. Ao receber, liga o LED vermelho por 5s
 *    e depois libera o sinal para a tarefa_verde.
 * 2. tarefa_verde: Espera seu sinal. Ao receber, liga o LED verde por 5s
 *    e depois libera o sinal para a tarefa_amarelo.
 * 3. tarefa_amarelo: Espera seu sinal. Ao receber, liga o LED amarelo por 3s
 *    e depois libera o sinal para a tarefa_vermelho, reiniciando o ciclo.
 *
 * ☑ Recursos utilizados:
 * - Sistema operacional de tempo real FreeRTOS.
 * - Funções de temporização do FreeRTOS (vTaskDelay).
 * - GPIOs digitais para controle do LED RGB.
 * - Comunicação serial (USB/UART) para exibir o estado atual do semáforo.
 *
 * ☑ Lógica de Cores:
 * - Vermelho: Pino GPIO 13 (RED) ativado.
 * - Verde: Pino GPIO 11 (GREEN) ativado.
 * - Amarelo: Pinos GPIO 13 (RED) e GPIO 11 (GREEN) ativados simultaneamente.
 * - O pino GPIO 12 (BLUE) permanece sempre desligado.
 *
 * * ☑ Estrutura Modular do Código:
 * O código está dividido em três partes claras:
 * 1. inicializar_hardware_semaforo(): Prepara e configura os pinos GPIO do LED.
 * 2. 3 tarefas dos semaforos: Contém a lógica de execução do semáforo (a máquina de estados).
 * 3. main(): Ponto de entrada que inicializa o sistema, o hardware e o escalonador de tarefas.
 *
 */

//=============================================================================================
// Inclusão de Bibliotecas (Headers)
//=============================================================================================

#include "pico/stdlib.h"    ///< Inclui as bibliotecas padrão do SDK do Raspberry Pi Pico para funções essenciais
como GPIO, UART, etc.
#include "FreeRTOS.h"       ///< Inclui o header principal do FreeRTOS, com as definições do Kernel.
#include "task.h"           ///< Inclui as funções de gerenciamento de tarefas do FreeRTOS (ex: xTaskCreate,
vTaskDelay).
#include "semphr.h"         ///< Inclui as funções de gerenciamento de semáforos do FreeRTOS (ex:
xSemaphoreCreateBinary, xSemaphoreTake).
#include <stdio.h>          ///< Inclui a biblioteca padrão de entrada e saída do C, usada aqui para a função
printf().
```

```
//=============================================================================================
// Definições e Variáveis Globais
//=============================================================================================

// --- Definição dos Pinos do LED RGB ---
const uint LED_RED_PIN = 13;    ///< Define o pino GPIO conectado ao canal Vermelho do LED RGB.
const uint LED_GREEN_PIN = 11;  ///< Define o pino GPIO conectado ao canal Verde do LED RGB.
const uint LED_BLUE_PIN = 12;   ///< Define o pino GPIO conectado ao canal Azul do LED RGB.

// --- Declaração dos Handles dos Semáforos ---
// Um "handle" é como um ponteiro ou uma "chave de acesso" para um objeto criado pelo FreeRTOS.
// Precisamos guardar esses handles em variáveis globais para que tanto a 'main' (que os cria)
// quanto as tarefas (que os utilizam) possam acessar os mesmos semáforos.
SemaphoreHandle_t semaforo_sinal_vermelho;  ///< Handle para o semáforo que controla a execução da tarefa vermelha.
SemaphoreHandle_t semaforo_sinal_verde;     ///< Handle para o semáforo que controla a execução da tarefa verde.
SemaphoreHandle_t semaforo_sinal_amarelo;   ///< Handle para o semáforo que controla a execução da tarefa amarela.

//=============================================================================================
// Seção de Funções
//=============================================================================================



/*
 * ================================
 * 1. Inicialização de Hardware
 * ================================
 */
/**
 * @brief Configura os pinos GPIO para o controle do LED RGB.
 * @note  Esta função deve ser chamada uma única vez no início do programa, antes da criação das tarefas
 * que utilizarão os pinos. Ela centraliza toda a inicialização de hardware.
 */
void inicializar_hardware_semaforo(void) {
    /// Inicializa os metadados de cada pino para que o SDK saiba que vamos utilizá-los.
    gpio_init(LED_RED_PIN);
    gpio_init(LED_GREEN_PIN);
    gpio_init(LED_BLUE_PIN);

    /// Define a direção de cada pino como SAÍDA (Output), permitindo que o microcontrolador envie sinais (0V ou
3.3V) para o LED.
    gpio_set_dir(LED_RED_PIN, GPIO_OUT);
    gpio_set_dir(LED_GREEN_PIN, GPIO_OUT);
    gpio_set_dir(LED_BLUE_PIN, GPIO_OUT);

    /// Garante um estado inicial seguro e conhecido, desligando todos os canais do LED.
    /// Isso evita que o LED fique aceso com uma cor inesperada ao ligar o dispositivo.
    gpio_put(LED_RED_PIN, 0);   // Nível lógico baixo (0V) desliga o LED.
    gpio_put(LED_GREEN_PIN, 0); // Nível lógico baixo (0V) desliga o LED.
    gpio_put(LED_BLUE_PIN, 0);  // Nível lógico baixo (0V) desliga o LED.
```

```c
}

/*
 * ================================
 * 2. Definição das Tarefas
 * ================================
 */
/**
 * @brief Tarefa responsável por controlar o estado "Vermelho" do semáforo.
 * @param params Ponteiro para parâmetros da tarefa (não utilizado neste projeto).
 * @note Esta tarefa entra em um loop infinito e só executa sua lógica após receber o sinal
 * do seu respectivo semáforo. Ao final, ela sinaliza a próxima tarefa do ciclo.
 */
void tarefa_vermelho(void *params) {
    /// O loop infinito é o padrão para tarefas no FreeRTOS. A tarefa nunca deve "retornar" ou sair.
    while (true) {
        /// A função xSemaphoreTake "tenta pegar" o semáforo. Como o segundo parâmetro é portMAX_DELAY,
        /// a tarefa ficará BLOQUEADA (sem consumir CPU) aqui indefinidamente, até que outra tarefa
        /// libere este semáforo usando xSemaphoreGive.
        xSemaphoreTake(semaforo_sinal_vermelho, portMAX_DELAY);

        // --- Início da seção crítica e medição de tempo ---
        absolute_time_t start = get_absolute_time();  /// Marca o tempo de início com alta precisão, usando o timer
de hardware do Pico.

        printf("Semaforo: VERMELHO\n");
        gpio_put(LED_RED_PIN, 1);   /// Liga o pino do LED vermelho.
        gpio_put(LED_GREEN_PIN, 0); /// Garante que o pino verde esteja desligado.
        gpio_put(LED_BLUE_PIN, 0);  /// Garante que o pino azul esteja desligado.

        absolute_time_t end = get_absolute_time();  /// Marca o tempo de fim.
        int64_t exec_time_us = absolute_time_diff_us(start, end);  /// Calcula a diferença em microssegundos.

        /// Exibe o tempo que o processador levou para executar o bloco de código acima.
        /// Como não há outras tarefas prontas para executar neste instante (devido à sincronização),
        /// este valor é uma boa aproximação do tempo de CPU consumido.
        printf("Tarefa VERMELHO - Tempo de CPU: %lld us\n\n", exec_time_us);
        // --- Fim da seção crítica ---

        /// A função vTaskDelay põe a tarefa no estado BLOQUEADO por um tempo determinado.
        /// Durante esses 5000ms (5s), outras tarefas de menor ou igual prioridade poderão executar.
        /// A macro pdMS_TO_TICKS converte milissegundos para "ticks" do sistema, a unidade de tempo do FreeRTOS.
        vTaskDelay(pdMS_TO_TICKS(5000));

        /// Libera o semáforo da próxima tarefa na sequência (Verde), permitindo que ela saia do estado
        /// de bloqueio e comece a sua execução. É como "passar o bastão" em uma corrida de revezamento.
        xSemaphoreGive(semaforo_sinal_verde);
    }
```

```c
}

/**
 * @brief Tarefa responsável por controlar o estado "Verde" do semáforo.
 * @param params Ponteiro para parâmetros da tarefa (não utilizado neste projeto).
 * @note Segue a mesma lógica da tarefa_vermelho, aguardando seu sinal e, ao final, sinalizando a tarefa_amarelo.
 */
void tarefa_verde(void *params) {
    while (true) {
        /// Bloqueia a tarefa até que a tarefa_vermelho chame xSemaphoreGive(semaforo_sinal_verde).
        xSemaphoreTake(semaforo_sinal_verde, portMAX_DELAY);

        absolute_time_t start = get_absolute_time();
        printf("Semaforo: VERDE\n");
        gpio_put(LED_RED_PIN, 0);
        gpio_put(LED_GREEN_PIN, 1);
        absolute_time_t end = get_absolute_time();
        int64_t exec_time_us = absolute_time_diff_us(start, end);

        printf("Tarefa VERDE - Tempo de CPU: %lld us\n\n", exec_time_us);

        /// Bloqueia por 5 segundos.
        vTaskDelay(pdMS_TO_TICKS(5000));

        /// Passa o "bastão" para a tarefa do semáforo amarelo.
        xSemaphoreGive(semaforo_sinal_amarelo);
    }
}

/**
 * @brief Tarefa responsável por controlar o estado "Amarelo" do semáforo.
 * @param params Ponteiro para parâmetros da tarefa (não utilizado neste projeto).
 * @note Esta tarefa simula a cor amarela ligando os LEDs vermelho e verde simultaneamente.
 * Ao final, ela sinaliza a tarefa_vermelho, fechando o ciclo.
 */
void tarefa_amarelo(void *params) {
    while (true) {
        /// Bloqueia a tarefa até que a tarefa_verde chame xSemaphoreGive(semaforo_sinal_amarelo).
        xSemaphoreTake(semaforo_sinal_amarelo, portMAX_DELAY);

        absolute_time_t start = get_absolute_time();
        printf("Semaforo: AMARELO\n");
        gpio_put(LED_RED_PIN, 1);   /// Ativa o vermelho para a cor amarela.
        gpio_put(LED_GREEN_PIN, 1); /// Ativa o verde para a cor amarela.
        absolute_time_t end = get_absolute_time();
        int64_t exec_time_us = absolute_time_diff_us(start, end);

        printf("Tarefa AMARELO - Tempo de CPU: %lld us\n\n", exec_time_us);
```

```
        /// Bloqueia por 3 segundos.
        vTaskDelay(pdMS_TO_TICKS(3000));

        /// Libera a tarefa do semáforo vermelho, reiniciando todo o ciclo.
        xSemaphoreGive(semaforo_sinal_vermelho);
    }
}

/*
 * ================================
 * 3. Função Principal (main)
 * ================================
 */
/**
 * @brief Função principal do programa (ponto de entrada).
 * @return int Código de retorno (não aplicável em sistemas embarcados).
 * @note Esta função é responsável por toda a inicialização do sistema. Após chamar vTaskStartScheduler(),
 * o controle é permanentemente transferido para o escalonador do FreeRTOS e esta função
 * efetivamente nunca mais é executada.
 */
int main() {
    /// Inicializa todas as interfaces de I/O padrão, principalmente a comunicação serial via USB
    /// para que a função printf() funcione e envie mensagens para o computador.
    stdio_init_all();

    /// Uma pausa simples, usando uma função do SDK do Pico. Útil para dar tempo ao monitor serial
    /// de se conectar ao dispositivo após a programação, garantindo que as primeiras mensagens não sejam perdidas.
    /// IMPORTANTE: sleep_ms() é uma espera "bloqueante" (busy-waiting) e não deve ser usada dentro das tarefas do
FreeRTOS.
    sleep_ms(2000);
    printf("Inicializando sistema de semaforo com FreeRTOS...\n");

    /// Chama a função dedicada para configurar o hardware.
    printf("Configurando hardware (GPIOs)...\n");
    inicializar_hardware_semaforo();

    /// Cria os semáforos binários. Eles são criados no estado "vazio" (ou "tomado"),
    /// ou seja, uma chamada a xSemaphoreTake bloquearia imediatamente.
    semaforo_sinal_vermelho = xSemaphoreCreateBinary();
    semaforo_sinal_verde = xSemaphoreCreateBinary();
    semaforo_sinal_amarelo = xSemaphoreCreateBinary();

    /// Cria as 3 tarefas que executarão concorrentemente.
    printf("Criando tarefas...\n");
    xTaskCreate(tarefa_vermelho,    // Ponteiro para a função da tarefa a ser executada.
                "VermelhoTask",     // Nome de texto da tarefa, útil para depuração.
```

```
                256,                // Tamanho da pilha (stack) em PALAVRAS (1 palavra = 4 bytes no Pico). 256*4 =
1024 bytes.
                NULL,               // Parâmetro a ser passado para a tarefa (nenhum neste caso).
                1,                  // Prioridade da tarefa (0 é a menor, números maiores indicam maior prioridade).
                NULL                // Handle da tarefa (não precisamos guardar, então é NULL).
            );

    xTaskCreate(tarefa_verde,    // Ponteiro para a função da tarefa
                "VerdeTask",     // Nome da tarefa (para debug)
                256,             // Tamanho da pilha (stack) em palavras
                NULL,            // Parâmetros da tarefa (não utilizados)
                1,               // Prioridade da tarefa
                NULL             // Handle da tarefa (não utilizado)
            );

    xTaskCreate(tarefa_amarelo, // Ponteiro para a função da tarefa
                "AmareloTask",  // Nome da tarefa (para debug)
                256,             // Tamanho da pilha (stack) em palavras
                NULL,            // Parâmetros da tarefa (não utilizados)
                1,               // Prioridade da tarefa
                NULL             // Handle da tarefa (não utilizado)
            );

    /// "Dá o pontapé inicial" liberando o semáforo da primeira tarefa (vermelho).
    /// Sem esta linha, todas as três tarefas ficariam bloqueadas para sempre em xSemaphoreTake e o sistema não faria
nada.
    printf("Iniciando ciclo do semaforo...\n");
    xSemaphoreGive(semaforo_sinal_vermelho);

    /// Inicia o escalonador do FreeRTOS.
    /// A partir deste ponto, o FreeRTOS toma controle total do processador e começa a gerenciar a execução das
tarefas.
    /// Esta função nunca retorna.
    printf("Iniciando escalonador do FreeRTOS.\n");
    vTaskStartScheduler();

    /// O código abaixo nunca deve ser alcançado se o escalonador iniciar corretamente.
    /// É um loop de segurança, caso haja algum problema (ex: falta de memória para a tarefa Idle).
    while (true) {}
}
```