Aluno do Embarcatech_37 no IFMA

Nome: Manoel Felipe Costa Furtado

Matrícula: 20251RSE.MTC0086

Atividade – Referente ao capítulo 03 da unidade 02

Tema do Capítulo - MQTT, CoAP, AMQP, STOMP

Prazo dia 15/06/2025 as 23:59

Enunciado: Complementação da Atividade_3_MQTT_2. Perceba que, ao ocorrer o envio do PING, o LED RGB permanece aceso na cor verde.

Qual a melhoria que deve ser realizada no novo projeto:

- Alterar a função **void tratar_mensagem(MensagemWifi msg)** de modo que seja realizada as seguites tarefas:
- Logo após ocorrer o PING o LED RGB deve mudar para uma cor aleatória que não seja o VERDE, padrão já definido. Na função, percebemos que ela ativa o LED para VERDE. Que o LED RGB permaneça na cor aleatória por 1 segundo, após ocorrer o PING, sinalizando no LED que houve o PING.
- Funções que ajudam na definição dos números aleatórios, já apresentadas no arquivo funcoes_neopixel.c das atividades do foreground, que permitem mudar a cor de forma aleatória:

- **void inicializar_aleatorio()** -> somente deve ser executada uma única vez.

- **void numero_aleatorio(int min, int max)** -> gera um número aleatório no intervalo de [min,max]

- Nome do arquivo principal: “Atividade_Uni_02_Cap_03.c” → Na pasta src.
- O arquivo “Atividade_Uni_02_Cap_03_auxiliar.c” → Também é importante.
- Vídeo mostrando o seu funcionamento
  Link: https://youtu.be/KfTWtlBBni0
- GitHub:
  https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Unidade_03/Cap_03/Atividade_03

Organização do projeto: Código foi modularizado.

Na Pasta src:

Arquivo / Módulo	Descrição da Responsabilidade
Atividade_Uni_02_Cap_03.c	Arquivo Principal (Núcleo 0): Orquestra o sistema, gerencia o loop de eventos, inicializa o hardware e o Núcleo 1.
Atividade_Uni_02_Cap_03_ auxiliar.c	Funções de Suporte: Contém a lógica auxiliar, como tratamento de mensagens, controle do LED e geração de números aleatórios.
configura_geral.h	Configuração Central: Armazena todas as constantes e definições do projeto (pinos, credenciais de rede, temporizações).
estado_mqtt.c / estado_mqtt.h	Estado Compartilhado: Define e declara variáveis globais para permitir a comunicação e sincronização de estado entre módulos.
funcao_wifi_nucleo1 (Referenciado)	Conectividade (Núcleo 1): Código dedicado que roda no Núcleo 1, responsável por gerenciar Wi-Fi e MQTT de baixo nível.
Bibliotecas (fila_circular, etc.)	Abstrações: Módulos que fornecem funcionalidades específicas, como a fila de mensagens e o controle de periféricos.

Na Pasta libs/OLED_:

Arquivo / Módulo	Descrição da Responsabilidade
ssd1306_font.h	Fonte Gráfica: Fornece os dados brutos (bitmaps) para cada caractere que pode ser desenhado na tela. É o nível mais baixo da abstração de texto.
ssd1306_i2c.c / ssd1306_i2c.h	Driver Principal (Baixo Nível): Implementa a comunicação direta via I^2C com o controlador SSD1306. Contém os comandos do hardware e as funções fundamentais para desenhar pixels, linhas e caracteres, utilizando a fonte do ssd1306_font.h.
ssd1306.h	Interface Pública (API): Atua como a interface pública unificada para o driver SSD1306. Agrupa as declarações de todas as funções de desenho e controle, facilitando a inclusão em outros módulos.
oled_utils.c / oled_utils.h	Utilitários Genéricos: Oferece funções de conveniência de nível mais alto, como uma rotina de inicialização parametrizada (setup_oled) e uma função para limpar a tela (oled_clear).
setup_oled.c / setup_oled.h	Configuração Específica do Projeto: Utiliza as funções do driver para inicializar o display com os pinos e configurações exatas definidas para a aplicação. É a rotina que efetivamente prepara o hardware para o seu projeto.
display.c / display.h	Interface do Usuário (Alto Nível): Provê uma função de conveniência para a aplicação (exibir_e_esperar), permitindo

	mostrar uma mensagem temporária na tela de forma simples, como um status de conexão ou um erro.

Na Pasta libs/WIFI_:

Arquivo / Módulo	Descrição da Responsabilidade
lwipopts.h	Configuração da Pilha de Rede (LWIP): Arquivo de configuração que personaliza o funcionamento da pilha TCP/IP (LWIP), ajustando o uso de memória e ativando/desativando protocolos (DHCP, DNS, TCP, etc.) para otimizar o desempenho no Pico.
conexao.c / conexao.h	Gerenciador de Rede (Núcleo 1): Contém a lógica que roda no segundo núcleo, responsável por conectar-se ao Wi-Fi, monitorar a conexão e tentar reconectar automaticamente em caso de falha. Comunica o status da rede e o IP para o Núcleo 0 via FIFO.
fila_circular.c / fila_circular.h	Fila de Mensagens Segura: Implementa uma fila circular protegida por mutex para armazenar as mensagens de status enviadas pelo Núcleo 1. Permite que o Núcleo 0 processe as mensagens de forma segura e desacoplada do Núcleo 1.
mqtt_lwip.c / mqtt_lwip.h	Cliente MQTT (Núcleo 0): Gerencia toda a lógica do cliente MQTT, como a conexão com o broker, a publicação de mensagens e o tratamento de callbacks (respostas do broker). É ativado pelo Núcleo 0 após a rede estar pronta.
display_utils.h, wifi_status.h, MQTTPlatform.h	Arquivos de Cabeçalho Auxiliares: Pequenos arquivos que declaram funções utilitárias (display_utils.h), variáveis de estado compartilhadas (wifi_status.h), ou adaptam bibliotecas para a plataforma específica (MQTTPlatform.h).
rgb_pwm_control.c / rgb_pwm_control.h	Controle do LED RGB via PWM: Este módulo abstrai o controle de um LED RGB. A função init_rgb_pwm configura os pinos GPIO para a função de PWM e inicializa o hardware de PWM para cada canal de cor. A função set_rgb_pwm permite definir uma cor específica, ajustando a intensidade (ciclo de trabalho) de cada canal (vermelho, verde e azul).

Benefícios desta abordagem:

- Main enxuta: A função principal fica clara e fácil de entender.
- Modularidade: Cada funcionalidade está separada em seu próprio módulo.
- Reusabilidade: Os módulos podem ser usados em outros projetos.
- Manutenibilidade: Fácil de fazer alterações em áreas específicas.
- Testabilidade: Cada módulo pode ser testado isoladamente.

Como entender o projeto:

1. Comece pelo main() em Atividade_Uni_02_Cap_03.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.

Nesse trabalho, foi feito poucas mudanças:

- Basicamente, em Atividade_Uni_02_Cap_03.c: As mudanças foram necessárias para habilitar a nova funcionalidade de geração de cores aleatórias.

extern void inicializar_aleatorio(void); // <-- ESTA LINHA FOI ADICIONADA

em inicia_hardware(), adicionei a chamada para inicializar_aleatorio()

- Em Atividade_Uni_02_Cap_03_auxiliar.c:

void tratar_mensagem(MensagemWiFi msg) { ...

```c
// --- INÍCIO DA MELHORIA IMPLEMENTADA ---
uint16_t r, g, b; // Variáveis para armazenar os componentes de cor (Vermelho, Verde, Azul).

// 1. Gera uma cor aleatória, garantindo que não seja um verde forte.
do {
    r = numero_aleatorio(0, 65535); // Gera valor para vermelho (0 a 100% do PWM).
    g = numero_aleatorio(0, 65535); // Gera valor para verde.
    b = numero_aleatorio(0, 65535); // Gera valor para azul.
} while (g > r && g > b && g > 32768); // A condição impede cores onde o verde é o componente mais forte e brilhante.

// 2. Define a cor aleatória gerada no LED para sinalizar visualmente o PING.
set_rgb_pwm(r, g, b);
render_on_display(buffer_oled, &area); // Atualiza o OLED junto com a mudança de cor do LED.

// 3. Mantém a cor aleatória visível por 1 segundo.
sleep_ms(1000);

// 4. Retorna o LED para a cor verde padrão, indicando que a conexão continua OK.
set_rgb_pwm(0, 65535, 0); // Verde sólido.
// --- FIM DA MELHORIA IMPLEMENTADA ---
```

...

- Código: src/Atividade_Uni_02_Cap_03.c

```c
/**
 * @file Atividade_Uni_02_Cap_03.c
 * @author Modificado Por Manoel Furtado
 * @version 1.2
 * @date 15 de junho de 2025
 * @brief Ponto de entrada principal do Núcleo 0 (Core 0) para o projeto com MQTT.
 *
 * @details
 * Este arquivo contém a lógica principal que roda no primeiro núcleo do microcontrolador
 * RP2040. Ele atua como o orquestrador geral do sistema, gerenciando a interface com
 * o usuário, a inicialização de hardware e a comunicação com o Núcleo 1.
```

```
 *
 * Suas responsabilidades são:
 * 1.  Inicialização do Sistema: Configura periféricos como o display OLED, o LED RGB (via PWM),
 *                               a comunicação serial e o gerador de números aleatórios.
 *
 * 2.  Lançamento do Núcleo 1: Inicia o segundo núcleo, que é dedicado às tarefas de rede
 *                             (Wi-Fi e operações de baixo nível do MQTT).
 *
 * 3.  Comunicação Inter-Core: Entra em um loop de eventos infinito onde monitora a FIFO (um buffer de
 *                             comunicação entre os núcleos) para receber mensagens do Núcleo 1.
 *
 * As mensagens podem ser status da conexão Wi-Fi, um endereço IP ou confirmações de PING.
 * 4.  Gerenciamento de Estado MQTT: Implementa uma máquina de estados simples que, após receber um
 *                                   endereço IP válido do Núcleo 1, inicia o cliente MQTT.
 *
 * 5.  Lógica de Aplicação: Utiliza um temporizador não-bloqueante para enviar periodicamente uma
 *                          mensagem "PING" via MQTT, verificando a conectividade de ponta a ponta.
 *
 * 6.  Tratamento de Mensagens: Utiliza uma fila circular local para armazenar mensagens recebidas
 *                              do Núcleo 1, desacoplando o recebimento do processamento e
 *                              mantendo o loop principal responsivo.
 */

// ============================================================================
// Includes de Bibliotecas
// ============================================================================
#include "fila_circular.h"      // Estrutura de dados para a fila de mensagens do Wi-Fi.
#include "rgb_pwm_control.h"    // Funções para controlar o LED RGB com PWM.
#include "configura_geral.h"    // Arquivo de configuração central com pinos, senhas, etc.
#include "oled_utils.h"         // Funções utilitárias para o display OLED.
#include "ssd1306_i2c.h"        // Driver de baixo nível para o display OLED SSD1306.
#include "mqtt_lwip.h"          // Funções relacionadas ao cliente MQTT.
#include "lwip/ip_addr.h"       // Para manipulação de endereços IP da stack lwIP.
#include "pico/multicore.h"     // Funções da SDK para gerenciamento dos dois núcleos.
#include <stdio.h>              // Biblioteca padrão de entrada/saída para printf.
#include "estado_mqtt.h"        // Variáveis globais compartilhadas (estado do sistema).

// ============================================================================
// Protótipos de Funções Externas
// ============================================================================
// A palavra-chave 'extern' indica que estas funções estão definidas em outro arquivo.
extern void funcao_wifi_nucleo1(void);      // Ponto de entrada do código que rodará no Núcleo 1.
extern void espera_usb();                   // Função de espera definida em Atividade_Uni_02_Cap_03_auxiliar.c.
extern void tratar_ip_binario(uint32_t ip_bin); // Função para processar o IP, definida no arquivo auxiliar.
extern void tratar_mensagem(MensagemWiFi msg);  // Função principal de tratamento de mensagens, no arquivo auxiliar.
extern void inicializar_aleatorio(void);    // Função para inicializar o gerador de números aleatórios, no arquivo
auxiliar.
```

```c
// ============================================================================
// Protótipos de Funções Locais
// ============================================================================
// Funções definidas e utilizadas apenas dentro deste arquivo.
void inicia_hardware();
void inicia_core1();
void verificar_fifo(void);
void tratar_fila(void);
void inicializar_mqtt_se_preciso(void);
void enviar_ping_periodico(void);

// ============================================================================
// Variáveis Globais do Arquivo
// ============================================================================
FilaCircular fila_wifi;         /**< Fila circular local para armazenar mensagens recebidas do Núcleo 1 antes do processamento. */
absolute_time_t proximo_envio;  /**< Armazena o timestamp do próximo envio de PING MQTT para controle de tempo não-bloqueante. */
char mensagem_str[50];          /**< Buffer de string reutilizável para formatar mensagens de depuração via printf. */
bool ip_recebido = false;       /**< Flag local que se torna verdadeiro após o recebimento do IP, usado como condição. */

/**
 * @brief Função principal, ponto de entrada do programa no Núcleo 0.
 */
int main() {
    // 1. Inicializa o hardware básico (USB, OLED, PWM, gerador aleatório).
    inicia_hardware();

    // 2. Inicializa os componentes lógicos e lança o código de rede no Núcleo 1.
    inicia_core1();

    // 3. Loop de eventos principal e infinito. Este é o coração do programa no Núcleo 0.
    while (true) {
        // Tarefa 1: Verificar se há novas mensagens do Núcleo 1 na FIFO.
        verificar_fifo();

        // Tarefa 2: Processar uma mensagem da fila local (se houver).
        tratar_fila();

        // Tarefa 3: Verificar se as condições para iniciar o MQTT foram atendidas.
        inicializar_mqtt_se_preciso();

        // Tarefa 4: Enviar a mensagem "PING" se o tempo programado tiver chegado.
        enviar_ping_periodico();

        // Pausa muito curta para ceder tempo de processamento e evitar que o loop consuma 100% da CPU.
```

```
        sleep_ms(50);
    }

    // Este ponto do código nunca deve ser alcançado em um sistema embarcado.
    return 0;
}

/**
 * @brief Verifica a FIFO de comunicação inter-core por novas mensagens do Núcleo 1.
 * @details Esta função implementa o protocolo de comunicação customizado entre os núcleos.
 * Ela lê um "pacote" de 32 bits e o interpreta.
 */
void verificar_fifo(void) {
    // Verifica de forma não-bloqueante se há dados na FIFO para serem lidos. Se não houver, retorna imediatamente.
    if (!multicore_fifo_rvalid()) return;

    // Lê um pacote de 32 bits da FIFO. Esta chamada é bloqueante, mas só é executada se houver dados.
    uint32_t pacote = multicore_fifo_pop_blocking();

    // Extrai os 16 bits mais significativos, que são usados como um identificador da mensagem (ID).
    uint16_t id_msg = pacote >> 16;

    // O ID 0xFFFE é um "código mágico" que sinaliza que o próximo dado na FIFO é o endereço IP.
    if (id_msg == 0xFFFE) {
        uint32_t ip_bin = multicore_fifo_pop_blocking(); // Lê o endereço IP de 32 bits da FIFO.
        tratar_ip_binario(ip_bin); // Chama a função auxiliar para processar e exibir o IP.
        ip_recebido = true; // Define o flag para sinalizar que o IP foi recebido.
        return; // Finaliza a função, pois a mensagem de IP foi tratada.
    }

    // Se não for um IP, os 16 bits inferiores contêm o status real do Wi-Fi.
    uint16_t status = pacote & 0xFFFF;
    // E o `id_msg` representa o número da tentativa de conexão ou o ID do PING.
    uint16_t tentativa = id_msg;

    // Tratamento de erro básico para um status desconhecido. O ID 0x9999 é reservado para o PING.
    if (status > 2 && tentativa != 0x9999) {
        snprintf(mensagem_str, sizeof(mensagem_str), "Status inválido: %u (tentativa %u)", status, tentativa);
        exibir_e_esperar("Status inválido.", 0);
        printf("%s\n", mensagem_str);
        return;
    }

    // Cria a estrutura da mensagem com os dados recebidos.
    MensagemWiFi msg = {.tentativa = tentativa, .status = status};
    // Tenta inserir a mensagem na fila circular local.
    if (!fila_inserir(&fila_wifi, msg)) {
        // Se a fila estiver cheia, exibe um aviso e descarta a mensagem.
```

```
        exibir_e_esperar("Fila cheia.", 0);
        printf("Fila cheia. Mensagem descartada.\n");
    }
}

/**
 * @brief Remove e processa uma única mensagem da fila circular local.
 * @details Ao processar uma mensagem por vez, o loop principal se mantém responsivo
 * e não fica preso tratando múltiplas mensagens de uma só vez.
 */
void tratar_fila(void) {
    MensagemWiFi msg_recebida;
    // Tenta remover um item da fila. Se a operação for bem-sucedida (fila não estava vazia)...
    if (fila_remover(&fila_wifi, &msg_recebida)) {
        // ...chama a função de tratamento principal no arquivo auxiliar.
        tratar_mensagem(msg_recebida);
    }
}

/**
 * @brief Inicia o cliente MQTT se as condições necessárias forem satisfeitas.
 * @details Esta função atua como uma máquina de estados simples. Ela só executa sua ação
 * uma única vez, quando o cliente MQTT ainda não foi iniciado E um endereço IP
 * válido já foi recebido do Núcleo 1.
 */
void inicializar_mqtt_se_preciso(void) {
    // Verifica as duas condições para iniciar o MQTT.
    if (!mqtt_iniciado && ultimo_ip_bin != 0) {
        printf("[MQTT] Condições atendidas. Iniciando cliente MQTT...\n");
        iniciar_mqtt_cliente(); // Chama a função que configura e conecta o cliente.
        mqtt_iniciado = true;   // Define o flag para `true` para que esta lógica não seja executada novamente.
        // Agenda o primeiro envio de PING para dali a `INTERVALO_PING_MS`.
        proximo_envio = make_timeout_time_ms(INTERVALO_PING_MS);
    }
}

/**
 * @brief Envia a mensagem "PING" para o tópico MQTT em intervalos regulares.
 * @details Usa a função `absolute_time_diff_us` para uma verificação de tempo
 * não-bloqueante. A mensagem só é enviada se o cliente MQTT já estiver iniciado
 * e se o tempo definido em `INTERVALO_PING_MS` tiver passado.
 */
void enviar_ping_periodico(void) {
    // Verifica se o MQTT está ativo e se o tempo para o próximo envio já chegou.
    // `absolute_time_diff_us` retorna um valor negativo ou zero se o tempo de `proximo_envio` foi atingido.
    if (mqtt_iniciado && absolute_time_diff_us(get_absolute_time(), proximo_envio) <= 0) {
        publicar_mensagem_mqtt("PING"); // Publica a mensagem "PING" no tópico padrão.
        ssd1306_draw_utf8_multiline(buffer_oled, 0, 48, "PING enviado...");
```

```
        render_on_display(buffer_oled, &area);
        // Agenda o próximo envio, renovando o temporizador.
        proximo_envio = make_timeout_time_ms(INTERVALO_PING_MS);
    }
}

/**
 * @brief Inicializa o hardware básico do sistema no Núcleo 0.
 */
void inicia_hardware() {
    stdio_init_all();      // Inicializa a E/S padrão (necessário para printf via USB).
    setup_init_oled();     // Inicializa a comunicação I2C e o controlador do display OLED.
    espera_usb();          // Aguarda a conexão serial ser estabelecida para não perder mensagens de debug.

    // MELHORIA: Inicializa o gerador de números aleatórios com uma semente baseada no tempo.
    // Isto é crucial para que as cores do LED sejam diferentes a cada reinicialização.
    inicializar_aleatorio();

    oled_clear(buffer_oled, &area); // Limpa o buffer de vídeo do OLED.
    render_on_display(buffer_oled, &area); // Envia o buffer limpo para a tela, efetivamente limpando-a.
}

/**
 * @brief Inicializa os componentes lógicos e lança o Núcleo 1.
 */
void inicia_core1() {
    // Exibe mensagens de inicialização no display OLED para feedback visual.
    exibir_e_esperar("Nucleo 0 OK", 0);
    exibir_e_esperar("Iniciando Core 1", 16);

    printf(">> Núcleo 0 iniciado. Aguardando mensagens do núcleo 1...\n");

    init_rgb_pwm();        // Inicializa os pinos e o PWM para o controle do LED RGB.
    fila_inicializar(&fila_wifi); // Inicializa a estrutura da fila circular de mensagens.

    // Lança a função `funcao_wifi_nucleo1` no segundo núcleo.
    // A partir deste ponto, os dois núcleos do RP2040 estarão executando código em paralelo.
    multicore_launch_core1(funcao_wifi_nucleo1);
}
```

- Código: src/Atividade_Uni_02_Cap_03_auxiliar.c

```
/**
 * @file Atividade_Uni_02_Cap_03_auxiliar.c
 * @author Modificado Por Manoel Furtado
 * @version 1.2
 * @date 15 de junho de 2025
 * @brief Funções auxiliares do Núcleo 0 para interação com o usuário e tratamento de dados.
```

```
 *
 * @details
 * Este arquivo contém funções de suporte que são chamadas a partir do arquivo principal.
 * O objetivo desta separação é manter o arquivo principal mais limpo e focado na
 * lógica do loop de eventos, enquanto este arquivo lida com as "tarefas de apoio".
 *
 * As funções aqui presentes lidam com:
 * - Utilitários: Funções para gerar números aleatórios.
 * - Interface com o Usuário: Exibição de mensagens e status no display OLED.
 * - Feedback Visual: Controle do LED RGB para indicar o estado da rede, incluindo
 *                    a nova lógica de piscar com cor aleatória após um PING.
 * - Processamento de Dados: Interpretação das mensagens recebidas do Núcleo 1 via FIFO.
 */

// ============================================================================
// Includes de Bibliotecas
// ============================================================================
#include "fila_circular.h"      // Para o tipo de dado `MensagemWiFi`.
#include "rgb_pwm_control.h"    // Para a função `set_rgb_pwm`.
#include "configura_geral.h"    // Para constantes como `PWM_STEP`.
#include "oled_utils.h"         // Para funções como `exibir_e_esperar`.
#include "ssd1306_i2c.h"        // Para funções de desenho no OLED.
#include "estado_mqtt.h"        // Para acesso a variáveis globais como `buffer_oled`.
#include <stdio.h>              // Para `printf`.
#include <stdlib.h>             // Necessário para as funções `rand()` e `srand()`.
#include "pico/time.h"          // Necessário para `get_absolute_time()` e `sleep_ms()`.

// ============================================================================
// Protótipos de Funções Locais
// ============================================================================
// Embora estas funções sejam globais neste arquivo, elas não são expostas
// a outros arquivos através de um .h, sendo "privadas" a este módulo.
void inicializar_aleatorio(void);
int numero_aleatorio(int min, int max);

/**
 * @brief Inicializa o gerador de números aleatórios usando o tempo do sistema como semente.
 * @details Esta função deve ser chamada uma única vez no início do programa.
 * A "semente" (seed) garante que a sequência de números gerada por `rand()`
 * seja diferente a cada vez que o dispositivo é ligado.
 */
void inicializar_aleatorio() {
    // Usa o tempo em microssegundos desde o boot como "semente" para o gerador.
    // `get_absolute_time()` retorna um timestamp, e `to_us_since_boot` o converte para um número.
    srand(to_us_since_boot(get_absolute_time()));
}

/**
```

```c
 * @brief Gera um número inteiro aleatório dentro de um intervalo especificado.
 * @param min O valor mínimo do intervalo (inclusivo).
 * @param max O valor máximo do intervalo (inclusivo).
 * @return Um número inteiro aleatório entre min e max.
 */
int numero_aleatorio(int min, int max) {
    // `rand()` gera um número pseudoaleatório. O operador '%' (módulo) ajusta este número para o intervalo desejado.
    return rand() % (max - min + 1) + min;
}

/**
 * @brief Aguarda em um loop bloqueante até que a conexão USB esteja pronta.
 * @details Útil para depuração, para garantir que as mensagens `printf` do início
 * não sejam perdidas se o monitor serial for aberto com atraso.
 */
void espera_usb() {
    // Loop bloqueante que só termina quando a conexão USB é detectada.
    while (!stdio_usb_connected()) {
        sleep_ms(200); // Pequena pausa para não sobrecarregar a CPU.
    }
    printf("Conexão USB estabelecida!\n");
}

/**
 * @brief Processa e exibe uma mensagem de status do Wi-Fi ou de confirmação de PING.
 * @details Esta função é o principal dispatcher de mensagens. Ela verifica o tipo de mensagem
 * (PING ou status de Wi-Fi) e age de acordo.
 * @param msg A estrutura `MensagemWiFi` contendo o status e o tipo de tentativa.
 */
void tratar_mensagem(MensagemWiFi msg) {
    // O identificador 0x9999 é um "código mágico" para indicar que esta é uma resposta de PING MQTT.
    if (msg.tentativa == 0x9999) {
        if (msg.status == 0) { // Status 0 para ACK de PING significa sucesso.
            ssd1306_draw_utf8_multiline(buffer_oled, 0, 32, "ACK do PING OK");

            // --- INÍCIO DA MELHORIA IMPLEMENTADA ---
            uint16_t r, g, b; // Variáveis para armazenar os componentes de cor (Vermelho, Verde, Azul).

            // 1. Gera uma cor aleatória, garantindo que não seja um verde forte.
            do {
                r = numero_aleatorio(0, 65535); // Gera valor para vermelho (0 a 100% do PWM).
                g = numero_aleatorio(0, 65535); // Gera valor para verde.
                b = numero_aleatorio(0, 65535); // Gera valor para azul.
            } while (g > r && g > b && g > 32768); // A condição impede cores onde o verde é o componente mais forte
e brilhante.

            // 2. Define a cor aleatória gerada no LED para sinalizar visualmente o PING.
            set_rgb_pwm(r, g, b);
```

```
            render_on_display(buffer_oled, &area); // Atualiza o OLED junto com a mudança de cor do LED.

            // 3. Mantém a cor aleatória visível por 1 segundo.
            sleep_ms(1000);

            // 4. Retorna o LED para a cor verde padrão, indicando que a conexão continua OK.
            set_rgb_pwm(0, 65535, 0); // Verde sólido.
            // --- FIM DA MELHORIA IMPLEMENTADA ---

        } else { // Se o status do PING não for 0, significa falha.
            ssd1306_draw_utf8_multiline(buffer_oled, 0, 32, "ACK do PING FALHOU");
            set_rgb_pwm(65535, 0, 0); // LED Vermelho para indicar falha.
        }
        render_on_display(buffer_oled, &area); // Atualiza o display com o status final do PING.
        return; // Finaliza a função, pois a mensagem de PING já foi tratada.
    }

    // Se a mensagem não for de PING, executa a lógica antiga para status de conexão Wi-Fi.
    const char *descricao = "";
    switch (msg.status) {
        case 0:
            descricao = "INICIALIZANDO";
            set_rgb_pwm(PWM_STEP, 0, 0); // Vermelho.
            break;
        case 1:
            descricao = "CONECTADO";
            set_rgb_pwm(0, PWM_STEP, 0); // Verde.
            break;
        case 2:
            descricao = "FALHA";
            set_rgb_pwm(0, 0, PWM_STEP); // Azul.
            break;
        default:
            descricao = "DESCONHECIDO";
            set_rgb_pwm(PWM_STEP, PWM_STEP, PWM_STEP); // Branco.
            break;
    }
    // Formata e exibe a mensagem de status no OLED.
    char linha_status[32];
    snprintf(linha_status, sizeof(linha_status), "Status Wi-Fi: %s", descricao);
    oled_clear(buffer_oled, &area);
    ssd1306_draw_utf8_multiline(buffer_oled, 0, 0, linha_status);
    render_on_display(buffer_oled, &area);
    sleep_ms(2000); // Exibe a mensagem por 2 segundos.
    oled_clear(buffer_oled, &area);
    render_on_display(buffer_oled, &area);
    printf("[NÚCLEO 0] Status: %s\n", descricao);
}
```

```c
/**
 * @brief Converte um endereço IP de formato binário (uint32_t) para string.
 * @details Após a conversão, a função exibe o IP no OLED, imprime no console, e
 * armazena o valor na variável global `ultimo_ip_bin`, que serve de gatilho
 * para o Núcleo 0 iniciar o cliente MQTT.
 * @param ip_bin O endereço IP de 32 bits recebido do Núcleo 1.
 */
void tratar_ip_binario(uint32_t ip_bin) {
    char ip_str[20]; // Buffer para armazenar o IP formatado como string.
    // Função da stack lwIP que converte um endereço IP binário para string no formato "A.B.C.D".
    ip4addr_ntoa_r((const ip4_addr_t*)&ip_bin, ip_str, sizeof(ip_str));
    // Exibe o IP no display.
    oled_clear(buffer_oled, &area);
    ssd1306_draw_utf8_string(buffer_oled, 0, 0, "IP Recebido:");
    ssd1306_draw_utf8_string(buffer_oled, 0, 16, ip_str);
    render_on_display(buffer_oled, &area);
    // Imprime no console para depuração.
    printf("[NÚCLEO 0] Endereço IP: %s\n", ip_str);
    // Atualiza a variável global de estado.
    ultimo_ip_bin = ip_bin;
}

/**
 * @brief Exibe uma mensagem de status do MQTT no display OLED e no terminal.
 * @details Função utilitária para centralizar e padronizar a forma como os
 * status do MQTT são reportados ao usuário.
 * @param texto A string de status a ser exibida.
 */
void exibir_status_mqtt(const char *texto) {
    // Escreve "MQTT: " seguido pelo texto de status no display.
    ssd1306_draw_utf8_string(buffer_oled, 0, 16, "MQTT: ");
    ssd1306_draw_utf8_string(buffer_oled, 40, 16, texto);
    render_on_display(buffer_oled, &area);
    printf("[MQTT] %s\n", texto);
}
```