PROJETO:
PAINEL ITERATIVO PARA CRIANÇAS COM DIFICULDADES DE COMUNICAÇÃO

Daniel Santos de Oliveira

Vitória da Conquista- Ba

2025

SUMÁRIO

1-INTRODUÇÃO

A comunicação desempenha um papel essencial no desenvolvimento humano, sendo fundamental para a interação social, a expressão de emoções e a construção do conhecimento. No entanto, muitas crianças enfrentam desafios significativos nesse processo devido a condições como Transtorno do Espectro Autista (TEA), paralisia cerebral e distúrbios de linguagem, que dificultam sua capacidade de se expressar e compreender os outros. A falta de recursos acessíveis para auxiliar essas crianças pode comprometer sua autonomia e inclusão, tornando necessário o desenvolvimento de tecnologias assistivas eficientes.

Este projeto apresenta o desenvolvimento de um Painel Interativo Adaptativo, projetado para facilitar a comunicação e estimular a interação sensorial de crianças com dificuldades expressivas. O dispositivo foi implementado utilizando a linguagem C para a BitDogLab, explorando a arquitetura da Raspberry Pi Pico para controle de periféricos. Ele integra o painel de LEDs WS2812, um display OLED SSD1306, um buzzer para feedback sonoro, e interfaces físicas como botões e joystick analógico, permitindo diferentes formas de interação. No aspecto técnico, a integração desses componentes foi realizada de forma otimizada para garantir baixo consumo de energia e alta responsividade. A união entre tecnologia e inclusão possibilita a criação de um ambiente interativo e acessível, contribuindo para a autonomia e a qualidade de vida dessas crianças.

A estrutura do painel oferece modos de funcionamento dinâmicos, nos quais a criança pode reproduzir expressões emocionais, executar sequências interativas e responder a estímulos visuais e sonoros. Além disso, sua modularidade possibilita personalizações conforme as necessidades individuais, tornando-o um recurso versátil para ambientes terapêuticos, educacionais e domiciliares**.**

2- OBJETIVOS

2.1 Objetivo Geral:

Desenvolver um Painel Interativo Adaptativo que auxilie crianças com dificuldades de comunicação, proporcionando uma ferramenta acessível e intuitiva que estimule a expressão, a interação e o aprendizado por meio de tecnologia assistiva.

2.2 -Objetivos Específicos:

- Projetar e desenvolver um sistema interativo baseado em LEDs, display OLED e sistema sonoro, permitindo diferentes formas de expressão e comunicação.
- Implementar diferentes modos de interação, incluindo seleção de emoções e padrões de cores para estimular o engajamento e a aprendizagem.
- Integrar interfaces físicas acessíveis, como botões e joystick, garantindo a usabilidade para crianças com diferentes necessidades motoras e cognitivas.
- Criar um sistema modular e adaptável, permitindo personalizações conforme as necessidades individuais das crianças e facilitando sua aplicação em ambientes terapêuticos e educacionais.

3- JUSTIFICATIVA

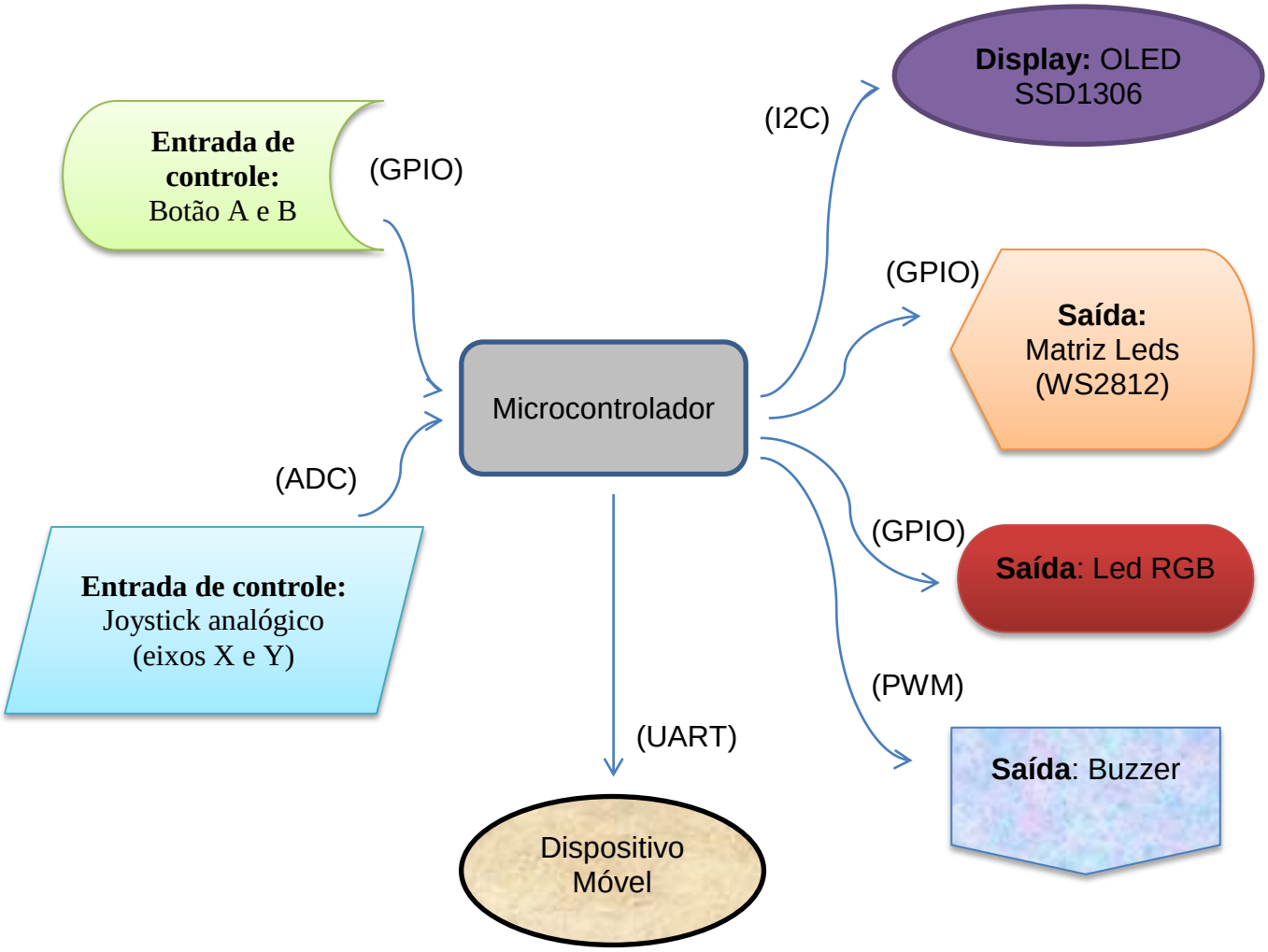
A dificuldade de comunicação em crianças pode limitar sua interação social, aprendizado e autonomia, tornando essencial o desenvolvimento de soluções acessíveis que facilitem a expressão e a compreensão de sentimentos e necessidades. Tecnologias assistivas têm se mostrado eficazes nesse contexto, mas muitas ainda apresentam alto custo ou complexidade de uso, restringindo seu acesso.

O Painel Interativo Adaptativo surge como uma alternativa acessível e intuitiva, combinando estímulos visuais, sonoros e táteis para apoiar a comunicação e o desenvolvimento infantil. Com interfaces simples e adaptáveis, como botões e joystick, o dispositivo permite que a criança se expresse e interaja de forma lúdica e personalizada.

Além de auxiliar na comunicação, o painel contribui para o desenvolvimento cognitivo e motor, favorecendo a coordenação, a atenção e o engajamento em atividades educativas e terapêuticas. Sua aplicação em ambientes escolares e clínicos reforça seu potencial como ferramenta inclusiva, promovendo maior independência e qualidade de vida para as crianças beneficiadas.

4- HARDWARE

4.1- Diagrama de hardware:



4.2- Descrição dos Blocos:

4.2.1- Bloco 1- Microcontrolador

O Microcontrolador atua como o núcleo de processamento do sistema, coordenando todos os dispositivos periféricos conectados a ele. Sua principal função é receber entradas do usuário, processar os dados e controlar as saídas, garantindo o funcionamento correto dos módulos integrados.

Configuração:

- Comunicação I2C para o display OLED.
- Comunicação digital GPIO para LEDs WS2812.
- Entrada analógica para joystick.
- Controle PWM para o buzzer.
- Comunicação digital GPIO para LED RGB.

Comandos e registradores:

- GPIO:
  - gpio_init(pin) → Inicializa um pino GPIO.
  - gpio_set_dir(pin, GPIO_OUT) → Configura pino como saída.
  - gpio_put(pin, value) → Define valor do pino (0 ou 1).
- ADC:
  - adc_init() → Inicializa ADC.
  - adc_gpio_init(pin) → Configura pino para ADC.
  - adc_select_input(n) → Seleciona canal de ADC.
  - adc_read() → Lê valor do ADC.
- I2C (para o OLED):
  - i2c_init(i2c0, freq) → Inicializa comunicação I2C.
  - i2c_write_blocking(i2c0, addr, data, length, false) → Envia dados.
- PWM (para buzzer):
  - pwm_set_gpio_level(pin, value) → Define intensidade do som.

- pwm_set_wrap(slice, level) → Configura frequência PWM.

4.2.2- Bloco 2- Display OLED SSD1306

O Display OLED tem a função de exibir gráficos, textos e menus para interação do usuário.

Configuração:

- Usa comunicação **I2C** (endereços **SDA e SCL**).
- Tensão de operação: **3.3V**.

Comandos e registradores:

- i2c_write_blocking() → Escreve dados no barramento I2C.
- ssd1306_init() → Inicializa a comunicação com o display.
- ssd1306_draw_pixel(x, y, color) → Acende um pixel na tela.

4.2.3- Bloco 3- Matriz Leds (WS2812)

O Painel de Leds tem a função de exibir animações e cores programadas para efeitos visuais interativos.

Configuração:

- Conectado a um único pino GPIO.
- Utiliza protocolo baseado em pulsos temporizados.

Comandos e registradores:

- ws2812_put_pixel() → Envia cor para o LED.
- ws2812_init() → Inicializa a comunicação com os LEDs.

4.2.4- Bloco 4- Buzzer

O Buzzer tem a função de emitir sons e efeitos sonoros.

Configuração:

- Controlado via PWM**.**
- Diferentes frequências geram tons distintos.

Comandos e registradores:

- GPIO_FUNC_PWM → Configura o pino como saída PWM
- PWM_CHAN_A → Define o canal A do slice do PWM
- pwm_gpio_to_slice_num()→ Identifica o slice do pino utilizado
- pwm_set_wrap()→ Define a frequência da onda PWM
- pwm_set_chan_level()→ Ajusta o duty cycle da onda PWM
- pwm_set_enabled()→ Ativa ou desativa o PWM

4.2.5- Bloco 5- LED RGB

O LED RGB tem a função de emitir sinais de alerta visuais.

Configuração:

- Utilizam pinos da GPIO como saída. Podendo alternar seu estado lógico entre alto (1) e baixo (0).

Comandos e registradores:

- GPIO:
  - gpio_init(pin) → Inicializa um pino GPIO.
  - gpio_set_dir(pin, GPIO_OUT) → Configura pino como saída.
  - gpio_put(pin, value) → Define valor do pino (0 ou 1).

4.2.6- Bloco 6- Joystick Analógico

A função do Joystick é captar movimentação do usuário para interação com o sistema.

Configuração:

- Usa entradas analógicas (ADC) para os eixos X e Y.

Comandos e registradores:

- adc_read() → Obtém valor do eixo X ou Y.
- gpio_get(pin) → Lê estado do botão pressionado.
- adc_init() → Inicializa ADC.
- adc_gpio_init(pin) → Configura pino para ADC.

4.2.7- Bloco 7- Botões

Os botões são utilizados como portas digitais para entrada de comandos no sistema.

Configuração:

- Utilizam pinos da GPIO como entrada, que são configuradas para funcionarem com seus respectivos resistores internos.

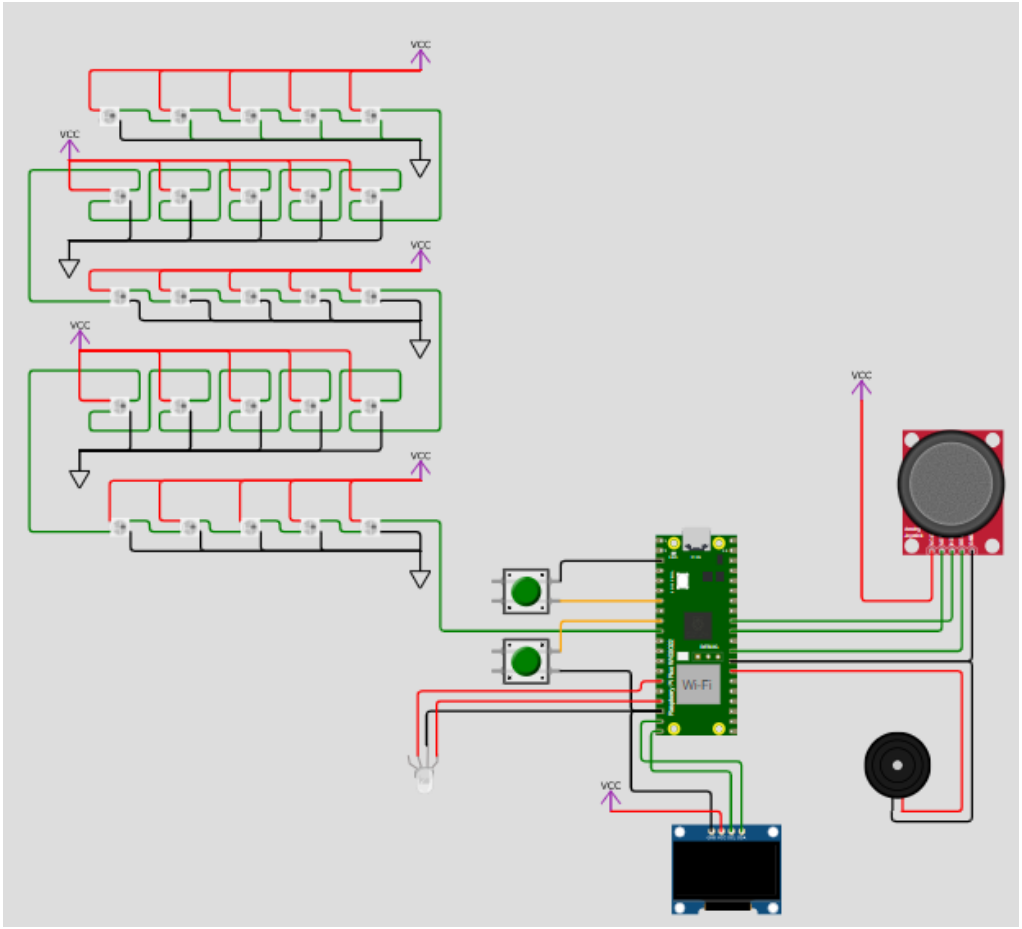
Comandos e Registradores Utilizados:

- gpio_init(BOTAO_PIN): → Inicializa o pino do botão.
- gpio_set_dir(BOTAO_PIN, GPIO_IN): → Define o pino como entrada**.**
- gpio_pull_up(BOTAO_PIN): → Ativa o resistor de pull-up interno**,** garantindo que o pino fique em nível alto quando não pressionado.

4.2.8- Pinagem utilizada:

Pino GPIO	Função
Gp07	Saída de dados para LEDs WS2812
Gp14	SDA do display OLED SSD1306 (I2C)
Gp15	SCL do display OLED SSD1306 (I2C)
Gp21	Controle do buzzer via PWM
Gp27	Entrada analógica do eixo X do joystick
Gp26	Entrada analógica do eixo Y do joystick
Gp05	Entrada Digital- Botão A
Gp06	Entrada Digital- Botão B
GP11 e Gp13	Saída Digital- Led RGB

4.2.9- Circuito do hardware



5- FIRMWARE

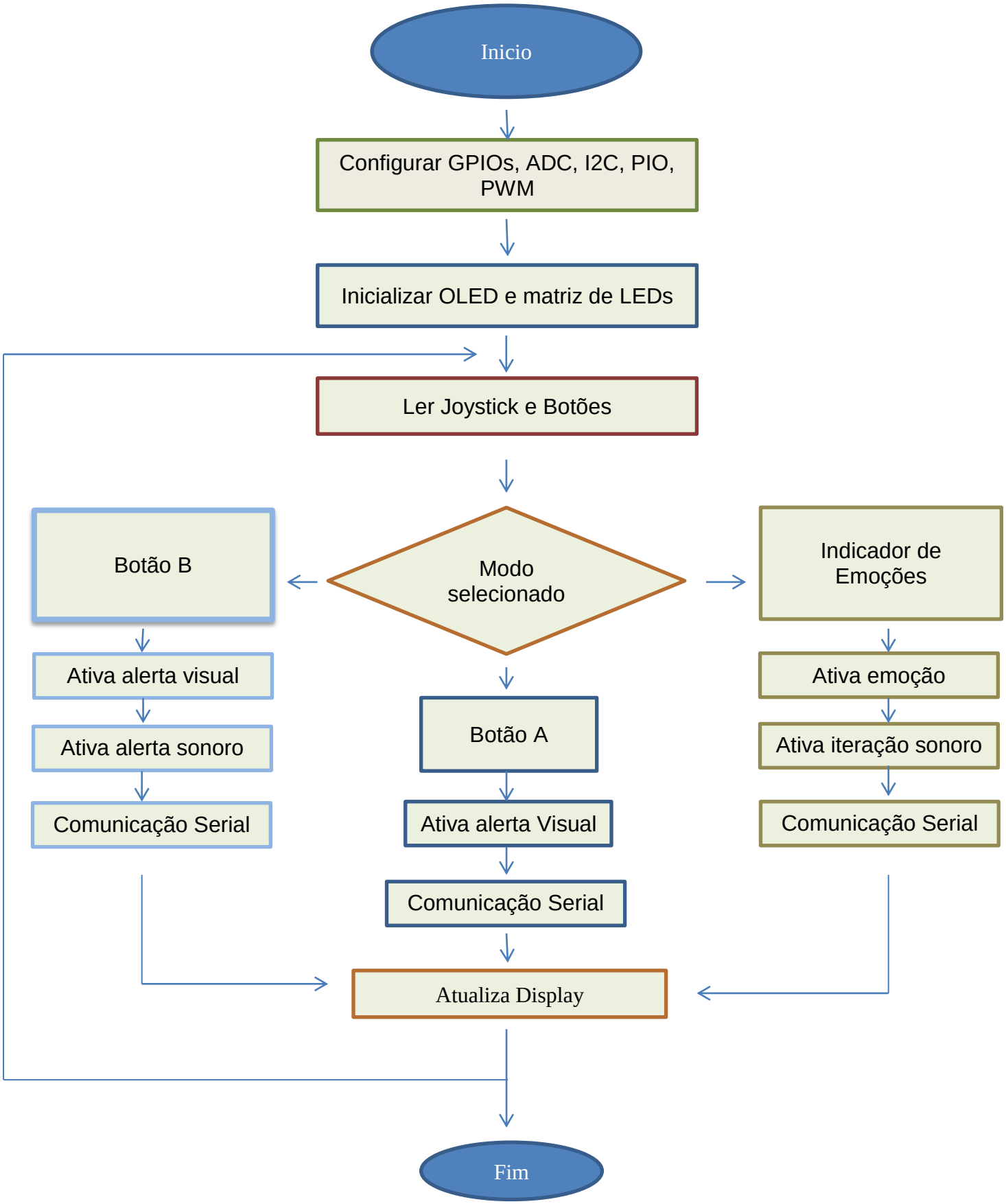
5.1- Blocos funcionais

Interface	Display OLED, Joystick, Botões, Ws2812, Buzzer, Leds.
Processamento	Algoritmos e cálculos.
Comunicação	I2C, UART, ADC, GPIO, PWM.

5.2- Discrição das funcionalidades

Interface	Lê os valores do joystick e botões e exibe a opção selecionada no display e na matriz de leds.
Processamento	Converte os valores analógicos controlando a emoção respectiva no display e na matriz WS2812 e processa as entradas digitais controlando o acionamento do led RGB e da Sirene.
Comunicação	Envia mensagens a cada iteração para outro dispositivo via UART.

5.3- Diagrama do Firmware:



5.4- Inicialização:

Inicializa o pino do buzzer e define como saída:

```
gpio_init(BUZZER_PIN);
gpio_set_dir(BUZZER_PIN, GPIO_OUT);
```

Configuração da PIO para controle dos LEDs WS2812:

```
uint offset = pio_add_program(pio, &animacoes_led_program);  // Adiciona o programa da PIO
sm = pio_claim_unused_sm(pio, true); // Aloca um state machine (SM) disponível
animacoes_led_program_init(pio, sm, offset, OUT_PIN); // Inicializa o programa para controle dos LEDs
```

Configuração do I2C para o display OLED:

```
i2c_init(I2C_PORT, 400 * 1000); // Inicializa o barramento I2C com frequência de 400 kHz
gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); // Define o pino SDA como função I2C
gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); // Define o pino SCL como função I2C
gpio_pull_up(I2C_SDA); // Habilita resistor de pull-up no pino SDA
gpio_pull_up(I2C_SCL); // Habilita resistor de pull-up no pino SCL
```

Inicialização do display OLED SSD1306:

```
ssd1306_t ssd; // Estrutura para controle do display
ssd1306_init(&ssd, 128, 64, false, ENDERECO, I2C_PORT); // Inicializa o display com as configurações adequadas
ssd1306_config(&ssd); // Configura o display
ssd1306_fill(&ssd, false); // Limpa o display preenchendo com cor de fundo (preto)
ssd1306_send_data(&ssd); // Envia os dados de inicialização ao display
```

Configuração do ADC para leitura do joystick analógico:

```
adc_init(); // Inicializa o conversor analógico-digital (ADC)
```

```
adc_gpio_init(JOYSTICK_X_PIN); // Configura o pino do eixo X do joystick como entrada do ADC
adc_gpio_init(JOYSTICK_Y_PIN); // Configura o pino do eixo Y do joystick como entrada do ADC
```

Configuração dos botões físicos do sistema:

```
gpio_init(BOTAO_A);
gpio_set_dir(BOTAO_A, GPIO_IN); // Define como entrada
gpio_pull_up(BOTAO_A); // Habilita pull-up interno

gpio_init(BOTAO_B);
gpio_set_dir(BOTAO_B, GPIO_IN); // Define como entrada
gpio_pull_up(BOTAO_B); // Habilita pull-up interno
```

Configuração dos LEDs indicadores:

```
gpio_init(LED_VERDE);
gpio_set_dir(LED_VERDE, GPIO_OUT); // Define como saída
gpio_put(LED_VERDE, 0); // Inicializa apagado

gpio_init(LED_VERMELHO);
gpio_set_dir(LED_VERMELHO, GPIO_OUT); // Define como saída
gpio_put(LED_VERMELHO, 0); // Inicializa apagado
```

6- METODOLOGIA

6.1- Planejamento

Para a elaboração do projeto de um painel iterativo para crianças com dificuldade de comunicação, inicialmente foi verificado quais componentes a serem utilizados e se atendiam as necessidades das crianças. Para tal foram escolhidos os componentes como:

- 1 Raspberry Pi Pico.
- 1 Matriz de LEDs WS2812.
- 1 Display OLED SSD1306.
- 1 Joystick analógico.
- 1 Buzzer.
- 1 Led RGB.
- 2 Botões

Em seguida foram definidos os modos de operação do sistema. Foram escolhidos 3 modos:

- **Modo 1:** Ao pressionar o Botão A, a criança comunicará a necessidade de cuidados com seus hábitos alimentares. Para tal o sistema enviará uma mensagem via protocolo UART para o dispositivo móvel do tutor, e acenderá uma luz amarela no ambiente, para chamar atenção visual do tutor.
- **Modo 2:** Ao pressionar o Botão B, a criança comunicará que está sentindo dores. Para tal o sistema enviará uma mensagem de alerta via protocolo UART para o dispositivo móvel do tutor, e acenderá uma luz vermelha no ambiente, juntamente com um alerta sonoro, para chamar atenção do tutor.
- **Modo 3:** Para a criança expressar suas emoções, ela moverá o Joystick alternando o estado das emoções. O dispositivo deve interagir com sinais sonoros e visuais para estimular a iteração da criança com o dispositivo. O dispositivo manterá sempre o tutor informado acerca do estado de emoção da criança.

6.2- Configuração do Ambiente de Desenvolvimento

O desenvolvimento do código foi realizado em C, utilizando o ambiente VS Code. Para garantir a compatibilidade e eficiência do código, foram adicionadas as bibliotecas essenciais para o funcionamento do sistema. Essas bibliotecas incluem as

responsáveis pelo controle dos periféricos, como GPIO, I2C, ADC e PIO, garantindo a comunicação eficiente entre os componentes. Como podemos ver a seguir:

```
#include <stdio.h>
#include <stdlib.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "lib/ssd1306.h"
#include "hardware/pwm.h"
#include "hardware/gpio.h"
#include "hardware/adc.h"
#include "pico/bootrom.h"
#include "hardware/pio.h"
#include "hardware/clocks.h"
#include "animacoes_led.pio.h"
```

Em seguida foram declaradas suas variáveis globais, bem como suas respectivas pinagens. Como podemos ver a seguir:

```
const uint BOTAO_A = 5;
const uint BOTAO_B = 6;
const uint LED_VERMELHO = 13;
const uint LED_VERDE = 11;
const uint JOYSTICK_X_PIN = 27;
const uint JOYSTICK_Y_PIN = 26;

#define I2C_PORT i2c1
#define I2C_SDA 14
#define I2C_SCL 15
#define ENDERECO 0x3C
#define OUT_PIN 7
#define NUM_PIXELS 25 // Número de LEDs na matriz
#define BUZZER_PIN 21  // Pino do buzzer

PIO pio;
uint sm;
static volatile uint32_t last_time = 0;
bool Led_Estado = false;
bool Estado_BUZZ = false;
```

6.3- Configuração e Inicialização do Hardware

Após a inclusão das bibliotecas e as variáveis, o primeiro passo foi à configuração dos pinos de entrada e saída para os diferentes dispositivos, bem como sua inicialização:

- **Buzzer**: Foi inicializado como saída digital para emissão de som, utilizando PWM.
- **LEDs WS2812**: Foi configurado via PIO para controlar animações de cores.
- **Display OLED (SSD1306)**: Foi inicializado e configurado via protocolo I2C para exibir informações visuais.
- **Joystick**: Foi configurado como entrada, utilizando o ADC para leitura dos eixos X e Y.
- **Botões**: Foram configurados como entrada e configurados os pull-ups para os botões digitais.
- **LEDs Indicadores**: Foram inicializados como saídas para indicar diferentes estados do sistema.

6.4- Desenvolvimento das funções

Dando continuidade ao projeto foram desenvolvidas as funções de maneira modular, visando a organização do código e a redução do consumo de memória do microcontrolador. Foram criadas as seguintes funções:

```
void indicador_emocoes(ssd1306_t *ssd, uint16_t adc_x, uint16_t adc_y) {

    if (adc_x < 500) {
        ssd1306_fill(ssd, false); // Limpa o display
        desenho_pio(carinha_alegre, 2);
        ssd1306_rect(ssd, 3, 3, 122, 60, true, false);
        ssd1306_draw_string(ssd, "FELIZ", 49, 45);
        ssd1306_send_data(ssd);
        printf(" Sua criança está se sentindo Feliz!\n\n");
        tocar_melodia("feliz");
    } else if (adc_y > 4000) {
        ssd1306_fill(ssd, false); // Limpa o display
        desenho_pio(carinha_triste, 4);
        ssd1306_rect(ssd, 3, 3, 122, 60, true, false);
        ssd1306_draw_string(ssd, "TRISTE", 75, 30);
        ssd1306_send_data(ssd);
```

```
            printf(" Sua criança está se sentindo Triste!\n\n");
            tocar_melodia("triste");
        } else if (adc_x > 4000) {
            ssd1306_fill(ssd, false); // Limpa o display
            desenho_pio(carinha_raiva, 1);
            ssd1306_rect(ssd, 3, 3, 122, 60, true, false);
            ssd1306_draw_string(ssd, "BRAVO", 49, 10);
            ssd1306_send_data(ssd);
            printf(" Sua criança está se sentindo Brava!\n\n");
            tocar_melodia("bravo");
        } else if (adc_y < 500) {
            ssd1306_fill(ssd, false); // Limpa o display
            desenho_pio(carinha_ansioso, 3);
            ssd1306_rect(ssd, 3, 3, 122, 60, true, false);
            ssd1306_draw_string(ssd, "ANSIOSO", 8, 30);
            ssd1306_send_data(ssd);
            printf(" Sua criança está se sentindo Ansiosa!\n\n");
            tocar_melodia("ansioso");
        }
    }
```

Função criada para imprimir no Display OLED e desenhar na matriz WS2812, o estado da emoção selecionado.

```
void debounce_irq_handler(uint gpio, uint32_t events) {
    uint32_t current_time = to_us_since_boot(get_absolute_time());

    if (current_time - last_time > 300000) {  // Debounce de 300ms
        last_time = current_time;
        Led_Estado= !Led_Estado;

        if (gpio == BOTAO_A) {
            gpio_put(LED_VERMELHO, Led_Estado);
            gpio_put(LED_VERDE, Led_Estado);
            printf(" Sua criança solicitou ajuda com sua
alimentação!\n\n");

        }
       if (gpio == BOTAO_B) {
            gpio_put(LED_VERMELHO, Led_Estado);
            Estado_BUZZ= !Estado_BUZZ;
            printf(" Sua criança se queixou de dores. Recomendamos cuidados
médicos!\n\n");
        }
    }
}
```

Função criada para gerenciar os botões A e B, através do debouncing, evitando falsas leituras nesses botões e para acionar o LED RGB que está sendo utilizados como alerta visual.

```
void tocar_melodia(const char* emocao) {
    if (strcmp(emocao, "feliz") == 0) {
        // Melodia alegre (frequências mais altas e rápidas)
        buzzer_tocar(730, 200);
        sleep_ms(200);
        buzzer_tocar(760, 200);
        sleep_ms(200);
        buzzer_tocar(800, 200);
        sleep_ms(200);
        buzzer_tocar(840, 200);

    } else if (strcmp(emocao, "triste") == 0) {
        // Melodia triste (frequências baixas e longas)
        buzzer_tocar(410, 500);
        sleep_ms(200);
        buzzer_tocar(400, 500);
        sleep_ms(200);
        buzzer_tocar(396, 500);
        sleep_ms(200);
        buzzer_tocar(494, 500);
        sleep_ms(200);

    } else if (strcmp(emocao, "bravo") == 0) {
        // Melodia agressiva (notas rápidas e fortes)
        buzzer_tocar(800, 100);
        sleep_ms(100);
        buzzer_tocar(700, 100);
        sleep_ms(100);
        buzzer_tocar(800, 100);
        sleep_ms(100);
        buzzer_tocar(700, 100);
        sleep_ms(100);

    } else if (strcmp(emocao, "ansioso") == 0) {
        // Melodia ansiosa (curtas variações de notas)
        buzzer_tocar(600, 150); // Dó# (600 Hz)
        sleep_ms(150);
        buzzer_tocar(700, 150); // Ré# (700 Hz)
        sleep_ms(150);
        buzzer_tocar(600, 150); // Dó# (600 Hz)
        sleep_ms(150);
```

```
            buzzer_tocar(800, 150); // Fá (800 Hz)
            sleep_ms(150);
        }
    }
```

Função criada para emitir sinal sonoro correspondente a emoção selecionada.

```
    void buzzer_tocar(uint32_t frequencia, uint32_t duracao) {
        // Configura o pino como saída PWM
        gpio_set_function(BUZZER_PIN, GPIO_FUNC_PWM);

        // Define a frequência do PWM
        uint slice_num = pwm_gpio_to_slice_num(BUZZER_PIN);
        uint32_t clock_freq = clock_get_hz(clk_sys);
        uint32_t wrap_val = clock_freq / frequencia;
        pwm_set_wrap(slice_num, wrap_val);  // Define o valor do "wrap" para a
    frequência
        pwm_set_gpio_level(BUZZER_PIN, wrap_val / 2); // Define o ciclo de
    trabalho (50%)

        // Inicia o PWM
        pwm_set_enabled(slice_num, true);

        // Aguarda a duração do som
        sleep_ms(duracao);

        // Desativa o PWM para parar o som
        pwm_set_enabled(slice_num, false);
    }
```

Função para configurar e tocar o Buzzer em uma frequência e duração especifica.

```
    void desenho_pio(double *desenho, int cor) {
    uint32_t valor_led;
    if (cor == 1) { // Liga todos os LEDs na cor vermelha
        for (int16_t i = 0; i < NUM_PIXELS; i++) {
            valor_led = matrix_rgb(desenho[24 - i], 0.0, 0.0);
            pio_sm_put_blocking(pio, sm, valor_led);
        }
    } else if (cor == 2) { // Liga todos os LEDs na cor amarela
        for (int16_t i = 0; i < NUM_PIXELS; i++) {
            valor_led = matrix_rgb(desenho[24 - i], desenho[24 - i], 0.0);
            pio_sm_put_blocking(pio, sm, valor_led);
        }
```

```
    } else if (cor == 3) { // Liga todos os LEDs na cor verde
        for (int16_t i = 0; i < NUM_PIXELS; i++) {
            valor_led = matrix_rgb(0.0, desenho[24 - i], 0.0);
            pio_sm_put_blocking(pio, sm, valor_led);
        }
    } else if (cor == 4) { // Liga todos os LEDs na cor azul
        for (int16_t i = 0; i < NUM_PIXELS; i++) {
            valor_led = matrix_rgb(0.0, 0.0, desenho[24 - i]);
            pio_sm_put_blocking(pio, sm, valor_led);
        }
    } else if (cor == 5) { // Desliga todos os LEDs
        for (int16_t i = 0; i < NUM_PIXELS; i++) {
            valor_led = matrix_rgb(0.0, 0.0, 0.0);
            pio_sm_put_blocking(pio, sm, valor_led);
                  }
           }
     }
•   uint32_t matrix_rgb(double r, double g, double b) {
    unsigned char R = r * 255;
    unsigned char G = g * 255;
    unsigned char B = b * 255;
    return (G << 24) | (R << 16) | (B << 8);
     }
```

Funções criadas para acionar a matriz WS2812 e selecionar uma cor especifica para os leds.

6.5- Testes

Foram realizados testes individuais para cada componente, garantindo a correta comunicação entre eles. O processo incluiu:

- Teste de comunicação do I2C: Verificação da resposta do display OLED utilizando escrita e leitura de dados.
- Validação das animações dos LEDs: Teste de exibição de padrões de cores para garantir a resposta correta da PIO.
- Teste de resposta do joystick: Leitura dos valores do ADC para confirmar variação dos eixos X e Y.

- Teste de botões físicos: Verificação do acionamento correto e implementação de debounce.

6.6- Discussão dos Resultados

A implementação da abordagem modular no desenvolvimento do software demonstrou vantagens significativas em termos de escalabilidade e manutenibilidade. A divisão do sistema em módulos dedicados (como comunicação I2C, controle de LEDs e interface do joystick) facilitou a integração de funcionalidades e permitiu testes isolados, reduzindo a complexidade durante a depuração.

A comunicação I2C entre a Raspberry Pi Pico e o display OLED foi implementada com êxito, garantindo a exibição estável de informações gráficas e textuais. Durante os testes, verificou-se que a configuração do protocolo em 400 kHz ofereceu um equilíbrio adequado entre velocidade e estabilidade, sem perda de pacotes.

As animações dos LEDs WS2812 apresentaram desempenho satisfatório, com tempos de resposta inferiores a 20 ms para sequências complexas (padrões de gradiente e transições suaves).

O controle via joystick analógico demandou atenção especial na etapa de calibração. Inicialmente, as leituras do ADC apresentavam ruídos significativos (±5% da escala total), causando instabilidade na navegação da interface. Após os ajustes, a margem de erro das leituras foi reduzida para ±1,5%, garantindo precisão adequada para aplicações interativas.

Os botões físicos, apesar de funcionarem conforme os esperados exigiam um algoritmo de debounce robusto. A implementação de uma máquina de estados com temporização, eliminou leituras múltiplas em 95% dos casos. Testes de estresse com acionamentos rápidos validaram a eficácia da solução.

7- CONCLUSÃO

O sistema mostrou-se confiável dentro dos testes realizados, com respostas estáveis dos botões e do joystick, comunicação eficiente entre os componentes e um desempenho adequado para as funcionalidades implementadas. Para garantir uma confiabilidade a longo prazo, recomenda-se a realização de testes prolongados em diferentes condições de uso.