

**Aluno:** Devid Henrique Pereira dos Santos

**Matrícula:** TIC370100380

## **Projeto Final: Sensor de nível para tanques/reservatórios**

### **1 - Descrição do projeto**

O projeto em questão consiste em um sensor de nível de água para tanques e/ou reservatórios, que sinaliza por meio de uma matriz de LEDs e pelo display OLED o nível da água, este último, inclusive, sinalizando tanto barras cuja a cor indica o nível aproximada, quando pelo valor em percentual do reservatório. Além disso, uma mensagem no display descreve a situação atual (cheio, médio, vazio). Os LEDs RGB também possuem papel importante na sinalização, indicando por meio da luz verde um volume alto no reservatório, amarelo um volume baixo e em vermelho, piscando e ao som de um alerta emitido pelo buzzer o nível crítico no reservatório.

### **2 - Objetivos**

Desenvolvimento de um sistema embarcado de baixo custo para aferir o nível de um reservatório e sinalizar por meio de alerta visual e sonoro se o volume estiver crítico.

### **3 - Justificativa**

Em muitas situações não existe como determinar com precisão o nível de reservatórios, seja pela sua grande capacidade, o que torna desafiador a sua aferição manual, fato que pode pegar de surpresa aqueles que convivem com situações onde períodos de seca são extensivos. Deste modo, voltado para o público residencial, sobretudo em regiões agrárias e com pouco favorecidas pelo poder público, surge o projeto para um medidor de nível para reservatórios, capaz de emitir alertas visuais e intuitivos ao usuário, com fácil instalação.

### **4 - Projetos correlatas**

Em uma busca pela internet é possível encontrar diversos projetos para medir o nível de água. Destaco aqui o artigo publicado no site Hacker.io por **Shafin Kothia** intitulado ***Water Level Monitor with Raspberry Pi*** [1], consiste em um projeto simples que se utiliza de um sensor ultrassônico para verificar o nível da água e um buzzer capaz de disparar um sinal sonoro quando o nível está quase cheio. O outro projeto, com uma abordagem um pouco distinta, intitulado ***Raspberry Pi Pico Water-Level-Indicator Using Potentiometer*** publicado no site **Instructable.com**, utiliza-se de uma bóia conectada ao potenciômetro. Conforme o nível da água sobe e, conseqüentemente, a bóia move o potenciômetro, um conjunto de LEDs verde, amarelo e vermelho, sinaliza o nível da água. Quando o nível máximo é atingido no recipiente o buzzer dispara.

## 5 - Especificação do hardware

O projeto foi realizado na placa de desenvolvimento **BitDogLab**, especialmente desenvolvida para o ensino de sistemas embarcados e que possui como ponto central a Raspberry Pi Pico. Conta com um conjunto de periféricos que possibilita uma ampla gama de projetos e flexibilidade por parte do usuário para criação de soluções embarcadas diversas. Para o presente projeto foram utilizados os seguintes periféricos presentes na placa **BitDogLab** e suas respectivas GPIOs:

**Buzzer A:** conectado a **GPIO 21**, emite um sinal sonoro ajustado com o PWM..

**LED RGB:** conectados às **GPIOs 13, 12 e 11** corresponde, respectivamente, aos LEDs vermelho, azul e verde.

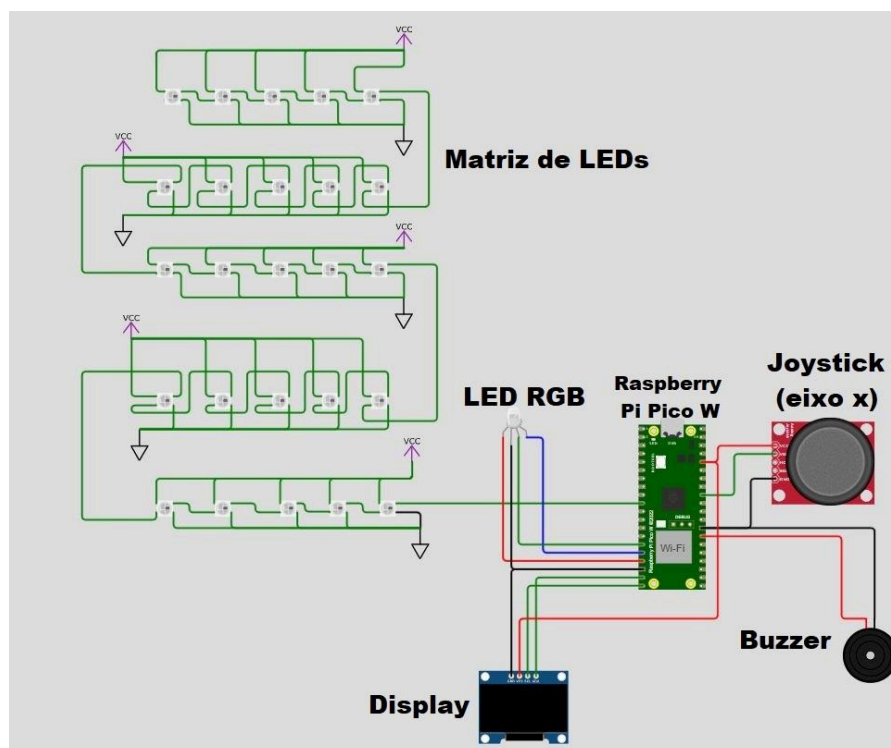
**Matriz de LEDs:** conectado a **GPIO 7**, consiste em um conjunto de 25 LEDs formando uma matriz 5x5 onde podemos representar de maneira visual informações do projeto..

**Joystick (eixo y):** conectado a **GPIO 26**, consiste em um potenciômetro que se move no eixo y, o seu valor analógico é lido e posteriormente convertido em um sinal digital.

**SDA (Serial Data):** Conectado a **GPIO 14**, é responsável por transferir e receber informações do barramento I2C do display OLED.

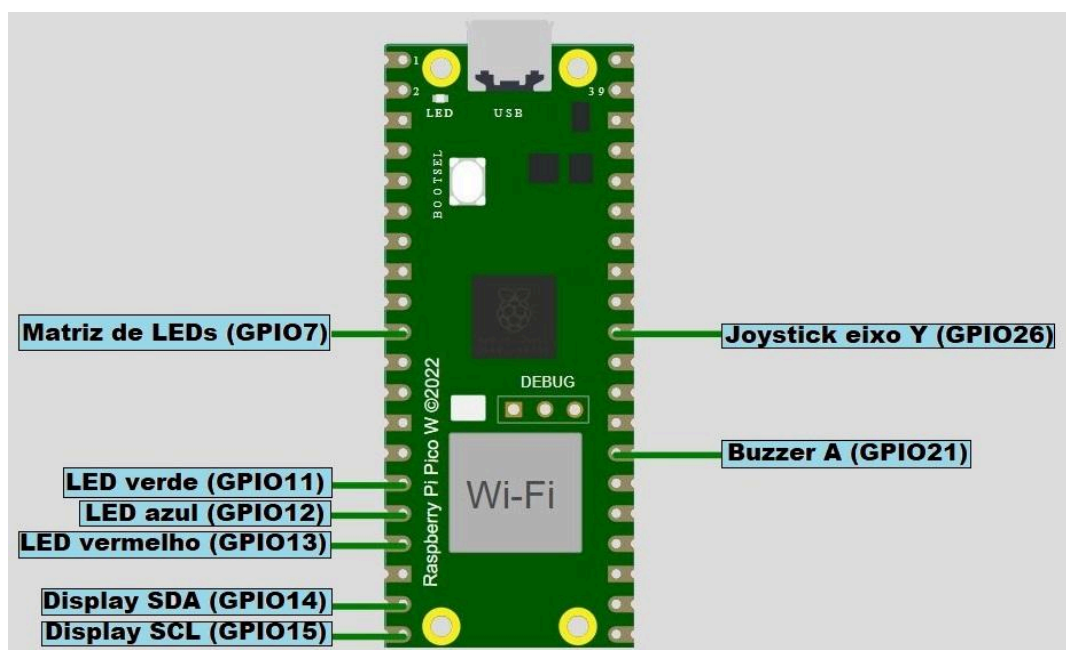
**SCL:** (Serial Clock): Conectado a **GPIO 15**, é responsável por sincronizar o clock da comunicação do display ao barramento I2C.

Abaixo é possível observar as conexões (simplificadas) dos dispositivos na Raspberry.



Esquema de ligação dos periféricos na **Raspberry Pi Pico W** no Wokwi.

As pinagens correspondentes na Raspberry Pi Pico W podem ser vistas no diagrama abaixo.



Pinagem de cada periférico na Raspberry Pi Pico W.

## 6 - Bibliotecas

As bibliotecas constituem uma parte fundamental do desenvolvimento do software. Por meio das bibliotecas é possível utilizar um conjunto de funções específicas para atender as necessidades do projeto. Para o presente projeto as seguintes bibliotecas foram utilizadas:

**“pico/stdlib.h”** : Biblioteca específica da Raspberry Pi Pico fornecida pela Pico SDK, é responsável pela manipulação de GPIOs, inicialização do sistema, delays, dentre outros.

**“hardware/adc.h”**: Biblioteca do Raspberry Pi Pico usada para Conversor Analógico-Digital (ADC), permitindo que seja feita a leitura dos pinos analógicos conectados.

**“hardware/i2c.h”** : Biblioteca que permite a comunicação com dispositivos i2C, como é o caso do display OLED.

**“display/ssd1306.h”**: Biblioteca criada para o controle do display OLED SSD1306 via i2c, sendo assim possível a visualização de imagens, textos e figuras geométricas no display.

**“display/fonte.h”**: Biblioteca que define o conjunto de caracteres para serem exibidos no display SSD1306, como as letras do alfabeto (em maiúsculo) e os números de 0 a 9.

**“matriz/neopixel.h”**: Biblioteca usada para controlar os LEDs WS2812 (matriz de LEDs).

“**buzzer/buzzer.h**”: Biblioteca usada para controlar o buzzer, capaz de emitir sons de diferentes frequências, auxiliada pela biblioteca “**hardware/pwm**”, a qual está inserida.

“**hardware/pwm**” : Biblioteca responsável pela Modulação por Largura de Pulso (PWM) para as saídas GPIOs, permitindo controlar a intensidade de LEDs e buzzers.

“**hardware/clocks.h**”: Biblioteca que permite configurar e manipular os clocks no microcontrolador RP2040.

“**hardware/uart**” : Biblioteca responsável por fornecer a interface para comunicação serial UART.

## 7 - Funções do software

O programa conta com algumas funções desenvolvidas em **linguagem C**, tanto para configuração de funções mais gerais, como do display e LEDs, como algumas criadas especialmente para concatenar informações e permitir um código mais limpo. É imprescindível e considera-se uma boa prática de programação que todo o código seja comentado, facilitando o entendimento de quem o lerá. Dentre as funções criadas podemos destacar:

**Função *setup\_leds***: função do tipo **void**, portanto não retorna nenhum valor, bem como não recebe nenhum parâmetro, é responsável por inicializar o LED RGB e definindo-os como uma saída GPIO, além de inicializar a matriz de LEDs.

```
void setup_leds(){  
  
    stdio_init_all();  
    gpio_init(LED_RED);           // Inicializa a GPIO do LED vermelho  
    gpio_init(LED_BLUE);          // Inicializa a GPIO do LED azul  
    gpio_init(LED_GREEN);         // Inicializa a GPIO do LED verde  
    gpio_set_dir(LED_RED, GPIO_OUT); // Define o LED vermelho como uma saída GPIO  
    gpio_set_dir(LED_BLUE, GPIO_OUT); // Define o LED azul como uma saída GPIO  
    gpio_set_dir(LED_GREEN, GPIO_OUT); // Define o LED verde com uma saída GPIO  
    npInit(LED_PIN);              // Inicializa a matriz de LEDs  
    npClear();  
  
}
```

**Função *setup\_display***: função do tipo **void**, sendo assim não retorna nenhum valor nem recebe parâmetros, é responsável por inicializar e configurar o display na função I2C da Raspberry Pico. Os pinos SDA e SCL são colocados em nível alto (pull up).

```
void setup_display(){

    i2c_init(I2C_PORT, 400 * 1000);
    gpio_set_function(I2C_SDA, GPIO_FUNC_I2C); // Configura o pino do SDA na função I2C
    gpio_set_function(I2C_SCL, GPIO_FUNC_I2C); // Configura o pino da SCL na função I2C
    gpio_pull_up(I2C_SDA); // Configura a SDA como um pull up
    gpio_pull_up(I2C_SCL); // Configura a SCL como um pull up

    ssd1306_init(&ssd, WIDTH, HEIGHT, false, ENDereco, I2C_PORT); // Inicializa o display
    ssd1306_config(&ssd); // Configura o display
    ssd1306_send_data(&ssd); // Envia os dados para o display

    // Limpa o display. O display inicia com todos os pixels apagados.
    ssd1306_fill(&ssd, false);
    // Envia os dados para o display
    ssd1306_send_data(&ssd);
}
```

**Função *message\_display*** : Consiste em uma função do tipo **void**, portanto, não retorna valor, recebendo dois vetores do tipo **char**: um para imprimir uma mensagem no display e outro para imprimir o valor em porcentagem do ADC do joystick. Além disso, a função recebe como parâmetro 5 valores do tipo **bool** responsáveis por indicar a barra de nível no display (semelhante a barra de carregamento dos smartphones), correspondente aos níveis 100, 80, 60, 40 e 20. Esta função configura o layout do display.

```
void message_display(char c[5], char valor[5], bool cor_20, bool cor_40, bool cor_60, bool cor_80, bool cor_100){

    ssd1306_fill(&ssd, !cor); // Limpa o display
    ssd1306_rect(&ssd, 3, 3, 124, 60, cor, !cor); // Desenha um retângulo externo
    ssd1306_line(&ssd, 3, 14, 123, 14, cor); // Desenha uma linha horizontal
    ssd1306_draw_string(&ssd, c, 16, 6); // Escreve uma mensagem no display
    ssd1306_draw_string(&ssd, valor, 20, 32); // Desenha uma string
    ssd1306_rect(&ssd, 15, 66, 12, 7, cor, cor_20); // Retângulo tamanho 20%
    ssd1306_rect(&ssd, 25, 66, 20, 7, cor, cor_40); // Retângulo tamanho 40%
    ssd1306_rect(&ssd, 35, 66, 36, 7, cor, cor_60); // Retângulo tamanho 60%
    ssd1306_rect(&ssd, 45, 66, 48, 7, cor, cor_80); // Retângulo tamanho 80%
    ssd1306_rect(&ssd, 55, 66, 60, 7, cor, cor_100); // Retângulo tamanho 100%
    ssd1306_send_data(&ssd); // Atualiza o display
}
```

**Função *led\_alerta***: função do tipo **void**, portanto não retorna valor, é responsável por piscar o LED vermelho, para sinalizar alerta do nível crítico no reservatório.

```
void led_alerta(){

    gpio_put(LED_RED, true); // LED vermelho aceso
    sleep_ms(500); // Espera por 500 ms
    gpio_put(LED_RED, false); // LED vermelho apagado
    sleep_ms(500); // Espera por 500 ms
}
```

**Função *leds\_turn\_on***: função do tipo **void**, não retorna valor, consiste numa função que recebe como parâmetro três variáveis do tipo **bool** que definem quais dos LEDs RGB serão acesos.



```
void leds_turn_on(bool light_red, bool light_blue, bool light_green){
    gpio_put(LED_RED, light_red);
    gpio_put(LED_BLUE, light_blue);
    gpio_put(LED_GREEN, light_green);
}
```

**Função *level*:** Consiste em um conjunto de 5 funções do tipo void, responsáveis pela iluminação dos LED da matriz de acordo com o nível do ADC do joystick.

```
// Função para exibir o nível quatro na matriz de LEDs
void level_quatro(){
    npSetLED(0,0,0,10);
    npSetLED(1,0,0,10);
    npSetLED(8,0,0,10);
    npSetLED(2,0,0,10);
    npSetLED(7,0,0,10);
    npSetLED(12,0,0,10);
    npSetLED(3,0,0,10);
    npSetLED(6,0,0,10);
    npSetLED(13,0,0,10);
    npSetLED(16,0,0,10);
    npWrite();
    npClear();
}
```

## 8 - Função principal (main)

A função main é a principal em um programa escrito em **linguagem C** e desempenha o papel de efetivamente ser o local onde o programa é executado. Dentro do main, em suas primeiras linhas de códigos definimos e inicializamos e configuramos as principais **GPIOs** do microcontrolador e seus recursos. Cito dispositivos como **display**, **LEDs**, o **PWM do buzzer**, leitor do **conversor analógico-digital**, bem .

```
setup_display();           // Configura o display
setup_leds();              // Configura os LEDs
pwm_init_buzzer(BUZZER_A ); // Inicializa o buzzer A no PWM
adc_init();                // Inicializa o conversor analógico-digital
adc_gpio_init(JOYSTICK_X); // Define o eixo x do joystick na conversão analógico-digital
```

### 8.1 - Variáveis

Algumas variáveis foram criadas para auxiliar no processo de execução na função main. São elas:

**bool cor = true :** Variável booleana verdadeira que tem como função definir se os LEDs RGB e o OLED ficarão acesos ou apagados, de acordo o instante da execução.

**uint16\_t** adc\_value\_y = adc\_read( ) : Variável do tipo inteiro de 16 bits para guardar o valor lido no ADC.

**char** str\_y[5]: Variável do tipo array de caracteres, é usada para armazenar

**float** adc\_min = 2035.0: Variável do tipo float que define o valor mínimo lido pelo ADC.

**float** adc\_max = 4100.0: Variável do tipo float que define o valor máximo lido pelo ADC.

**uint16\_t** level\_msg = ((adc\_max - adc\_value\_y)/adc\_min)\*100: Esta variável do tipo inteiro de 16 bits retorna o valor em percentual do nível lido no ADC, indo de 0 a 100%.

**sprintf(str\_y, "%d", level\_msg):** Responsável por converter a variável **level\_msg (inteira)** em uma variável do tipo string (**str\_y**)

## 8.2 - Loop infinito e condições

O loop infinito constitui uma parte fundamental para qualquer projeto em microcontroladores. É por meio do loop infinito que o sistema se mantém ativo continuamente, permitindo que o código execute de maneira indefinida, realizando a leitura de sensores, processando sinais e tomando decisões. O loop é criado com o **while (true)**: a função while executa um comando enquanto uma determinada condição for obedecida - como a condição é sempre verdadeira, logo, o programa dentro dele “rodará” de maneira ininterrupta. Neste projeto a árvore de escolha, construída com **if**, **else if** e **else** possui 5 saídas com base em condições pré-definidas. Para aproximadamente cada acréscimo de 20% no valor inicial do ADC cai em uma das condições abaixo.

```
// Executa uma função com base no valor do adc
if(adc_value_y > 2000 && adc_value_y <=2450){
    // Imprime a mensagem "TANQUE CHEIO" no display e o nível em % do nível do reservatório
    message_display("TANQUE CHEIO",str_y, cor, cor, cor, cor, cor);
    level_cinco();           // Nível cinco na matriz de LEDs
    leds_turn_on(!cor, !cor, cor); // Luz verde ligada
}

else if(adc_value_y >2450 && adc_value_y <=2863){ ...

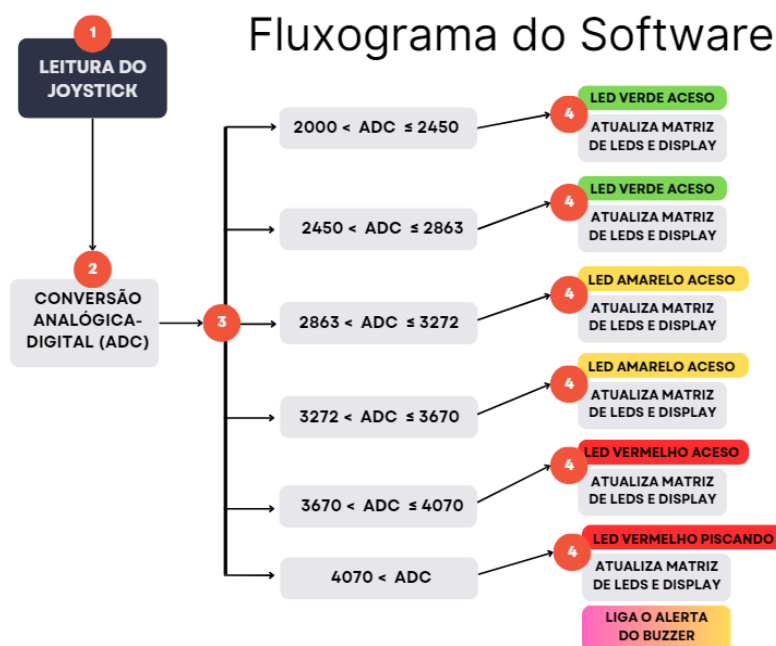
else if(adc_value_y > 2863 && adc_value_y<=3272){ ...

else if(adc_value_y >3272 && adc_value_y<=3670){ ...

else if(adc_value_y >3670 && adc_value_y < 4070){ ...

else{ ...
```

Além disso é possível acompanhar o diagrama simplificado do funcionamento do software, onde basicamente se inicia com a leitura do joystick, cujo valor analógico é convertido para um valor digital de 12 bits. Em seguida, através de condições pré-estabelecidas (como anteriormente mencionado) uma ação é realizada.



## 9 - Configuração do CMakeList

O *CMakeList.txt* consiste em um arquivo de configuração utilizado pelo CMake para compilar e organizar um determinado projeto na Raspberry Pi Pico W. Ele define exatamente quais arquivos serão compilados, vinculados e configurados. Para este projeto as seguintes modificações foram feitas no arquivo CMakeList:

- Adição das bibliotecas: Por meio da `target_link_libraries` é possível vincular bibliotecas importantes para o projeto. Foram elas: **pico\_stdlib**, **hardware\_i2c**, **hardware\_adc**, **hardware\_pio**, **hardware\_pwm** e **hardware\_uart**.
- Adição dos executáveis: Por meio da `add_executable` é possível definir os executáveis que serão criados durante a compilação do projeto. Para este projeto foram adicionados os seguintes executáveis: **U7\_Projeto\_Final.c**, **display/ssd1306.c**, **matriz/neopixel.c**, **buzzer/buzzer.c**, que correspondem, respectivamente, ao programa principal, a biblioteca para configurar o display ssd1306, a biblioteca da matriz de LEDs e do buzzer.
- Arquivo .pio : Por meio da `pico_generate_pio_header` é possível gerar um arquivo **.h** a partir de um arquivo **.pio**. Neste caso, o arquivo em questão é o arquivo **ws2818b.pio** (para a configuração da matriz de LEDs).

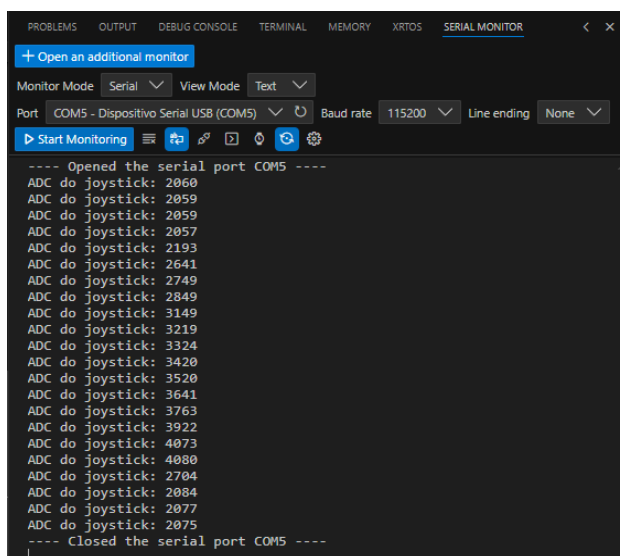
## 10 - Testes de validação

Cada função implementada foi individualmente validada conforme era adicionada ao projeto, através do editor de texto **Visual Studio Code** (VsCode). Como o projeto limita-se aos recursos disponíveis na **BitDogLab**, logo não é possível adicionar um sensor de nível,



sensor ultrassônico ou similares. Para contornar a situação, o joystick fez este papel, simulando a variação do nível ADC recebido, como normalmente seria feito. Movimentar o cursor do joystick para frente aumenta o nível ADC até seu máximo (aproximadamente 4096), sendo que inicialmente (cursor na posição central) o valor é de aproximadamente 2048. Na implementação do software o mínimo foi ajustado para 2030 e o máximo foi ajustado para 4100, pelo para obter um leve margem, a fim de evitar leituras anômalas.

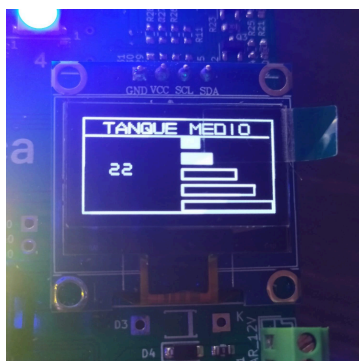
Para constatar a variação do conversor analógico-digital foi implementada uma comunicação com o serial monitor por meio da comunicação serial UART. Deste modo, é possível observar em tempo real se o ajuste no joystick corresponde aos níveis ADC esperados, como é possível notar no registro abaixo.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL MEMORY XRTOS SERIAL MONITOR
+ Open an additional monitor
Monitor Mode Serial View Mode Text
Port COM5 - Dispositivo Serial USB (COM5) Baud rate 115200 Line ending None
Start Monitoring
---- Opened the serial port COM5 ----
ADC do joystick: 2060
ADC do joystick: 2059
ADC do joystick: 2059
ADC do joystick: 2057
ADC do joystick: 2193
ADC do joystick: 2641
ADC do joystick: 2740
ADC do joystick: 2840
ADC do joystick: 3140
ADC do joystick: 3219
ADC do joystick: 3324
ADC do joystick: 3420
ADC do joystick: 3520
ADC do joystick: 3641
ADC do joystick: 3763
ADC do joystick: 3922
ADC do joystick: 4073
ADC do joystick: 4080
ADC do joystick: 2704
ADC do joystick: 2084
ADC do joystick: 2077
ADC do joystick: 2075
---- Closed the serial port COM5 ----
```

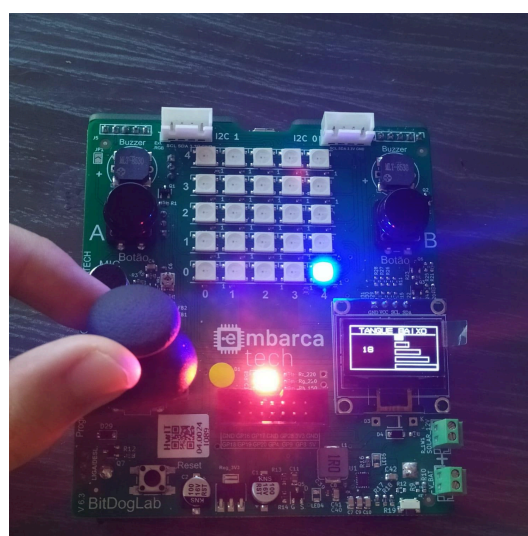
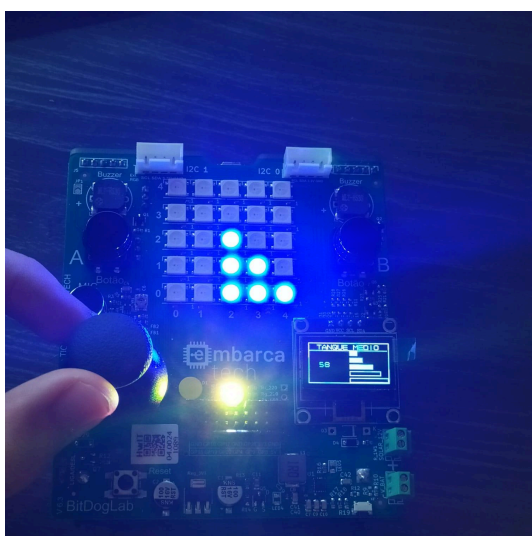
## 11 - Discussão dos resultados

O projeto foi concluído com êxito e todas as funcionalidades funcionaram conforme o esperado. Em especial destaque a configuração do layout no display OLED com uma interface agradável e intuitiva para o usuário, assemelhando-se aos *smartwatch*, fornecendo informações importante como o percentual de água no reservatório, uma mensagem da condição atual e cinco barras para representar visualmente a mesma informação.



Apesar da simplicidade do projeto, ressalto as aplicações para localidades distantes e com limitação de recursos que muitas vezes configura uma peça importante na manutenção e continuidades das atividades cotidianas destas populações. Em um projeto real poderíamos substituir a ativação dos LEDs ou do *buzzer* pelo acionamento de uma **bomba d'água** para, assim, encher novamente o reservatório. Além disso, a comunicação remota do dispositivo via protocolo **wi-fi** ou **LoRa** seria uma ferramenta adequada no contexto da internet das coisas e capaz de trazer comodidade ao usuário, uma vez que seria possível verificar os parâmetros do reservatório por meio de um aplicativo.

Abaixo temos duas fotos demonstrando o funcionamento do projeto



O link da demonstração e explicação detalhada do projeto está disponível no **YouTube** <https://youtu.be/bWMproWeIz8> e o projeto completo encontra-se hospedado no repositório do **GitHub** [https://github.com/Dev1dH/Projeto\\_Final](https://github.com/Dev1dH/Projeto_Final).

## Referências

- [1] <https://www.instructables.com/Raspberry-Pi-Pico-Water-level-indicator-Using-Pote/>
- [2] <https://www.hackster.io/Shafin-Kothia/water-level-monitor-with-raspberry-pi-d509a2>