

# Aplicação com comunicação sem fio para IoT

Nome: **José Adriano Filho**

Matrícula: **2025101109806**

Unidade2-Capítulo3

1. Utilizando a placa Bitdoglab, crie um programa para monitorar o status de um pino, que possa ser definido como entrada, da placa Bitdoglab, e enviar, a cada 1 segundo o status atual para o servidor HiveMQ, utilizando o protocolo MQTT. Além disso, como desafio extra, acrescente algum sensor e envie a informação desse sensor para o servidor.

**R.: link do código:**

[https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/button\\_temp\\_mqtt](https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/button_temp_mqtt)

O código foi desenvolvido baseado em exemplos encontrados na internet já que existe uma dificuldade de documentações sobre MQTT em linguagem C. Procuramos desenvolver o monitoramento do botão e a leitura do sensor de temperatura interno, sendo que a cada um segundo enviamos o status e a medida em graus centígrados para o broker MQTT na plataforma da HiveMQ.

```
1  /* -----
2  / Projeto: Botão e Temperatura MQTT
3  / Descrição: Este código lê a temperatura do sensor e o estado de um botão, enviando os dados para um broker MQTT.
4  / Bibliotecas: pico-sdk, lwIP, CYW43
5  / Autor: José Adriano
6  / Obs: Necessário efetuar ajustes nos arquivos CMakeLists.txt e lwipopts.h para compilar e funcionar corretamente.
7  / Data de Criação: 22/06/2025
8  /-----
9  */
10 #include <stdio.h>
11 #include <string.h>
12 #include "pico/stdlib.h"
13 #include "hardware/gpio.h"
14 #include "hardware/adc.h"
15 #include "pico/cyw43_arch.h"
16 #include "lwip/apps/mqtt.h"
17 #include "lwip/ip_addr.h"
18 #include "lwip/dns.h"
19
20 // Configurações Wi-Fi
21 #define WIFI_SSID "Lu e Deza"
22 #define WIFI_PASSWORD "liukin1208"
23
24 // Configurações MQTT
25 #define MQTT_BROKER "broker.hivemq.com"
26 #define MQTT_BROKER_PORT 1883
27 #define MQTT_TOPIC "pico/status"
28
29 // Configurações do Botão
30 #define BUTTON_GPIO 5
31
```

Inicialmente fizemos os includes necessários para a comunicação via WiFi, bem como algumas definições importantes.

```

32 // Variáveis Globais
33 static mqtt_client_t *mqtt_client;
34 static ip_addr_t broker_ip;
35 static bool mqtt_connected = false;
36
37 // Protótipo das Funções
38 static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
39 void publish_msg(bool button_pressed, float temp_c);
40 float read_temperature();
41 void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg);
42

```

Acima algumas variáveis importantes e os protótipos das funções necessários para executar a leitura do sensor, a resolução do DNS, ou seja, converter o nome para acesso ao broker em um ip fixo, o callback para conexão MQTT e a função de publicar nossa mensagem no tópico.

A seguir, de acordo com as assinaturas das funções temos os códigos das funções para conexão do MQTT e publicar o status do botão e a temperatura interna.

```

111 // Callback de conexão MQTT
112 static void mqtt_connection_callback(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
113     if (status == MQTT_CONNECT_ACCEPTED) {
114         printf("[MQTT] Conectado ao broker!\n");
115         mqtt_connected = true;
116     } else {
117         printf("[MQTT] Falha na conexão MQTT. Código: %d\n", status);
118         mqtt_connected = false;
119     }
120 }
121
122 // Publicar botão + temperatura
123 void publish_msg(bool button_pressed, float temp_c) {
124     if (!mqtt_connected) {
125         printf("[MQTT] Não conectado, não publicando dados\n");
126         return;
127     }
128
129     char payload[128];
130     snprintf(payload, sizeof(payload),
131             "{\"botao\":\"%s\", \"temperatura\":%.2f\",
132             button_pressed ? \"ON\" : \"OFF\",
133             temp_c);
134
135     printf("[MQTT] Publicando: tópico='%s', mensagem='%s'\n", MQTT_TOPIC, payload);
136
137     err_t err = mqtt_publish(mqtt_client, MQTT_TOPIC, payload, strlen(payload), 0, 0, NULL, NULL);
138     if (err == ERR_OK) {
139         printf("[MQTT] Publicação OK\n");
140     } else {
141         printf("[MQTT] Erro ao publicar: %d\n", err);
142     }
143 }

```

A seguir as outras duas funções, responsáveis pela leitura da temperatura e a resolução do DNS.

```

145 // Leitura da temperatura
146 float read_temperature() {
147     uint16_t raw = adc_read();
148     const float conversion_factor = 3.3f / (1 << 12);
149     float voltage = raw * conversion_factor;
150     float temp_c = 27.0f - (voltage - 0.706f) / 0.001721f;
151     return temp_c;
152 }
153
154 // Callback de DNS
155 void dns_check_callback(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
156     if (ipaddr != NULL) {
157         broker_ip = *ipaddr;
158         printf("[DNS] Resolvido: %s -> %s\n", name, ipaddr_ntoa(ipaddr));
159
160         struct mqtt_connect_client_info_t client_info = {
161             .client_id = "pico-client",
162             .keep_alive = 60,
163             .client_user = NULL,
164             .client_pass = NULL,
165             .will_topic = NULL,
166             .will_msg = NULL,
167             .will_qos = 0,
168             .will_retain = 0
169         };
170
171         printf("[MQTT] Conectando ao broker...\n");
172         mqtt_client_connect(mqtt_client, &broker_ip, MQTT_BROKER_PORT, mqtt_connection_callback, NULL, &client_info);
173     } else {
174         printf("[DNS] Falha ao resolver DNS para %s\n", name);
175     }
176 }
177

```

Abaixo o código desenvolvido para a função principal main.

```

43 // Função Principal
44 int main() {
45     stdio_init_all();
46     sleep_ms(2000);
47     printf("\n=== Iniciando MQTT Button + Temperature ===\n");
48
49     // Inicializa Wi-Fi
50     if (cyw43_arch_init()) {
51         printf("Erro na inicialização do Wi-Fi\n");
52         return -1;
53     }
54     cyw43_arch_enable_sta_mode();
55
56     printf("[Wi-Fi] Conectando...\n");
57     if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
58         printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
59         return -1;
60     } else {
61         printf("[Wi-Fi] Conectado com sucesso!\n");
62     }
63
64     // Configura GPIO do botão
65     gpio_init(BUTTON_GPIO);
66     gpio_set_dir(BUTTON_GPIO, GPIO_IN);
67     gpio_pull_up(BUTTON_GPIO); // Pull-up ativado
68
69     // Inicializa ADC para temperatura
70     adc_init();
71     adc_set_temp_sensor_enabled(true);
72     adc_select_input(4);
73
74     // Inicializa cliente MQTT
75     mqtt_client = mqtt_client_new();
76
77     // Resolve DNS do broker MQTT
78     err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_check_callback, NULL);
79     if (err == ERR_OK) {
80         dns_check_callback(MQTT_BROKER, &broker_ip, NULL);
81     } else if (err == ERR_INPROGRESS) {
82         printf("[DNS] Resolvendo...\n");
83     } else {
84         printf("[DNS] Erro ao resolver DNS: %d\n", err);
85         return -1;
86     }
87

```

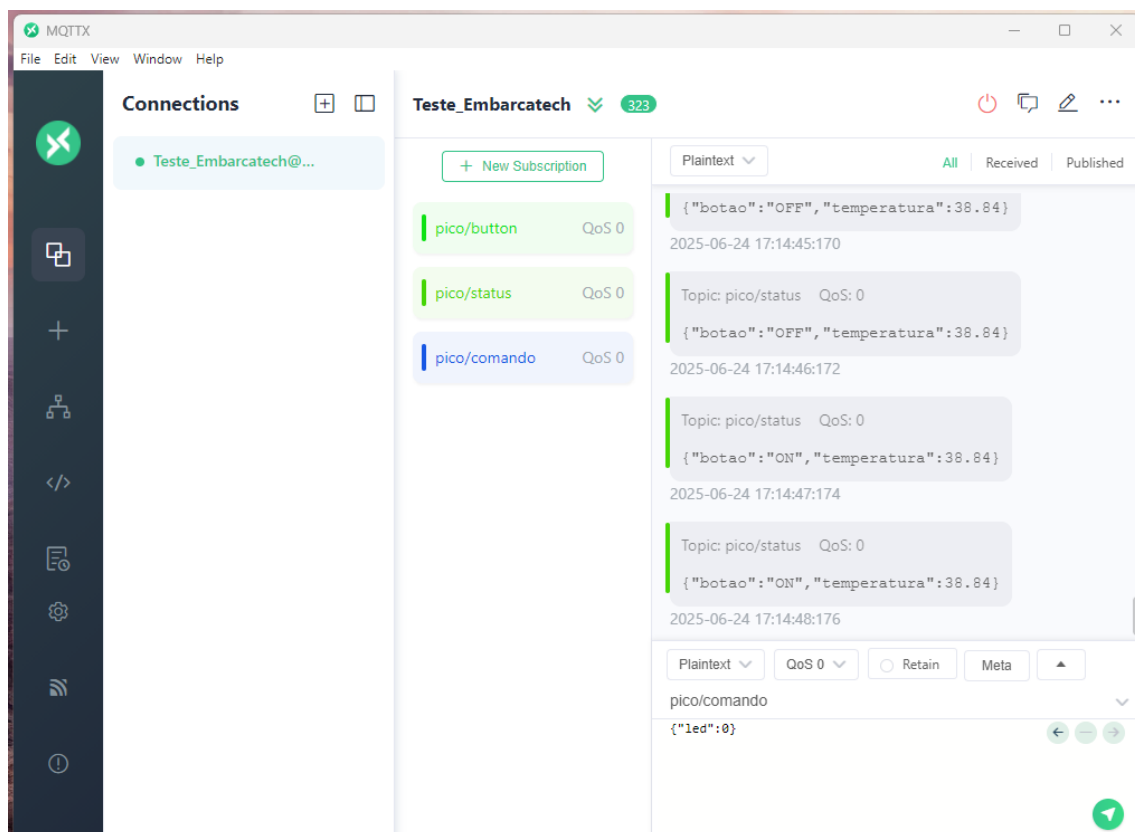
```

88 // Loop principal
89 while (true) {
90     // Atualiza tarefas de rede
91     cyw43_arch_poll();
92
93     // Lê botão
94     bool button_state = !gpio_get(BUTTON_GPIO); // Inversão por pull-up
95
96     // Lê temperatura
97     float temp_c = read_temperature();
98     printf("[TEMP] Temperatura atual: %.2f °C\n", temp_c);
99
100    // Publica ambos no mesmo tópico
101    publish_msg(button_state, temp_c);
102
103    // Espera 1 segundo
104    sleep_ms(1000);
105 }
106
107 cyw43_arch_deinit();
108 return 0;
109 }

```

A cada 1 segundo executamos a atualização do status do botão e do valor do sensor de temperatura.

Os testes foram executados com o cliente MQTT “MQTTX” rodando em um pc e inscrito no tópico “pico/status”.



Na saída serial temos alguns “prints” para monitoramento e debug da aplicação.

```

[DNS] Resolvido: broker.hivemq.com -> 3.65.119.144
[MQTT] Conectando ao broker...
[MQTT] Conectado ao broker!
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"OFF","temperatura":41.18}'
[MQTT] Publicação OK
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"ON","temperatura":41.18}'
[MQTT] Publicação OK
[TEMP] Temperatura atual: 41.18 °C
[MQTT] Publicando: tópico='pico/status', mensagem='{"botao":"OFF","temperatura":41.18}'

```

2. Utilizando a placa Bitdoglab, crie um programa para receber dados do servidor HiveMQ, utilizando o protocolo MQTT. Os dados recebidos devem ligar ou desligar um LED, que deve ser ligado a placa. Além disso, como desafio extra, implemente um LCD ou outro tipo de display para monitorar o recebimento de mensagens.

**R.: link do código:**

[https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/led\\_lcd\\_mqtt](https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/U2C3T3/led_lcd_mqtt)

Nossa proposta agora é comandar um led por comando enviado pelo protocolo MQTT, bem como sinalizar no display da BigDogLab o status dos acontecimentos durante a execução.

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4  #include "pico/stdlib.h"
5  #include "hardware/gpio.h"
6  #include "hardware/i2c.h"
7  #include "pico/cyw43_arch.h"
8  #include "lwip/apps/mqtt.h"
9  #include "lwip/ip_addr.h"
10 #include "lwip/dns.h"
11
12 #include "ssd1306.h"
13
14 // ===== Configurações =====
15 #define WIFI_SSID "Lu e Deza"
16 #define WIFI_PASSWORD "liukin1208"
17
18 #define MQTT_BROKER "broker.hivemq.com"
19 #define MQTT_PORT_BROKER 1883
20
21 #define MQTT_TOPIC_PUB "pico/status"
22 #define MQTT_TOPIC_SUB "pico/comando"
23
24 #define BUTTON_GPIO 5
25 #define LED_GPIO 12
26
27 #define I2C_SDA 14
28 #define I2C_SCL 15

```

Aqui temos os includes iniciais e as definições necessárias para o desenvolvimento do programa.

```

30 // ===== Variáveis Globais =====
31 static mqtt_client_t *mqtt_client;
32 static ip_addr_t broker_ip;
33 static bool mqtt_connected = false;
34 static bool button_last_state = false;
35
36 // ===== Protótipos =====
37 void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status);
38 void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len);
39 void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags);
40 void publish_button_state(bool pressed);
41 void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg);

```

Na imagem acima temos as variáveis necessárias, já que nos baseamos em nosso programa anterior, bem como os protótipos das funções desenvolvidas.

Na página seguinte temos a codificação das duas funções.

```

123 // ===== Callbacks e Publicação =====
124 void dns_found_cb(const char *name, const ip_addr_t *ipaddr, void *callback_arg) {
125     if (ipaddr != NULL) {
126         broker_ip = *ipaddr;
127         printf("[DNS] %s -> %s\n", name, ipaddr_ntoa(ipaddr));
128
129         struct mqtt_connect_client_info_t ci = {
130             .client_id = "pico-client",
131             .keep_alive = 60,
132             .client_user = NULL,
133             .client_pass = NULL,
134             .will_topic = NULL,
135             .will_msg = NULL,
136             .will_qos = 0,
137             .will_retain = 0
138         };
139
140         printf("[MQTT] Conectando...\n");
141         mqtt_client_connect(mqtt_client, &broker_ip, MQTT_PORT_BROKER, mqtt_connection_cb, NULL, &ci);
142     } else {
143         printf("[DNS] Falha na resolução\n");
144     }
145 }
146
147 void mqtt_connection_cb(mqtt_client_t *client, void *arg, mqtt_connection_status_t status) {
148     if (status == MQTT_CONNECT_ACCEPTED) {
149         printf("[MQTT] Conectado!\n");
150         mqtt_connected = true;
151
152         mqtt_set_inpub_callback(client, mqtt_incoming_publish_cb, mqtt_incoming_data_cb, NULL);
153
154         err_t err = mqtt_subscribe(client, MQTT_TOPIC_SUB, 0, NULL, NULL);
155         if (err != ERR_OK) {
156             printf("[MQTT] Erro subscribe: %d\n", err);
157         } else {
158             printf("[MQTT] Inscrito no topico %s\n", MQTT_TOPIC_SUB);
159         }
160
161         ssd1306_clear();
162         ssd1306_draw_string(0, 0, "MQTT Conectado");
163         ssd1306_show();
164     } else {
165         printf("[MQTT] Falha conexão: %d\n", status);
166         mqtt_connected = false;
167     }
168 }

```

```

170 void mqtt_incoming_publish_cb(void *arg, const char *topic, u32_t tot_len) {
171     printf("[MQTT] Mensagem recebida no topico: %s\n", topic);
172 }
173
174 void mqtt_incoming_data_cb(void *arg, const u8_t *data, u16_t len, u8_t flags) {
175     char msg[len + 1];
176     memcpy(msg, data, len);
177     msg[len] = 0;
178     printf("[MQTT] Payload: %s\n", msg);
179
180     if (strstr(msg, "\\led\\:1")) {
181         gpio_put(LED_GPIO, 1);
182     } else if (strstr(msg, "\\led\\:0")) {
183         gpio_put(LED_GPIO, 0);
184     }
185
186     ssd1306_clear();
187     ssd1306_draw_string(0, 0, "Recebido:");
188     ssd1306_draw_string(0, 10, msg);
189     ssd1306_draw_string(0, 30, "LED:");
190     ssd1306_draw_string(40, 30, gpio_get(LED_GPIO) ? "ON" : "OFF");
191     ssd1306_show();
192 }
193
194 void publish_button_state(bool pressed) {
195     if (!mqtt_connected) return;
196
197     char payload[64];
198     snprintf(payload, sizeof(payload), "\\button\\:%d", pressed ? 1 : 0);
199
200     err_t err = mqtt_publish(mqtt_client, MQTT_TOPIC_PUB, payload, strlen(payload), 0, 0, NULL, NULL);
201     if (err == ERR_OK) {
202         printf("[MQTT] Publicado: %s\n", payload);
203     } else {
204         printf("[MQTT] Erro publicar: %d\n", err);
205     }
206 }

```

Após o desenvolvimento das funções, montamos nosso algoritmo para a função principal, conforme mostram as imagens abaixo.

```

43 // ===== Função Principal =====
44 int main() {
45     stdio_init_all();
46     sleep_ms(2000);
47     printf("\n=== Iniciando ===\n");
48
49     // Inicializa Wi-Fi
50     if (cyw43_arch_init()) {
51         printf("Erro na inicialização Wi-Fi\n");
52         return -1;
53     }
54     cyw43_arch_enable_sta_mode();
55
56     printf("[Wi-Fi] Conectando...\n");
57     if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 10000)) {
58         printf("[Wi-Fi] Falha na conexão Wi-Fi\n");
59         return -1;
60     }
61     printf("[Wi-Fi] Conectado!\n");
62 }

```

```

63 // Inicializa I2C e OLED
64 i2c_init(i2c1, 400000);
65 gpio_set_function(I2C_SDA, GPIO_FUNC_I2C);
66 gpio_set_function(I2C_SCL, GPIO_FUNC_I2C);
67 gpio_pull_up(I2C_SDA);
68 gpio_pull_up(I2C_SCL);
69
70 ssd1306_init(i2c1);
71 ssd1306_clear();
72 ssd1306_draw_string(0, 0, "Wi-Fi OK");
73 ssd1306_draw_string(0, 10, "Conectando MQTT");
74 ssd1306_show();
75
76 // Configura Botão
77 gpio_init(BUTTON_GPIO);
78 gpio_set_dir(BUTTON_GPIO, GPIO_IN);
79 gpio_pull_up(BUTTON_GPIO);
80
81 // Configura LED
82 gpio_init(LED_GPIO);
83 gpio_set_dir(LED_GPIO, GPIO_OUT);
84
85 // Inicializa MQTT
86 mqtt_client = mqtt_client_new();
87 err_t err = dns_gethostbyname(MQTT_BROKER, &broker_ip, dns_found_cb, NULL);
88 if (err == ERR_OK) {
89     dns_found_cb(MQTT_BROKER, &broker_ip, NULL);
90 } else if (err == ERR_INPROGRESS) {
91     printf("[DNS] Resolvendo...\n");
92 } else {
93     printf("[DNS] Erro DNS: %d\n", err);
94     return -1;
95 }

```

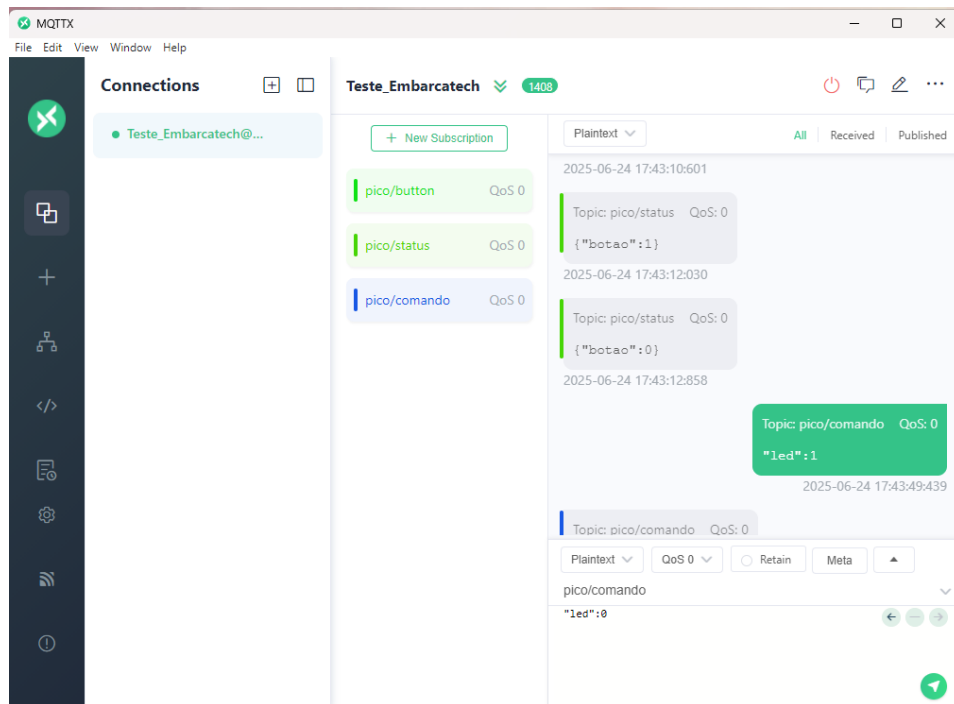
```

97 // Loop Principal
98 while (true) {
99     cyw43_arch_poll();
100
101     bool button_state = !gpio_get(BUTTON_GPIO);
102     if (button_state != button_last_state) {
103         printf("[BOTAO] %s\n", button_state ? "ON" : "OFF");
104         publish_button_state(button_state);
105
106         ssd1306_clear();
107         ssd1306_draw_string(0, 0, "Botao:");
108         ssd1306_draw_string(50, 0, button_state ? "ON" : "OFF");
109         ssd1306_draw_string(0, 10, "LED:");
110         ssd1306_draw_string(50, 10, gpio_get(LED_GPIO) ? "ON" : "OFF");
111         ssd1306_show();
112
113         button_last_state = button_state;
114     }
115
116     sleep_ms(200);
117 }
118
119 cyw43_arch_deinit();
120 return 0;
121 }

```



Como na versão anterior, utilizamos o “MQTTX” como cliente para enviar e receber os comandos MQTT do broker da HiveMQ.



Fotos da placa BigDogLab executando o firmware:

