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.

btn_sensor_server.c > main()

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

#define WIFI_SSID "xxxxxxxxx"       // Nome da rede Wi-Fi
#define WIFI_PASSWORD "xxxxxxxxx"   // Senha da rede Wi-Fi

#define SERVER_IP "192.168.1.28"    // IP do servidor para onde enviar os dados
#define SERVER_PORT 34567           // Porta do servidor

#define BUTTON_PIN 5                // GPIO do botão
#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.

```c
//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 temso criticidade nos dados, optamos pelo mesmo pela simplicidade em codificar.

```c
// 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;
}
```

```c
// 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);
}
```

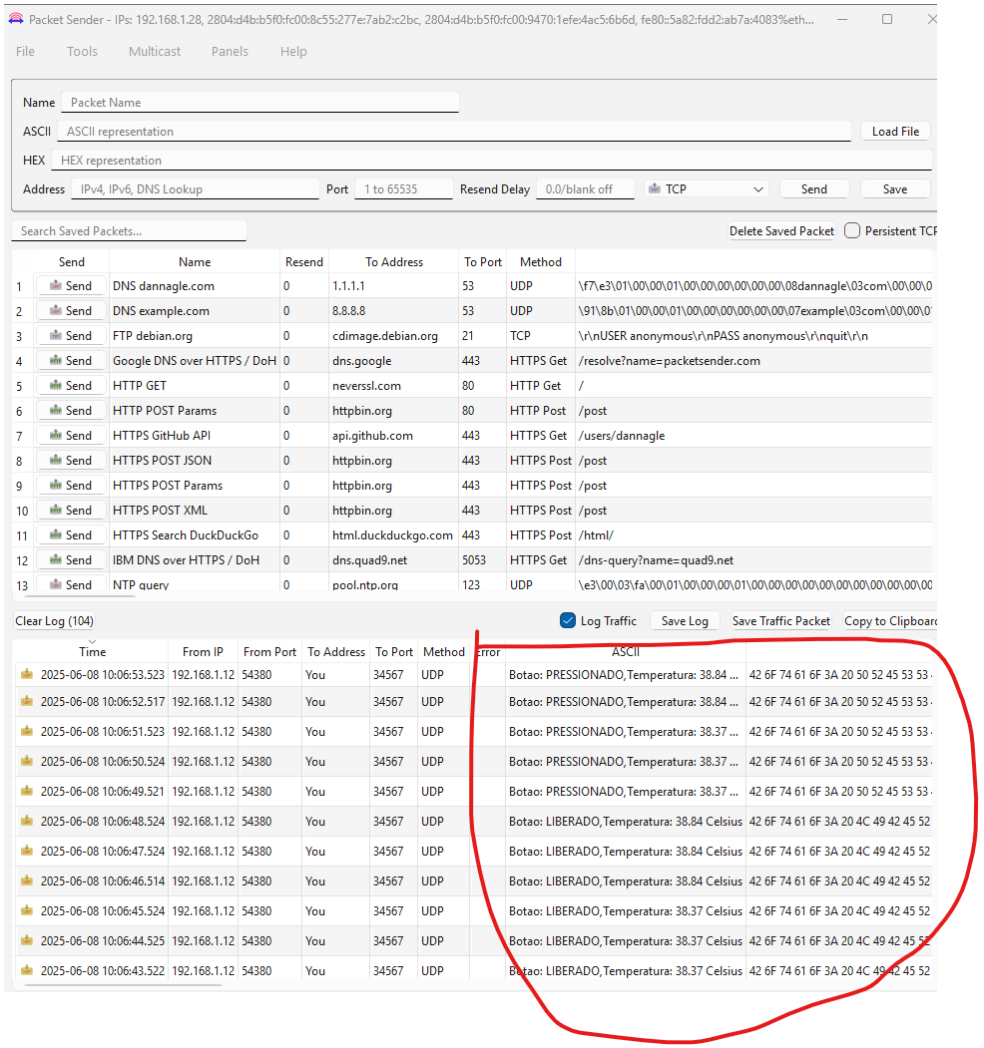
Abaixo o código desenvolvido para a função principal main.

```c
// 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;
    }
```

```c
    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.



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 imaginaria 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.

```c
#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 "xxxxxxxx"
#define WIFI_PASSWORD "xxxxxxxx"

// 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.

```c
// Prototipos 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.

```c
// 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";
}
```

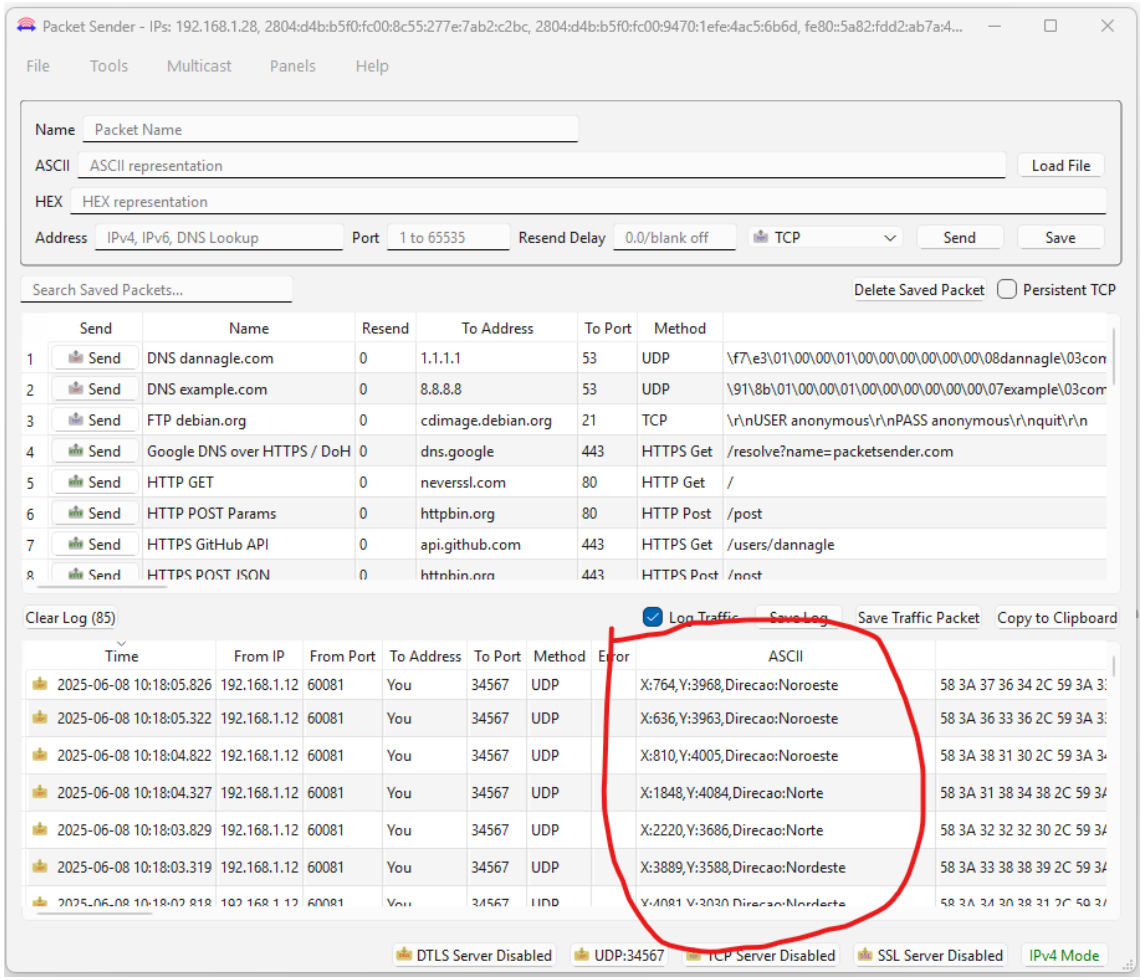
```c
// Função para enviar dados via UDP
void send_udp(struct udp_pcb *pcb, const char *msg) {
    struct pbuf *p = pbuf_alloc(PBUF_TRANSPORT, strlen(msg), PBUF_RAM);
    if (!p) return;
    memcpy(p->payload, msg, strlen(msg));

    ip_addr_t dest_ip;
    ipaddr_aton(SERVER_IP, &dest_ip);

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

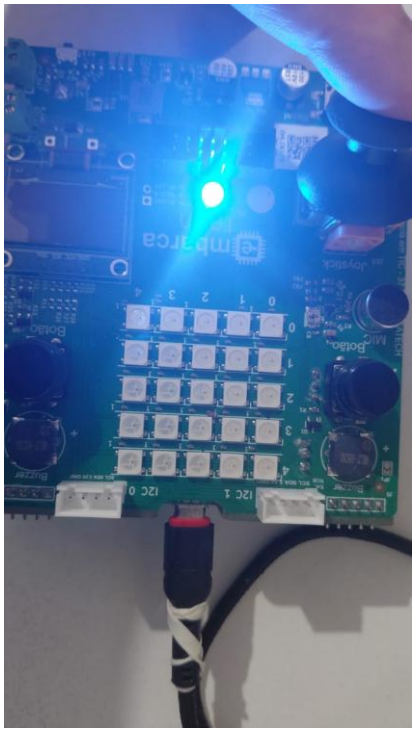
Escolhemos utilizar o protocolo UDP para enviarmos os dados já que não há criticidade no envio, assim não necessitamos de confirmação do envio. Na página seguinte temos nossa função principal, inicializando os periféricos, efetuando a conexão com WiFi, lendo as informações do joystick, convertendo em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

```c
// 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;
    }
```

```c
    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:





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_nuvem

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