

Aplicação com comunicação sem fio para IoT

Nome: **José Adriano Filho**

Matrícula: **2025101109806**

Unidade2-Capítulo3

1. Com base no código apresentado na aula do capítulo 3, da unidade 2, crie um programa para monitorar os status do botão da placa e enviar, a cada 1 segundo os status para um servidor. Além disso, como um 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/btnServer>

O código foi desenvolvido baseado no código da lição, assim procuramos desenvolver o monitoramento do botão e a cada um segundo recebemos uma requisição no servidor web e enviamos o status do mesmo.

```
C btn_sensor_server.c > main()
1  #include <stdio.h>
2  #include "pico/stdlib.h"
3  #include "hardware/adc.h"
4  #include "hardware/gpio.h"
5  #include "pico/cyw43_arch.h"
6  #include "lwip/netif.h"
7  #include "lwip/pbuf.h"
8  #include "lwip/udp.h"
9  #include "lwip/ip_addr.h"
10
11 #define WIFI_SSID "xxxxxxxx" // Nome da rede Wi-Fi
12 #define WIFI_PASSWORD "xxxxxxxx" // Senha da rede Wi-Fi
13
14 #define SERVER_IP "192.168.1.28" // IP do servidor para onde enviar os dados
15 #define SERVER_PORT 34567 // Porta do servidor
16
17 #define BUTTON_PIN 5 // GPIO do botão
18 #define ADC_TEMP 4 // Canal ADC do sensor de temperatura interno
```

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

```
//Protótipos de funções
void mostra_ip(); // Função para exibir o IP da placa
const char* le_botao(); // Função para ler o estado do botão
float le_temperatura(); // Função para ler a temperatura
void envia_udp(struct udp_pcb *pcb, const char *msg); // Função para enviar dados via UDP
```

Acima os protótipos das funções, que em nosso caso são três: uma para iniciar o servidor, uma para receber a conexão TCP e uma para receber as requisições.

Seguindo a assinatura dos protótipos desenvolvemos as funções que executarão a leitura do status do botão, o valor de temperatura do sensor interno da placa BitDogLab e o envio via protocolo UDP, já que não temos criticidade nos dados, optamos pelo mesmo pela simplicidade em codificar.

```

// Função para obter o IP da placa
void mostra_ip() {
    struct netif *netif = netif_default;
    if (netif) {
        printf("Endereço IP da placa: %s\n", ipaddr_ntoa(&netif->ip_addr));
    } else {
        printf("Erro ao obter o IP\n");
    }
}

// Função para ler o estado do botão
const char* le_botao() {
    return gpio_get(BUTTON_PIN) ? "LIBERADO" : "PRESSIONADO"; // Invertido devido ao pull-up interno
}

// Função para ler a temperatura
float le_temperatura() {
    adc_select_input(ADC_TEMP);
    uint16_t raw = adc_read();
    float voltage = raw * 3.3f / (1 << 12);
    return 27.0 - (voltage - 0.706) / 0.001721;
}

```

```

// Função para enviar dados via UDP
void envia_udp(struct udp_pcb *pcb, const char *msg) {
    struct pbuf *p = pbuf_alloc(PBUF_TRANSPORT, strlen(msg), PBUF_RAM);
    if (!p) return;
    memcpy(p->payload, msg, strlen(msg));

    ip_addr_t dest_ip;
    ipaddr_aton(SERVER_IP, &dest_ip);

    udp_sendto(pcb, p, &dest_ip, SERVER_PORT);
    pbuf_free(p);
}

```

Abaixo o código desenvolvido para a função principal main.

```

// Função principal
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("Iniciando...\n");

    // Configurar GPIO do botão
    gpio_init(BUTTON_PIN);
    gpio_set_dir(BUTTON_PIN, GPIO_IN);
    gpio_pull_up(BUTTON_PIN);

    // Inicializar ADC
    adc_init();
    adc_set_temp_sensor_enabled(true);

    // Inicializar Wi-Fi
    if (cyw43_arch_init()) {
        printf("Erro ao inicializar Wi-Fi\n");
        return -1;
    }

    cyw43_arch_enable_sta_mode();

    printf("Conectando ao Wi-Fi...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 30000)) {
        printf("Falha na conexão Wi-Fi\n");
        return -1;
    }

    printf("Wi-Fi conectado!\n");

    // Exibir IP da placa
    mostra_ip();

    // Inicializar UDP
    struct udp_pcb *pcb = udp_new();
    if (!pcb) {
        printf("Erro ao criar PCB UDP\n");
        return -1;
    }
}

```

```

while (true) {
    const char* button_state = le_botao();
    float temperature = le_temperatura();

    // Criar mensagem com os dados
    char msg[100];
    snprintf(msg, sizeof(msg), "Botao: %s, Temperatura: %.2f Celsius", button_state, temperature);
    printf("Enviando: %s\n", msg);

    // Enviar via UDP
    envia_udp(pcb, msg);

    sleep_ms(1000);
}

cyw43_arch_deinit();
return 0;
}

```

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

Os testes foram executados com o software “**Packet Sender**” que cria um servidor local em nossa máquina.

The screenshot shows the Packet Sender application interface. The top section contains fields for Name, ASCII, and HEX representations, along with Address, Port, Resend Delay, and TCP options. Below this is a table of saved packets with columns for Send, Name, Resend, To Address, To Port, and Method. The bottom section shows a detailed traffic log with columns for Time, From IP, From Port, To Address, To Port, Method, Error, and ASCII. A red circle highlights the ASCII column, showing the data being sent: "Botao: PRESSIONADO, Temperatura: 38.84 ..." and "Botao: LIBERADO, Temperatura: 38.84 Celsius".

Send	Name	Resend	To Address	To Port	Method
1	DNS dannagle.com	0	1.1.1.1	53	UDP
2	DNS example.com	0	8.8.8.8	53	UDP
3	FTP debian.org	0	cdimage.debian.org	21	TCP
4	Google DNS over HTTPS / DoH	0	dns.google	443	HTTPS Get
5	HTTP GET	0	neverssl.com	80	HTTP Get
6	HTTP POST Params	0	httpbin.org	80	HTTP Post
7	HTTPS GitHub API	0	api.github.com	443	HTTPS Get
8	HTTPS POST JSON	0	httpbin.org	443	HTTPS Post
9	HTTPS POST Params	0	httpbin.org	443	HTTPS Post
10	HTTPS POST XML	0	httpbin.org	443	HTTPS Post
11	HTTPS Search DuckDuckGo	0	html.duckduckgo.com	443	HTTPS Post
12	IBM DNS over HTTPS / DoH	0	dns.quad9.net	5053	HTTPS Get
13	NTP querv	0	pool.ntp.org	123	UDP

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII
2025-06-08 10:06:53.523	192.168.1.12	54380	You	34567	UDP		Botao: PRESSIONADO, Temperatura: 38.84 ...
2025-06-08 10:06:52.517	192.168.1.12	54380	You	34567	UDP		Botao: PRESSIONADO, Temperatura: 38.84 ...
2025-06-08 10:06:51.523	192.168.1.12	54380	You	34567	UDP		Botao: PRESSIONADO, Temperatura: 38.37 ...
2025-06-08 10:06:50.524	192.168.1.12	54380	You	34567	UDP		Botao: PRESSIONADO, Temperatura: 38.37 ...
2025-06-08 10:06:49.521	192.168.1.12	54380	You	34567	UDP		Botao: PRESSIONADO, Temperatura: 38.37 ...
2025-06-08 10:06:48.524	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.84 Celsius
2025-06-08 10:06:47.524	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.84 Celsius
2025-06-08 10:06:46.514	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.84 Celsius
2025-06-08 10:06:45.524	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.37 Celsius
2025-06-08 10:06:44.525	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.37 Celsius
2025-06-08 10:06:43.522	192.168.1.12	54380	You	34567	UDP		Botao: LIBERADO, Temperatura: 38.37 Celsius

2. Com base no código apresentado na aula do capítulo 3, da unidade 2, crie um programa para ler a posição do joystick e enviar a posição X e Y para um servidor via Wi-Fi. Além disso, como um desafio extra, crie uma rosa dos ventos imaginária e envie para o aplicativo a posição (Norte, Sul Leste, Oeste, Nordeste, Sudeste, Noroeste e Sudoeste) selecionada no joystick.

R.: link do código:

<https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/RosaDosVentos>

Assim como foi pedido, desenvolvemos um código que monitora um dispositivo joystick e ao movimentar codificamos sua posição em direções de uma rosa dos ventos.

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/adc.h"
#include "pico/cyw43_arch.h"
#include "lwip/pbuf.h"
#include "lwip/udp.h"
#include "lwip/ip_addr.h"

// Configurações do Wi-Fi
#define WIFI_SSID "xxxxxxx"
#define WIFI_PASSWORD "xxxxxxx"

// IP do servidor UDP
#define SERVER_IP "192.168.1.32"
#define SERVER_PORT 12345

// Pinos do joystick
#define ADC_PIN_X 26 // GP26 -> ADC0
#define ADC_PIN_Y 27 // GP27 -> ADC1
```

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

```
// Protótipos de funções
const char* direcao_geografica(uint16_t x, uint16_t y);
void send_udp(struct udp_pcb *pcb, const char *msg);
```

Nossos protótipos das funções são dois, que mostram as duas funções necessárias para o funcionamento do sistema. Uma delas será responsável por decodificar os valores numéricos lidos do joystick e converter para posições geográficas tipo norte, sul, leste, oeste, nordeste e assim sucessivamente, simulando uma rosa dos ventos.

Na página seguinte temos a codificação das duas funções.

```
// Função para obter direção na rosa dos ventos
const char* direcao_geografica(uint16_t x, uint16_t y) {
    const uint16_t centro = 2048;
    const uint16_t zona_central = 500;

    int deltaX = x - centro;
    int deltaY = y - centro;

    if (abs(deltaX) < zona_central && abs(deltaY) < zona_central) {
        return "Centro";
    }

    if (deltaY > zona_central) {
        if (deltaX > zona_central) return "Nordeste";
        else if (deltaX < -zona_central) return "Noroeste";
        else return "Norte";
    } else if (deltaY < -zona_central) {
        if (deltaX > zona_central) return "Sudeste";
        else if (deltaX < -zona_central) return "Sudoeste";
        else return "Sul";
    } else {
        if (deltaX > zona_central) return "Leste";
        else if (deltaX < -zona_central) return "Oeste";
    }

    return "Centro";
}
```

```
// Função para enviar dados via UDP
void send_udp(struct udp_pcb *pcb, const char *msg) {
    struct pbuf *p = pbuf_alloc(PBUF_TRANSPORT, strlen(msg), PBUF_RAM);
    if (!p) return;
    memcpy(p->payload, msg, strlen(msg));

    ip_addr_t dest_ip;
    ipaddr_aton(SERVER_IP, &dest_ip);

    udp_sendto(pcb, p, &dest_ip, SERVER_PORT);
    pbuf_free(p);
}
```

Escolhemos utilizar o protocolo UDP para enviarmos os dados já que não há criticidade no envio, assim não necessitamos de confirmação do envio. Na página seguinte temos nossa função principal, inicializando os periféricos, efetuando a conexão com WiFi, lendo as informações do joystick, convertendo em posições geográficas e enviando para um servidor.

```

// Função principal
int main() {
    stdio_init_all();
    sleep_ms(2000);
    printf("Iniciando...\n");

    // Inicializar ADC
    adc_init();
    adc_gpio_init(ADC_PIN_X);
    adc_gpio_init(ADC_PIN_Y);

    // Inicializar Wi-Fi
    if (cyw43_arch_init()) {
        printf("Erro ao inicializar Wi-Fi\n");
        return -1;
    }

    cyw43_arch_enable_sta_mode();

    printf("Conectando ao Wi-Fi...\n");
    if (cyw43_arch_wifi_connect_timeout_ms(WIFI_SSID, WIFI_PASSWORD, CYW43_AUTH_WPA2_AES_PSK, 30000)) {
        printf("Falha na conexão Wi-Fi\n");
        return -1;
    }

    printf("Wi-Fi conectado!\n");

    // Inicializar UDP
    struct udp_pcb *pcb = udp_new();
    if (!pcb) {
        printf("Erro ao criar PCB UDP\n");
        return -1;
    }
}

```

```

while (true) {
    // Ler eixo X
    adc_select_input(0);
    uint16_t x = adc_read();

    // Ler eixo Y
    adc_select_input(1);
    uint16_t y = adc_read();

    // Obter direção
    const char *direcao = direcao_geografica(x, y);

    // Criar mensagem
    char msg[100];
    snprintf(msg, sizeof(msg), "X:%d,Y:%d,Direcao:%s", x, y, direcao);
    printf("Enviando: %s\n", msg);

    // Enviar via UDP
    send_udp(pcb, msg);

    sleep_ms(500);
}

cyw43_arch_deinit();
return 0;
}

```

Na página uma imagem de um servidor no PC recebendo os dados da placa:

Packet Sender - IPs: 192.168.1.28, 2804:d4:b5:f0:fc00:8c55:277e:7ab2:c2bc, 2804:d4:b5:f0:fc00:9470:1efe:4ac5:6b6d, fe80:5a82:ddd2:ab7a:4...

File Tools Multicast Panels Help

Name Packet Name

ASCII ASCII representation Load File

HEX HEX representation

Address IPv4, IPv6, DNS Lookup Port 1 to 65535 Resend Delay 0.0/blank off TCP Send Save

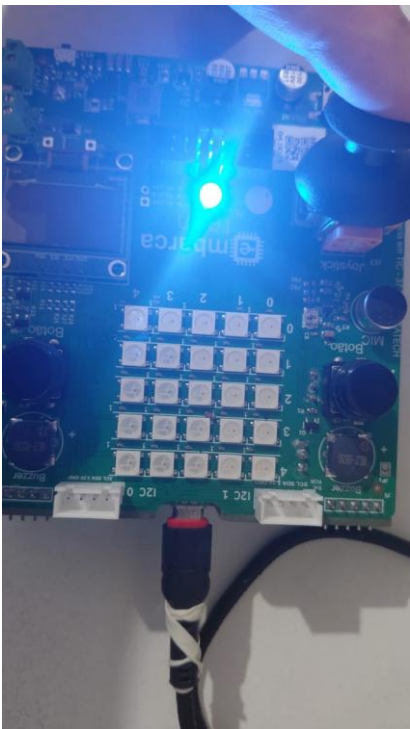
Search Saved Packets... Delete Saved Packet Persistent TCP

	Send	Name	Resend	To Address	To Port	Method	
1	Send	DNS dannagle.com	0	1.1.1.1	53	UDP	\f7\01\00\00\01\00\00\00\00\00\00\00\08dannagle\03con
2	Send	DNS example.com	0	8.8.8.8	53	UDP	\91\8b\01\00\00\01\00\00\00\00\00\00\00\07example\03con
3	Send	FTP debian.org	0	cdimage.debian.org	21	TCP	\nUSER anonymous\nPASS anonymous\nquit\n
4	Send	Google DNS over HTTPS / DoH	0	dns.google	443	HTTPS Get	/resolve?name=packetsender.com
5	Send	HTTP GET	0	neverssl.com	80	HTTP Get	/
6	Send	HTTP POST Params	0	httpbin.org	80	HTTP Post	/post
7	Send	HTTPS GitHub API	0	api.github.com	443	HTTPS Get	/users/dannagle
8	Send	HTTPS POST JSON	0	httpbin.org	443	HTTPS Post	/post

Clear Log (85) Log Traffic Save Log Save Traffic Packet Copy to Clipboard

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII	
2025-06-08 10:18:05.826	192.168.1.12	60081	You	34567	UDP		X:764,Y:3968,Direcao:Noroeste	58 3A 37 36 34 2C 59 3A 3
2025-06-08 10:18:05.322	192.168.1.12	60081	You	34567	UDP		X:636,Y:3963,Direcao:Noroeste	58 3A 36 33 36 2C 59 3A 3
2025-06-08 10:18:04.822	192.168.1.12	60081	You	34567	UDP		X:810,Y:4005,Direcao:Noroeste	58 3A 38 31 30 2C 59 3A 3
2025-06-08 10:18:04.327	192.168.1.12	60081	You	34567	UDP		X:1848,Y:4084,Direcao:Norte	58 3A 31 38 34 38 2C 59 3A
2025-06-08 10:18:03.829	192.168.1.12	60081	You	34567	UDP		X:2220,Y:3686,Direcao:Norte	58 3A 32 32 32 30 2C 59 3A
2025-06-08 10:18:03.319	192.168.1.12	60081	You	34567	UDP		X:3889,Y:3588,Direcao:Nordeste	58 3A 33 38 38 39 2C 59 3A
2025-06-08 10:18:02.818	192.168.1.12	60081	You	34567	UDP		X:4081,Y:3030,Direcao:Nordeste	58 3A 3A 30 38 31 2C 59 3A

DTLS Server Disabled UDP:34567 TCP Server Disabled SSL Server Disabled IPv4 Mode



Placa utilizada no envio dos dados.

Questão desafio:

Servidor na nuvem: Refaça as tarefas anteriores, utilizando um servidor na nuvem, como por exemplo: AWS, Google e entre outros.

Nesta questão, executamos a modificação no programa para enviar os dados para um servidor em nuvem, abaixo o link no github do código modificado:

https://github.com/EngAdriano/Residencia/tree/main/Exercicios/U2C3Tarefas/btn_sensor_nu_vem

Na tela abaixo o resultado na tela do servidor em nuvem.

The screenshot displays the Webhook.site web application. The top navigation bar includes links for Docs & API, Features & Pricing, Terms, Privacy & Security, and Support. Below the navigation bar, there's a toolbar with icons for various actions like Share, Schedule, Form Builder, CSV Export, Custom Actions, Replay, XHR Redirect, and Redirect Now. The main interface is divided into two panels. The left panel, titled 'INBOX (100/100)', shows a list of incoming POST requests from the IP address 189.106.13.206. The right panel, titled 'Request Details & Headers', shows the details of a selected request. A red rectangle highlights the 'Raw Content' section, which contains the text 'Temperatura: 39.78 C, Botao: LIBERADO'. Below this, the 'Custom Actions Output' section is visible, showing 'No action output' and a 'Create Custom Action' button.

Webhook.site Docs & API Features & Pricing Terms, Privacy & Security Support

32a77e20 Share Schedule Form Builder CSV Export Custom Actions Replay XHR Redirect Redirect Now More

INBOX (100/100) Oldest First Search Query

Upgrade to a Webhook.site account to unlock unlimited requests
This URL received the maximum of 100 requests and can't accept more requests, emails or DNSHooks. New requests start from our terms of service to use Webhook.site for any kind of load testing or benchmarking.
[Upgrade Now](#)

Request Details & Headers

POST http://webhook.site/32a77e20-204e-46f1-b195-b5632fbd9e4c

Host 189.106.13.206 Whois Shodan Netify Censys VirusTotal

Date 07/06/2025 15:16:00 (há 43 minutos)

Size 37 bytes

Time 0.000 sec

ID 6ef38764-9f15-4fac-9671-6ce3e6790713

Note [Add Note](#)

Query strings
None

Request Content

Raw Content
Temperatura: 39.78 C, Botao: LIBERADO

Custom Actions Output
No action output [Create Custom Action](#)