Kit Eletrónica [RISC-V]

Este é um guia para a placa de desenvolvimento ESP32-C6-DevKitM-1! Neste capítulo introdutório pelo mundo da eletrónica, exploraram-se as poderosas capacidades deste pequeno, mas versátil dispositivo. Quer sejas aprendiz no mundo dos microcontroladores ou já tenhas alguma experiência, este guia fornecer-te-á instruções passo a passo e exemplos práticos para te ajudar a tirar o máximo proveito do teu kit de eletrónica. Começaremos com conceitos básicos e progrediremos para projetos mais complexos. Vamos avançar e começar a criar!

Instalação

Esta página constiui um guia para a instalação do Arduino Integrated Development Environment (IDE), uma ferramenta de *software* utilizada no Departamento de Engenharia Eletrotécnica e de Computadores da Universidade de Coimbra. O **Arduino IDE** é essencial para estudantes e investigadores envolvidos em vários projetos de engenharia, particularmente aqueles relacionados com disciplinas de Engenharia Eletrotécnica e de Computadores. O guia abrange o processo passo a passo de *download*, instalação e configuração do *software* em diferentes plataformas, garantindo que os utilizadores tenham os recursos e as instruções necessários para configurar corretamente o **Arduino IDE**. Este recurso foi concebido para ajudar os utilizadores a superar possíveis desafios durante o processo de instalação, assegurando que possam utilizar o **Arduino IDE** de forma eficiente para os seus propósitos académicos e de investigação.

Contextualização:

O **ESP32-C6-DevKitM-1** é uma placa de desenvolvimento que integra o microcontrolador **ESP32-C6**, parte da linha avançada de soluções **IoT** da *Espressif*. Este microcontrolador integra um CPU RISC-V de 32 bits, oferecendo conectividade sem fios com *Wi-Fi* 6 e *Bluetooth* 5 (LE). Projetado para criadores, o **ESP32-C6-DevKitM-1** proporciona uma plataforma compacta e versátil para a criação e teste de aplicações **IoT**, com amplo suporte para operações seguras, de alto desempenho e baixo consumo de energia. A placa é especialmente indicada para prototipagem e implementação de dispositivos inteligentes.

esp32c6 000	esp32c6 00
Principais componentes do microcontrolador **ESP32-C6-DevKitM-1** e elementos principais utilizados para a configuração e simulação de projetos de engenharia.	

Procedimento:

		esp32c6 01		
Primeira etapa do processo de instalação do **Arduino IDE**: Acesso ao portal oficial para *download* do *software*.				

Apresentam-se de seguida as etapas fundamentais para a configuração do Arduino IDE

- instalar o pacote de placas ESP32, abrindo o menu Preferências navegando até File > Preferences. Procurar na parte inferior do menu Preferências "Additional boards manager URLS" e copiar o link JSON seguinte nesse campo:

```
https://espressif.github.io/arduino-esp32/package_esp32_dev_index.json
```

esp32c6 002	esp32c6 003

- Abrir a ferramenta Boards Manager, pesquisar por

```
espressif ESP32
```

e instalar a versão mais recente. Este processo de instalação pode demorar algum tempo.

esp32c6 004	esp32c6 005

- Selecionar o dispositivo ESP32 adequado

```
ESP32C6 Dev Module
```

esp32c6 006	esp32c6 007	esp32c6 008

Primeiro Programa:

Clicar no link para continuar para primeiro programa

Referências

[1] Espressif Systems, "ESP32-C6-DevKitM-1 User Guide," Espressif Systems, Aug. 2023. [Online]. Available: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32c6/esp32-c6-devkitm-1/user_guide.html. [Accessed: 27-Aug-2024].

[2] Arduino, "Download and install Arduino IDE," Arduino, Sep. 2021. [Online]. Available: https://support.arduino.cc/hc/en-us/articles/360019833020-Download-and-install-Arduino-IDE. [Accessed: 27-Aug-2024].

Primeiro Programa

Boas-vindas ao guia introdutório para o seu primeiro programa Arduino! A programação com IDE Arduino é uma excelente maneira de mergulhar no mundo dos sistemas embebidos, permitindo controlar e interagir com o mundo físico através de código. Envolve a escrita de código no Ambiente de Desenvolvimento Integrado (IDE) do Arduino, onde um programa é chamado de *sketch*. Neste guia, vai aprender sobre as estruturas fundamentais de um *sketch* Arduino: as funções **setup()** e **loop()**.

A função **setup()** é onde se inicializam as definições e configurações, sendo **executada apenas uma vez** quando a placa Arduino é ligada ou reiniciada. É aqui que se definem os modos dos pinos, se inicia a comunicação serial e se realizam outras tarefas de configuração necessárias para um dado projeto.

Por outro lado, a função **loop()** contém a lógica principal do programa e é **executada continuamente**, permitindo que a plca de desenvolvimento a programar realize tarefas repetitivas, como ler sensores, controlar saídas e responder a eventos.

Ao dominar estes conceitos, estará apto a criar projetos dinâmicos e interativos que respondem ao mundo ao seu redor. Vamos começar!

O código seguinte faz com que o *Light-emitting diode* (LED) da placa de desenvolvimento *ESP32-C6-DevKitM-1* pisque, alternando entre ligado e desligado em intervalos regulares. Além disso, apresenta-se de seguida uma representação ilustrativo para referência.

```
/*
  BlinkRGB

  Demonstrates usage of onboard RGB LED on some ESP dev boards.

  Calling digitalWrite(RGB_BUILTIN, HIGH) will use hidden RGB driver.

  RGBLedWrite demonstrates control of each channel:
  void neopixelWrite(uint8_t pin, uint8_t red_val, uint8_t green_val, uint8_t blue_val)

  WARNING: After using digitalWrite to drive RGB LED it will be impossible to drive the same
pin
    with normal HIGH/LOW level
*/
//#define RGB_BRIGHTNESS 64 // Change white brightness (max 255)
```

```
// the setup function runs once when you press reset or power the board
void setup() {
  // No need to initialize the RGB LED
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(RGB_BUILTIN, HIGH);  // Turn the RGB LED white
  delay(1000);
  digitalWrite(RGB_BUILTIN, LOW);  // Turn the RGB LED off
  delay(1000);
}
```

tft.gif
Montagem exemplificativa

Utilização do Monitor Serial no Arduino IDE

O Monitor Serial é uma ferramenta crucial dentro do Arduino IDE que permite enviar e receber dados entre a placa de desenvolvimento e o computador. Esta funcionalidade é extremamente útil para depuração, monitorização de leituras de sensores e interação com o programa do Arduino em tempo real.

Como Funciona o Monitor Serial

O Monitor Serial comunica com a placa de desenvolvimento através de uma conexão serial, que normalmente é estabelecida através do cabo USB que liga a placa ao computador. Esta conexão permite a transmissão de dados em texto entre o seu computador e a placa de desenvolvimento. Podem enviar-se comandos do Monitor Serial para a placa de desenvolvimento, e, por sua vez, a placa de desenvolvimento pode enviar de volta dados, que são exibidos na janela do Monitor Serial.

Para utilizar o Monitor Serial, siga estes passos:

1. **Inicializar a Comunicação Serial:** No *sketch*, deve inicializar-se a comunicação serial utilizando a função `*Serial.begin()*`. Esta função recebe um único argumento, a taxa de transmissão, que define a velocidade da comunicação em bits por segundo (bps). Uma taxa de transmissão comum é 9600.

```
void setup() {
Serial.begin(9600); // Inicia a comunicação serial a uma taxa de 9600 bps
}
```

2. **Enviar Dados para o Monitor Serial:** Pode enviar-se dados da placa de desenvolvimento para o Monitor Serial utilizando as funções `*Serial.print()*` ou `*Serial.println()*`. A função `*Serial.print()*` envia dados sem adicionar uma nova linha no final, enquanto que `*Serial.println()*` envia os dados seguidos de um caractere de nova linha.

```
void loop() {
  Serial.println("Olá, Mundo!"); // Envia "Olá, Mundo!" para o Monitor serial
  delay(1000); // Aguarda 1 segundo
}
```

3. **Receber Dados do Monitor Serial:** O Monitor Serial também pode enviar dados para a placa de desenvoldimento. Podem ler-se estes dados utilizando no *sketch* as funções `*Serial.read()*`, `*Serial.available()*`, e outras funções relacionadas. No exemplo seguinte ilustra-se a leitura de um caractereusando o Monitor Serial:

```
void loop() {
  if (Serial.available() > 0) { // Verifica se há dados disponíveis para leitura
    char receivedChar = Serial.read(); // Lê o próximo caractere disponível
    Serial.print("Recebido: ");
    Serial.println(receivedChar); // Imprime o caractere recebido
  }
}
```

Segundo Programa

O *sketch* seguinte ilustra como monificar o *sketch* do primeiro programa anteriormente sugerido para começar a aproveitar as possibilidades do Monitor Serial:

```
/*
  BlinkRGB

  Demonstrates usage of onboard RGB LED on some ESP dev boards.

  Calling digitalWrite(RGB_BUILTIN, HIGH) will use hidden RGB driver.

  RGBLedWrite demonstrates control of each channel:
  void neopixelWrite(uint8_t pin, uint8_t red_val, uint8_t green_val, uint8_t blue_val)

  WARNING: After using digitalWrite to drive RGB LED it will be impossible to drive the same
pin
    with normal HIGH/LOW level
*/
```

```
//#define RGB_BRIGHTNESS 64 // Change white brightness (max 255)

// the setup function runs once when you press reset or power the board
void setup() {
  // No need to initialize the RGB LED
  Serial.begin(9600);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(RGB_BUILTIN, HIGH);  // Turn the RGB LED white
  Serial.println("LED ligado"); // Imprime o caractere recebido
  delay(1000);
  digitalWrite(RGB_BUILTIN, LOW);  // Turn the RGB LED off
  Serial.println("LED desligado"); // Imprime o caractere recebido
  delay(1000);
}
```

Depois garantir a selecionação do dispositivo ESP32 adequado (*ESP32C6 Dev Module*), copiar o código anterior, compilar e carregar o código para a placa ESP32, deverá ser possível verificar o resultado.

tft.gif	tft.gif

[Programa] LED com efeito dinâmico de mudança de cor

O seguinte código demonstra como controlar o LED RGB numa placa de desenvolvimento ESP32-C6. Especificamente, o código de exemplo mostra como criar um efeito dinâmico de mudança de cor, fazendo um ciclo através de diferentes cores no LED RGB da placa. Este exemplo serve como uma ilustração prática de como aproveitar as capacidades do ESP32-C6 para produzir padrões de luz visualmente atraentes. Ao executar este código, os utilizadores podem observar como o LED RGB transita suavemente por um espectro de cores, tornando-o uma referência útil para desenvolver efeitos semelhantes baseados em luz em outros projetos.

esp32c6 01	esp32c6 01

```
Adafruit NeoPixel
```

Usando a ferramenta de gestão de bibliotecas do IDE Arduino procurar e instalar a biblioteca: Adafruit_NeoPixel.

```
#include <Adafruit_NeoPixel.h>
#include <cmath>

// Pin and LED configuration
constexpr uint8_t LED_PIN = 8;
constexpr uint8_t NUM_LEDS = 1;

// NeoPixel object
Adafruit_NeoPixel rgbLed(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

// Color structure
struct RGB {
    uint8_t r, g, b;
};

void setup() {
    rgbLed.begin();  // Initialize the RGB LED
    rgbLed.show();   // Turn off the LED (as it's initialized to all 0s)
}
```

```
void setColor(const RGB& color) {
    rgbLed.setPixelColor(0, rgbLed.Color(color.r, color.g, color.b));
    rgbLed.show();
}

// Convert HSV to RGB
RGB hsvToRgb(float h, float s, float v) {
    float c = v * s;
    float x = c * (1 - std::abs(std::fmod(h / 60.0, 2) - 1));
    float m = v - c;
    float r, g, b;

    if (h >= 0 && h < 60) {
        r = c, g = x, b = 0;
    } else if (h >= 60 && h < 120) {
        r = x, g = c, b = 0;
    } else if (h >= 120 && h < 180) {
        r = 0, g = c, b = x;
    } else if (h >= 180 && h < 240) {
        r = 0, g = x, b = c;
    } else if (h >= 240 && h < 300) {
        r = x, g = 0, b = c;
    } else {
        r = c, g = 0, b = x;
    }

    return {
        static_cast<uint8_t>((r + m) * 255),
        static_cast<uint8_t>((g + m) * 255),
        static_cast<uint8_t>((b + m) * 255)
    };
}

void loop() {
    constexpr unsigned long CYCLE_DURATION = 10000;  // 10 seconds for a full color cycle
    constexpr unsigned long STEPS = 1000;            // Number of steps in the cycle

    unsigned long startTime = millis();
    unsigned long currentTime;
```

```
    while (true) {
        currentTime = millis();
        float progress = static_cast<float>((currentTime - startTime) % CYCLE_DURATION) /
CYCLE_DURATION;

        float hue = progress * 360.0f;  // Hue ranges from 0 to 360
        RGB color = hsvToRgb(hue, 1.0f, 1.0f);  // Full saturation and value

        setColor(color);

        delay(CYCLE_DURATION / STEPS);  // Small delay for smooth transition
    }
}
```

tft.gif	tft.gif
Compilar e carregar o código para a placa ESP32-C6.	

tft.gif
Montagem exemplificativa

Referências

[1] H4Fide, "ESP32-C6-RGB-LED-Control," GitHub, repository, 2024. [Online]. Available: https://github.com/h4fide/ESP32-C6-RGB-LED-Control. [Accessed: 27-Aug-2024].

[Programa] LED a piscar

Objetivo:

Utilizar o *ESP32-C6-DevKitM-1* para alimentar um LED e obter um piscar controlado de forma programática.

Lista de material:

- *ESP32-C6-DevKitM-1*
- Breadboard
- Resistência de 330 ohm (Ω)
- Díodo emissor de luz (LED) vermelho
- Cabo USB C
- Fios de ligação

aaa.png

Contextualização:

Em eletrónica, um circuito LED é um circuito usado para alimentar um díodo emissor de luz (LED). O circuito deve fornecer corrente suficiente para acender o LED e evitar danos ao mesmo. O circuito mais simples para acionar um LED é através de uma resistência em série [3].

ESP32-C6-DevKitM-1

Apesar de se aconselhar a consulta da página do fabricante sobre o kit *ESP32-C6-DevKitM-1* para mais detalhe, disponibilizam-se de seguida duas figuras com os componentes principais do kit bem como o esquema de terminais para referência.

esp32-devkitc-functional-overview.jpg

esp32-devkitC-v4-pinout.png

Breadboard

Uma breadboard ou placa de ensaio ou matriz de contactos é uma placa com furos e conexões condutoras tipicamente utilizada para a montagem de protótipos e projetos em estado inicial [1]. Deve ser tido em consideração que internamente a breadboard também é composta por contactos metálicos, tipicamente organizados como se ilustra na figura em baixo. As linhas a azul e verde simbolizam ligações metálicas internas, ou seja, e por exemplo, tal como esquematizado, deve considerar-se que o orifício a1 está ligado internamente aos orifícios b1, c1, d1 e e1.

breadboard.png

Resistência

Uma resistência é um componente elétrico passivo de dois terminais que implementa resistência elétrica como um elemento de circuito [4]. Ainda que não seja do âmbito do presente documento detalhar este conceito, disponibiliza-se de seguida o esquema de quatro bandas e tabela de referência [2], tipicamente usados para indicar os valores associados, neste caso, as resistências.

resistor_b.png

Díodo emissor de luz (LED)

Um díodo emissor de luz (LED) é um dispositivo semicondutor que emite luz quando uma corrente flui através deste [3]. Destacar da figura seguinte a distinção entre ânodo e cátodo, tipicamente materializada por um comprimento diferente dos terminais de ligação (terminal do ânodo mais longo que o do cátodo), e um chanfro (ou superfície plana) no lado do cátodo.

led_cat_an.png

Procedimento:

As saídas digitais do *ESP32-C6-DevKitM-1* podem ser programas para assumir determinados valores de tensão como 0V (GND) ou 3.3V (VCC). Fazendo uso desta possibilidade, e ligando o LED ao ESP32, deverá então ser possível obter um piscar do LED controlado de forma programática.

Escolhendo os terminais 5 e GND do ESP32 como referência deve considerar-se a seguinte montagem:

pisca1.png	pisca2aa.png

Instruções:

- Montar o circuito representado anteriormente
- Ligar a placa ESP32 ao computador por via de cabo USB C
- Abrir o IDE Arduino
- Selecionar o dispositivo ESP32 adequado (*ESP32C6 Dev Module*)
- Copiar o seguinte código

```
/*
Pisca-pisca

Liga o LED por um segundo, desliga por um segundo e assim sucessivamente.
*/



// a função setup é executada pontualmente quando o botão reset é premido ou a placa é
alimentada
void setup() {
  // inicializa o pino digital 5 como saída.
```

```
  pinMode(5, OUTPUT);
}

// a função loop é executada sucessivamente, uma e outra vez, e por aí em diante
void loop() {
  digitalWrite(5, HIGH);   // liga o LED
  delay(1000);              // espera por um segundo
  digitalWrite(5, LOW);    // desliga o LED
  delay(1000);              // espera por um segundo
}
```

- Compilar e carregar o código para a placa ESP32
- Verificar o resultado

blink

Referências

[1] Wikipedia. Breadboard. url: https://en.wikipedia.org/wiki/Breadboard (acedido em 18/08/2023).

[2] Wikipedia. Electronic color code. url: https://en.wikipedia.org/wiki/Electronic_color_code (acedido em 18/08/2023).

[3] Wikipedia. LED circuit. url: https://en.wikipedia.org/wiki/LED_circuit (acedido em 18/08/2023).

[4] Wikipedia. Resistor. url: https://en.wikipedia.org/wiki/Resistor (acedido em 18/08/2023).

[Programa] Interruptor

Objetivo:

Utilizar o *ESP32-C6-DevKitM-1* em conjugação com um interruptor e monitorizar estados.

Lista de material:

- *ESP32-C6-DevKitM-1*
- Breadboard
- Cabo USB C
- Interruptor (tipo botão)
- Fios de ligação

material_list.png

Contextualização:

Um interruptor é um componente elétrico que pode desconectar ou conectar o caminho condutor de um circuito elétrico, interrompendo a corrente elétrica ou conduzindo-a de um terminal para outro [1].

Interruptor

Um interruptor tipo botão tem tipicamente quatro terminais que estão conectados internamente aos pares. Por isso, numa utilização típica, apenas dois terminais, que não ligados internamente, são usados, como se ilustra na imagem seguinte:

switch_a.png	switch_b.png	switch_c.png	switch_d.png

Ao ser pressionado o interruptor alterna entre os estados conetado e não conetado:

switch_e.png	switch_e.png
(pressionado)	(não pressionado)
switch_f.png	switch_g.png
(fechado)	(aberto)

Procedimento:

Escolhendo o terminal 23 e o GND do ESP32 como referência deve considerar-se o seguinte circuito:

interruptor_esq.png

Instruções:

- Montar o circuito esquematizado anteriormente
- Ligar a placa ESP32 ao computador por via de cabo USB micro
- Abrir o IDE Arduino
- Selecionar o dispositivo ESP32 adequado （ *ESP32C6 Dev Module* ）
- Copiar o seguinte código

```
/*
interruptor□tipo botão
*/□

#define BUTTON_PIN 23  // definição de terminal GPIO23 ligado ao interruptor

// Variáveis
int lastState = LOW;  // o estado anterior do terminal de entrada
int currentState;     // o estado atual do terminal de entrada

void setup() {
  // inicializa a comunicação serial a 115200 bits por segundo:
  Serial.begin(115200);
  // inicializa o terminal do interruptor tipo botão:
  // (o terminal de entrada toma o valor de HIGH quando o interruptor estiver aberto,
  // e LOW quando estiver fechado)
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  // leitura do estado do interruptor tipo botão:
  currentState = digitalRead(BUTTON_PIN);

  if (lastState == HIGH && currentState == LOW) {
    Serial.println("o interruptor foi pressionado");
    digitalWrite(RGB_BUILTIN, HIGH);
  }
  else if (lastState == LOW && currentState == HIGH) {
    Serial.println("o interruptor foi solto");
    digitalWrite(RGB_BUILTIN, LOW);
  }
```

```
  // guarda o último estado
  lastState = currentState;
}
```

- Compilar e carregar o código para a placa ESP32
- Verificar o resultado no monitor de comunicação serial do IDE Arduino

switch_f.png

Referências

[1] Wikipedia. Switch. url: https://en.wikipedia.org/wiki/Switch (acedido em 22/08/2023).

[Programa] Potenciómetro

Objectivo:

Utilizar o *ESP32-C6-DevKitM-1* para ler valor de tensão de saída de um potenciómetro.

Lista de material:

- *ESP32-C6-DevKitM-1*
- Breadboard
- Potenciómetro de 10kΩ
- Fios de ligação
- Cabo USB C

Fuw7bnV.jpeg

Contextualização:

Um potenciómetro é um componente eletrónico que incorpora uma resistência elétrica ajustável [1].

Potenciómetro

Um potenciómetro possui tipicamente três terminais incluindo um contacto ajustável, deslizante ou rotativo, que forma um divisor de tensão ajustável [1].

eg_out_2.png	eg_0_2.png	eg_120_2.png	eg_240_2.png

ESP32-C6-DevKitM-1

Apesar de se aconselhar a consulta da página do fabricante sobre o kit *ESP32-C6-DevKitM-1* é possível considerar que em determinadas situações os terminais de entrada do microcontrolador ESP32 podem converter tensão (algo como uma quantidade analógica entre 0V e 3.3V) em um valor digital inteiro entre 0 e 4095 (num total de 2^{12}=4096 representações possíveis).

esp32.png

Procedimento:

Atentando ao objetivo enunciado, e escolhendo os terminais 2 (para o terminal de saída do potenciómetro), 3.3V e o GND do ESP32 como referência deve considerar-se a seguinte montagem:

interruptor_montagem_2.png

Instruções:

- Montar o circuito esquematizado anteriormente
- Ligar a placa ESP32 ao computador por via de cabo USB C
- Abrir o IDE Arduino
- Selecionar o dispositivo ESP32 adequado (*ESP32C6 Dev Module*)
- Copiar o seguinte código

```
#include <Adafruit_NeoPixel.h>
#include <cmath>



// Pin and LED configuration
constexpr uint8_t LED_PIN = 8;
constexpr uint8_t NUM_LEDS = 1;

// NeoPixel object
Adafruit_NeoPixel rgbLed(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

// Color structure
struct RGB {
    uint8_t r, g, b;
};

void setup() {
    rgbLed.begin();  // Initialize the RGB LED
    rgbLed.show();   // Turn off the LED (as it's initialized to all 0s)
    Serial.begin(115200);
}

void setColor(const RGB& color) {
    rgbLed.setPixelColor(0, rgbLed.Color(color.r, color.g, color.b));
    rgbLed.show();
}

// Convert HSV to RGB
RGB hsvToRgb(float h, float s, float v) {
    float c = v * s;
    float x = c * (1 - std::abs(std::fmod(h / 60.0, 2) - 1));
    float m = v - c;
    float r, g, b;
```

```
    if (h >= 0 && h < 60) {
        r = c, g = x, b = 0;
    } else if (h >= 60 && h < 120) {
        r = x, g = c, b = 0;
    } else if (h >= 120 && h < 180) {
        r = 0, g = c, b = x;
    } else if (h >= 180 && h < 240) {
        r = 0, g = x, b = c;
    } else if (h >= 240 && h < 300) {
        r = x, g = 0, b = c;
    } else {
        r = c, g = 0, b = x;
    }

    return {
        static_cast<uint8_t>((r + m) * 255),
        static_cast<uint8_t>((g + m) * 255),
        static_cast<uint8_t>((b + m) * 255)
    };
}

float floatMap(float x, float in_min, float in_max, float out_min, float out_max) {
return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

void loop() {
    constexpr unsigned long CYCLE_DURATION = 10000;  // 10 seconds for a full color cycle
    constexpr unsigned long STEPS = 1000;            // Number of steps in the cycle

    unsigned long startTime = millis();
    unsigned long currentTime;

    while (true) {
        // leitura do valor de entrada no terminal analógico GPIO33:
      int analogValue = analogRead(2);
        Serial.println(analogValue);
      // Normalização dos valores lidos para a tensão do potenciómetro
      float voltage = floatMap(analogValue, 0, 4095, 0, 1.0);
        Serial.println(voltage);
```

```
        currentTime = millis();
        //float progress = static_cast<float>((currentTime - startTime) % CYCLE_DURATION) /
CYCLE_DURATION;

        //float hue = progress * 360.0f;  // Hue ranges from 0 to 360
        float hue = voltage * 360.0f;  // Hue ranges from 0 to 360
        RGB color = hsvToRgb(hue, 1.0f, 1.0f);  // Full saturation and value
        //RGB color = hsvToRgb(255.0f, voltage, 1.0f);

        setColor(color);

        delay(CYCLE_DURATION / STEPS);  // Small delay for smooth transition
    }
}
```

- Compilar e carregar o código para a placa ESP32
- Verificar o resultado

out_hand_2.png

Referências:

[1] Wikipedia. Potentiometer. url: https://en.wikipedia.org/wiki/Potentiometer (acedido em 23/08/2023).

[Programa] SHT40 (exemplo I²C)

Objetivo:

Utilizar o *ESP32-C6-DevKitM-1* para ler valores de temperatura de um sensor SHT40.

Lista de material:

- *ESP32-C6-DevKitM-1*
- Breadboard
- Sensor SHT40
- Fios de ligação
- Cabo USB C

8_material_list.png

Contextualização:

Grove - Temperature & Humidity Sensor (SHT40)

O sensor de temperatura e humidade da Seeed SHT40 é um sensor digital com interface I²C padrão [2].

Inter-Integrated Circuit (I²C) [3]

O *Inter-Integrated Circuit* (I²C)[3] é um barramento serial que permite a co-existência de múltiplos controladores e periféricos (historicamente designados por *master*/*slave*) e considera essencialmente duas ligações:

SDA	Serial Data Line
SDL	Serial Clock Line

Ambas são bidirecionais e recorrem a resistências *pull-up*. As tensões típicas utilizadas são de +5 V ou +3,3 V, embora sejam permitidos sistemas com outras tensões.

O design de referência I²C possui um espaço de endereço de 7 *bits*, com uma extensão de 10 *bits* raramente utilizada. As velocidades de barramento I²C comuns são o modo padrão de 100 *kbit/s* existindo outras possibilidades para casos específicos.

A comunicação I²C segue uma estrutura organizada em pacotes de dados. Cada transação no barramento I²C inicia com uma condição de S*tart*, seguida pelo envio de um *byte* de endereço, que

identifica o dispositivo preriférico com o qual o controlador deseja comunicar. Após o endereço, o controlador pode enviar ou receber dados, dependendo da natureza da transação (escrita ou leitura).

Os principais elementos dos protocolos de mensagem do I²C incluem:

1. **Condição de Início (*Start*):** O controlador gera uma transição do sinal SDA de alto para baixo enquanto SCL está alto, indicando o início de uma comunicação.
2. ***Byte* de Endereço:** Após a condição de S*tart*, o controlador envia um *byte* de endereço que contém o endereço do periférico alvo e um *bit* de leitura/escrita. O periférico com o endereço correspondente responde com um *bit* de reconhecimento (*ACK*).
3. **Dados:** Os dados são transmitidos em *bytes*, seguidos por um *bit* de reconhecimento (*ACK*) enviado pelo receptor após cada *byte*. Se o receptor não reconhecer o *byte*, ele envia um *bit* de não reconhecimento (*NACK*).
4. **Condição de Paragem (*Stop*):** A comunicação termina com uma condição de *Stop*, onde o controlador gera uma transição do sinal SDA de baixo para alto enquanto SCL está alto. Isso sinaliza o fim da transação no barramento I²C.

Essa estrutura simples e eficiente permite que o I²C suporte a comunicação entre múltiplos dispositivos de forma organizada, minimizando a necessidade de fios e facilitando a integração em sistemas eletrónicos complexos.

Desenvolvido pela Philips Semiconductor em 1982, o I²C é ideal para aplicações que exigem comunicação eficiente e de curto alcance entre componentes eletrónicos. Este protocolo suporta a comunicação entre múltiplos dispositivos, onde o dispositivo contolador comanda o barramento I²C e os peféricos respondem aos comandos recebidos. A simplicidade do I²C, combinada com a sua flexibilidade, torna-o uma escolha popular para interligar sensores, displays, memória e outros periféricos a microcontroladores em diversos projetos.

De seguida apresenta-se uma aplicação prática do I²C usando a placa de desenvolvimento *ESP32-C6-DevKitM-1*, que possui suporte integrado para I²C, em conjunto com o sensor de temperatura e humidade SHT40.

Procedimento:

Atentando nos terminais 3, 4, GND e 5V do ESP32 como referência deve considerar-se o seguinte ilustração de montagem:

8_i2c_setup.png

Instruções:

- Montar o circuito esquematizado anteriormente
- Ligar a placa ESP32 ao computador por via de cabo USB C
- Abrir o IDE Arduino
- Usando a ferramenta de gestão de bibliotecas do IDE Arduino procurar e instalar a biblioteca **Adafruit SHT4X** ([1])

- Selecionar o dispositivo ESP32 adequado (*ESP32C6 Dev Module*)
- Copiar o seguinte

```
#include <Wire.h>
#include <Adafruit_SHT4x.h>

// Definição de uma nova instância da classe TwoWire
TwoWire myWire = TwoWire(0x44); // Usa o barramento I2C no ID 0 (podes alterar conforme necessário)
Adafruit_SHT4x sht40 = Adafruit_SHT4x();

void setup() {
  digitalWrite(RGB_BUILTIN, LOW);  // Turn the RGB LED off
  Serial.begin(115200);

  // Inicia a instância myWire no barramento I2C utilizando pinos SDA e SCL personalizados
  myWire.begin(4, 3); // Pinos SDA=4 e SCL=3 (substitua pelos pinos que preferires)

  // Inicializa o sensor SHT40 usando a instância personalizada de TwoWire
  if (!sht40.begin(&myWire)) {
    Serial.println("Falha ao inicializar o SHT40. Verifique a conexão I2C.");
    while (1) {
      delay(10);
    }
  }
  sht40.setPrecision(SHT4X_HIGH_PRECISION);
  sht40.setHeater(SHT4X_NO_HEATER);
  Serial.println("Sensor SHT40 inicializado com sucesso!");
}

void loop() {
  digitalWrite(RGB_BUILTIN, LOW);  // Turn the RGB LED off

  sensors_event_t humidity, temp;
  sht40.getEvent(&humidity, &temp);

  Serial.print("Temperatura: ");
  Serial.print(temp.temperature);
  Serial.println(" °C");
```

```
  Serial.print("Humidade: ");
  Serial.print(humidity.relative_humidity);
  Serial.println(" %");

  delay(2000);  // Espera 2 segundos antes de nova leitura

  digitalWrite(RGB_BUILTIN, HIGH);  // Turn the RGB LED white
}
```

- Compilar e carregar o código para a placa ESP32
- Verificar o resultado

SotAp.gif	8_img_out_b.png

Referências:

[1] Adafruit. Adafruit Sensirion SHT40, SHT41 & SHT45 Temperature & Humidity Sensors. url: https://learn.adafruit.com/adafruit-sht40-temperature-humidity-sensor (acedido em 29/08/2023).

[2] Seeed Studio. Grove - Temperature & Humidity Sensor. url: https://wiki.seeedstudio.com/Grove-SHT4x/ (acedido em 29/08/2023).

[3] Wikipedia. I2C. url: https://en.wikipedia.org/wiki/I%C2%B2C (acedido em 29/08/2023).

[Programa] Lâmpada Zigbee

(provisório) [1]

8_material_list.png

Lista de Material

8_material_list.png

Notas (part I/II)

Este exemplo mostra como configurar o **dispositivo final Zigbee** e utilizá-lo como lâmpada de domótica de ligar/desligar.

Requisitos

Uma placa de desenvolvimento (ESP32-C6) como **dispositivo final Zigbee** (carregada com o exemplo **Lâmpada Zigbee**)

Arduino IDE

- Antes de Compilar/Verificar, seleccionar a placa correcta: `Ferramentas -> Placa`.
- Selecionar o modo Zigbee do dispositivo final: `Ferramentas -> Modo Zigbee: Zigbee ED (dispositivo final)`
- Selecionar Esquema de Partição para Zigbee: `Ferramentas -> Esquema de Partição: Zigbee 4MB com spiffs`
- Seleccione a porta COM: `Ferramentas -> Porta: xxx` em que `xxx` é a porta COM detectada.

Lâmpada Zigbee

```
// Copyright 2023 Espressif Systems (Shanghai) PTE LTD
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at

//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
```

```
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/**
 * @brief This example demonstrates simple Zigbee light bulb.
 *
 * The example demonstrates how to use ESP Zigbee stack to create a end device light bulb.
 * The light bulb is a Zigbee end device, which is controlled by a Zigbee coordinator.
 *
 * Proper Zigbee mode must be selected in Tools->Zigbee mode
 * and also the correct partition scheme must be selected in Tools->Partition Scheme.
 *
 * Please check the README.md for instructions and more detailed description.
 */

#ifndef ZIGBEE_MODE_ED
#error "Zigbee end device mode is not selected in Tools->Zigbee mode"
#endif

#include "esp_zigbee_core.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "ha/esp_zigbee_ha_standard.h"

#define LED_PIN RGB_BUILTIN

/* Default End Device config */
#define ESP_ZB_ZED_CONFIG()                                                         \
  {                                                                                 \
    .esp_zb_role = ESP_ZB_DEVICE_TYPE_ED, .install_code_policy = INSTALLCODE_POLICY_ENABLE, \
    .nwk_cfg = {                                                                    \
      .zed_cfg =                                                                    \
        {                                                                           \
          .ed_timeout = ED_AGING_TIMEOUT,                                           \
          .keep_alive = ED_KEEP_ALIVE,                                              \
        },                                                                          \
    },                                                                              \
  }
```

```
#define ESP_ZB_DEFAULT_RADIO_CONFIG() \
  { .radio_mode = ZB_RADIO_MODE_NATIVE, }

#define ESP_ZB_DEFAULT_HOST_CONFIG() \
  { .host_connection_mode = ZB_HOST_CONNECTION_MODE_NONE, }

/* Zigbee configuration */
#define INSTALLCODE_POLICY_ENABLE   false /* enable the install code policy for security */
#define ED_AGING_TIMEOUT            ESP_ZB_ED_AGING_TIMEOUT_64MIN
#define ED_KEEP_ALIVE               3000                                    /* 3000 millisecond
*/
#define HA_ESP_LIGHT_ENDPOINT       10                                      /* esp light bulb
device endpoint, used to process light controlling commands */
#define ESP_ZB_PRIMARY_CHANNEL_MASK ESP_ZB_TRANSCEIVER_ALL_CHANNELS_MASK /* Zigbee primary
channel mask use in the example */

/********************** Zigbee functions **************************/
static void bdb_start_top_level_commissioning_cb(uint8_t mode_mask) {
  ESP_ERROR_CHECK(esp_zb_bdb_start_top_level_commissioning(mode_mask));
}

void esp_zb_app_signal_handler(esp_zb_app_signal_t *signal_struct) {
  uint32_t *p_sg_p = signal_struct->p_app_signal;
  esp_err_t err_status = signal_struct->esp_err_status;
  esp_zb_app_signal_type_t sig_type = (esp_zb_app_signal_type_t)*p_sg_p;
  switch (sig_type) {
    case ESP_ZB_ZDO_SIGNAL_SKIP_STARTUP:
      log_i("Zigbee stack initialized");
      esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_INITIALIZATION);
      break;
    case ESP_ZB_BDB_SIGNAL_DEVICE_FIRST_START:
    case ESP_ZB_BDB_SIGNAL_DEVICE_REBOOT:
      if (err_status == ESP_OK) {
        log_i("Device started up in %s factory-reset mode", esp_zb_bdb_is_factory_new() ? "" :
"non");
        if (esp_zb_bdb_is_factory_new()) {
          log_i("Start network formation");
          esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_NETWORK_STEERING);
        } else {
```

```
          log_i("Device rebooted");
        }
      } else {
        /* commissioning failed */
        log_w("Failed to initialize Zigbee stack (status: %s)", esp_err_to_name(err_status));
      }
      break;
    case ESP_ZB_BDB_SIGNAL_STEERING:
      if (err_status == ESP_OK) {
        esp_zb_ieee_addr_t extended_pan_id;
        esp_zb_get_extended_pan_id(extended_pan_id);
        log_i(
          "Joined network successfully (Extended PAN ID:
%02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x, PAN ID: 0x%04hx, Channel:%d, Short Address:
0x%04hx)",
          extended_pan_id[7], extended_pan_id[6], extended_pan_id[5], extended_pan_id[4],
extended_pan_id[3], extended_pan_id[2], extended_pan_id[1],
          extended_pan_id[0], esp_zb_get_pan_id(), esp_zb_get_current_channel(),
esp_zb_get_short_address()
        );
      } else {
        log_i("Network steering was not successful (status: %s)",
esp_err_to_name(err_status));
        esp_zb_scheduler_alarm((esp_zb_callback_t)bdb_start_top_level_commissioning_cb,
ESP_ZB_BDB_MODE_NETWORK_STEERING, 1000);
      }
      break;
    default: log_i("ZDO signal: %s (0x%x), status: %s", esp_zb_zdo_signal_to_string(sig_type),
sig_type, esp_err_to_name(err_status)); break;
  }
}

static esp_err_t zb_action_handler(esp_zb_core_action_callback_id_t callback_id, const void
*message) {
  esp_err_t ret = ESP_OK;
  switch (callback_id) {
    case ESP_ZB_CORE_SET_ATTR_VALUE_CB_ID: ret =
zb_attribute_handler((esp_zb_zcl_set_attr_value_message_t *)message); break;
    default:                               log_w("Receive Zigbee action(0x%x) callback",
callback_id); break;
```

```
  }
  return ret;
}

static void esp_zb_task(void *pvParameters) {
  esp_zb_cfg_t zb_nwk_cfg = ESP_ZB_ZED_CONFIG();
  esp_zb_init(&zb_nwk_cfg);
  esp_zb_on_off_light_cfg_t light_cfg = ESP_ZB_DEFAULT_ON_OFF_LIGHT_CONFIG();
  esp_zb_ep_list_t *esp_zb_on_off_light_ep =
esp_zb_on_off_light_ep_create(HA_ESP_LIGHT_ENDPOINT, &light_cfg);
  esp_zb_device_register(esp_zb_on_off_light_ep);
  esp_zb_core_action_handler_register(zb_action_handler);
  esp_zb_set_primary_network_channel_set(ESP_ZB_PRIMARY_CHANNEL_MASK);

  //Erase NVRAM before creating connection to new Coordinator
  esp_zb_nvram_erase_at_start(true);  //Comment out this line to erase NVRAM data if you are
conneting to new Coordinator

  ESP_ERROR_CHECK(esp_zb_start(false));
  esp_zb_main_loop_iteration();
}

/* Handle the light attribute */

static esp_err_t zb_attribute_handler(const esp_zb_zcl_set_attr_value_message_t *message) {
  esp_err_t ret = ESP_OK;
  bool light_state = 0;

  if (!message) {
    log_e("Empty message");
  }
  if (message->info.status != ESP_ZB_ZCL_STATUS_SUCCESS) {
    log_e("Received message: error status(%d)", message->info.status);
  }

  log_i(
    "Received message: endpoint(%d), cluster(0x%x), attribute(0x%x), data size(%d)", message-
>info.dst_endpoint, message->info.cluster, message->attribute.id,
    message->attribute.data.size
  );
```

```
  if (message->info.dst_endpoint == HA_ESP_LIGHT_ENDPOINT) {
    if (message->info.cluster == ESP_ZB_ZCL_CLUSTER_ID_ON_OFF) {
      if (message->attribute.id == ESP_ZB_ZCL_ATTR_ON_OFF_ON_OFF_ID && message->attribute.data.type == ESP_ZB_ZCL_ATTR_TYPE_BOOL) {
        light_state = message->attribute.data.value ? *(bool *)message->attribute.data.value : light_state;
        log_i("Light sets to %s", light_state ? "On" : "Off");
        neopixelWrite(LED_PIN, 255 * light_state, 255 * light_state, 255 * light_state);  // Toggle light
      }
    }
  }
  return ret;
}

/********************* Arduino functions **************************/
void setup() {
  // Init Zigbee
  esp_zb_platform_config_t config = {
    .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
    .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
  };
  ESP_ERROR_CHECK(esp_zb_platform_config(&config));

  // Init RMT and leave light OFF
  neopixelWrite(LED_PIN, 0, 0, 0);

  // Start Zigbee task
  xTaskCreate(esp_zb_task, "Zigbee_main", 4096, NULL, 5, NULL);
}

void loop() {
  //empty, zigbee running in task
}
```

Notas (part II/II)

Este exemplo mostra como configurar o **coordenador Zigbee** e utilizá-lo como interruptor de luz de ligar/desligar para domótica.

Requisitos

- Uma placa de desenvolvimento (ESP32-C6) em modo **coordenador Zigbee** (carregada com o exemplo **Interruptor Zigbee**)
- fios de ligação
- interruptor

Configurar o projeto

Confirmar a definição do GPIO do interruptor Switch consultando a definição de `GPIO_INPUT_IO_TOGGLE_SWITCH`.

2Pp0dOk.png

Arduino IDE

- Antes de Compilar/Verificar, seleccionar a placa correcta: `Ferramentas -> Placa`.
- Selecionar o modo Coordenador Zigbee: `Ferramentas -> Modo Zigbee: Zigbee ZCZR (coordenador)`.
- Selecionar Esquema de Partição para Zigbee: `Ferramentas -> Esquema de Partição: Zigbee 4MB com spiffs`.
- Seleccionar a porta COM: `Ferramentas -> Porta: xxx em que `xxx` é a porta COM detectada.

Interruptor Zigbee

```
// Copyright 2023 Espressif Systems (Shanghai) PTE LTD
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at

//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

/**
 * @brief This example demonstrates simple Zigbee light switch.
 *
 * The example demonstrates how to use ESP Zigbee stack to control a light bulb.
```

```
 * The light bulb is a Zigbee end device, which is controlled by a Zigbee coordinator.
 * Button switch and Zigbee runs in separate tasks.
 *
 * Proper Zigbee mode must be selected in Tools->Zigbee mode
 * and also the correct partition scheme must be selected in Tools->Partition Scheme.
 *
 * Please check the README.md for instructions and more detailed description.
 */

#ifndef ZIGBEE_MODE_ZCZR
#error "Zigbee coordinator mode is not selected in Tools->Zigbee mode"
#endif

#include "esp_zigbee_core.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "ha/esp_zigbee_ha_standard.h"

/* Switch configuration */
#define GPIO_INPUT_IO_TOGGLE_SWITCH GPIO_NUM_9
#define PAIR_SIZE(TYPE_STR_PAIR)    (sizeof(TYPE_STR_PAIR) / sizeof(TYPE_STR_PAIR[0]))

typedef enum {
  SWITCH_ON_CONTROL,
  SWITCH_OFF_CONTROL,
  SWITCH_ONOFF_TOGGLE_CONTROL,
  SWITCH_LEVEL_UP_CONTROL,
  SWITCH_LEVEL_DOWN_CONTROL,
  SWITCH_LEVEL_CYCLE_CONTROL,
  SWITCH_COLOR_CONTROL,
} switch_func_t;

typedef struct {
  uint8_t pin;
  switch_func_t func;
} switch_func_pair_t;

typedef enum {
  SWITCH_IDLE,
  SWITCH_PRESS_ARMED,
```

```
  SWITCH_PRESS_DETECTED,
  SWITCH_PRESSED,
  SWITCH_RELEASE_DETECTED,
} switch_state_t;

static switch_func_pair_t button_func_pair[] = {{GPIO_INPUT_IO_TOGGLE_SWITCH,
SWITCH_ONOFF_TOGGLE_CONTROL}};

/* Default Coordinator config */
#define
ESP_ZB_ZC_CONFIG()
            \

{
                  \
    .esp_zb_role = ESP_ZB_DEVICE_TYPE_COORDINATOR, .install_code_policy =
INSTALLCODE_POLICY_ENABLE, .nwk_cfg = { \
      .zczr_cfg
=
    \

{
            \
          .max_children =
MAX_CHILDREN,                                                                  \

},
            \

}
                \
  }

#define ESP_ZB_DEFAULT_RADIO_CONFIG() \
  { .radio_mode = ZB_RADIO_MODE_NATIVE, }

#define ESP_ZB_DEFAULT_HOST_CONFIG() \
  { .host_connection_mode = ZB_HOST_CONNECTION_MODE_NONE, }

typedef struct light_bulb_device_params_s {
```

```
  esp_zb_ieee_addr_t ieee_addr;
  uint8_t endpoint;
  uint16_t short_addr;
} light_bulb_device_params_t;

/* Zigbee configuration */
#define MAX_CHILDREN                10                                      /* the max amount of
connected devices */
#define INSTALLCODE_POLICY_ENABLE   false                                   /* enable the install
code policy for security */
#define HA_ONOFF_SWITCH_ENDPOINT    1                                       /* esp light switch
device endpoint */
#define ESP_ZB_PRIMARY_CHANNEL_MASK ESP_ZB_TRANSCEIVER_ALL_CHANNELS_MASK /* Zigbee primary
channel mask use in the example */

/********************* Zigbee functions **************************/
static void esp_zb_buttons_handler(switch_func_pair_t *button_func_pair) {
  if (button_func_pair->func == SWITCH_ONOFF_TOGGLE_CONTROL) {
    /* implemented light switch toggle functionality */
    esp_zb_zcl_on_off_cmd_t cmd_req;
    cmd_req.zcl_basic_cmd.src_endpoint = HA_ONOFF_SWITCH_ENDPOINT;
    cmd_req.address_mode = ESP_ZB_APS_ADDR_MODE_DST_ADDR_ENDP_NOT_PRESENT;
    cmd_req.on_off_cmd_id = ESP_ZB_ZCL_CMD_ON_OFF_TOGGLE_ID;
    log_i("Send 'on_off toggle' command");
    esp_zb_zcl_on_off_cmd_req(&cmd_req);
  }
}

static void bdb_start_top_level_commissioning_cb(uint8_t mode_mask) {
  ESP_ERROR_CHECK(esp_zb_bdb_start_top_level_commissioning(mode_mask));
}

static void bind_cb(esp_zb_zdp_status_t zdo_status, void *user_ctx) {
  if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
    log_i("Bound successfully!");
    if (user_ctx) {
      light_bulb_device_params_t *light = (light_bulb_device_params_t *)user_ctx;
      log_i("The light originating from address(0x%x) on endpoint(%d)", light->short_addr,
light->endpoint);
      free(light);
```

```
    }
  }
}

static void user_find_cb(esp_zb_zdp_status_t zdo_status, uint16_t addr, uint8_t endpoint, void
*user_ctx) {
  if (zdo_status == ESP_ZB_ZDP_STATUS_SUCCESS) {
    log_i("Found light");
    esp_zb_zdo_bind_req_param_t bind_req;
    light_bulb_device_params_t *light = (light_bulb_device_params_t
*)malloc(sizeof(light_bulb_device_params_t));
    light->endpoint = endpoint;
    light->short_addr = addr;
    esp_zb_ieee_address_by_short(light->short_addr, light->ieee_addr);
    esp_zb_get_long_address(bind_req.src_address);
    bind_req.src_endp = HA_ONOFF_SWITCH_ENDPOINT;
    bind_req.cluster_id = ESP_ZB_ZCL_CLUSTER_ID_ON_OFF;
    bind_req.dst_addr_mode = ESP_ZB_ZDO_BIND_DST_ADDR_MODE_64_BIT_EXTENDED;
    memcpy(bind_req.dst_address_u.addr_long, light->ieee_addr, sizeof(esp_zb_ieee_addr_t));
    bind_req.dst_endp = endpoint;
    bind_req.req_dst_addr = esp_zb_get_short_address();
    log_i("Try to bind On/Off");
    esp_zb_zdo_device_bind_req(&bind_req, bind_cb, (void *)light);
  }
}

void esp_zb_app_signal_handler(esp_zb_app_signal_t *signal_struct) {
  uint32_t *p_sg_p = signal_struct->p_app_signal;
  esp_err_t err_status = signal_struct->esp_err_status;
  esp_zb_app_signal_type_t sig_type = (esp_zb_app_signal_type_t)*p_sg_p;
  esp_zb_zdo_signal_device_annce_params_t *dev_annce_params = NULL;
  switch (sig_type) {
    case ESP_ZB_ZDO_SIGNAL_SKIP_STARTUP:
      log_i("Zigbee stack initialized");
      esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_INITIALIZATION);
      break;
    case ESP_ZB_BDB_SIGNAL_DEVICE_FIRST_START:
    case ESP_ZB_BDB_SIGNAL_DEVICE_REBOOT:
      if (err_status == ESP_OK) {
        log_i("Device started up in %s factory-reset mode", esp_zb_bdb_is_factory_new() ? "" :
```

```
"non");
        if (esp_zb_bdb_is_factory_new()) {
          log_i("Start network formation");
          esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_NETWORK_FORMATION);
        } else {
          log_i("Device rebooted");
          log_i("Opening network for joining for %d seconds", 180);
          esp_zb_bdb_open_network(180);
        }
      } else {
        log_e("Failed to initialize Zigbee stack (status: %s)", esp_err_to_name(err_status));
      }
      break;
    case ESP_ZB_BDB_SIGNAL_FORMATION:
      if (err_status == ESP_OK) {
        esp_zb_ieee_addr_t extended_pan_id;
        esp_zb_get_extended_pan_id(extended_pan_id);
        log_i(
          "Formed network successfully (Extended PAN ID:
%02x:%02x:%02x:%02x:%02x:%02x:%02x:%02x, PAN ID: 0x%04hx, Channel:%d, Short Address:
0x%04hx)",
          extended_pan_id[7], extended_pan_id[6], extended_pan_id[5], extended_pan_id[4],
extended_pan_id[3], extended_pan_id[2], extended_pan_id[1],
          extended_pan_id[0], esp_zb_get_pan_id(), esp_zb_get_current_channel(),
esp_zb_get_short_address()
        );
        esp_zb_bdb_start_top_level_commissioning(ESP_ZB_BDB_MODE_NETWORK_STEERING);
      } else {
        log_i("Restart network formation (status: %s)", esp_err_to_name(err_status));
        esp_zb_scheduler_alarm((esp_zb_callback_t)bdb_start_top_level_commissioning_cb,
ESP_ZB_BDB_MODE_NETWORK_FORMATION, 1000);
      }
      break;
    case ESP_ZB_BDB_SIGNAL_STEERING:
      if (err_status == ESP_OK) {
        log_i("Network steering started");
      }
      break;
    case ESP_ZB_ZDO_SIGNAL_DEVICE_ANNCE:
      dev_annce_params = (esp_zb_zdo_signal_device_annce_params_t
```

```
*)esp_zb_app_signal_get_params(p_sg_p);
      log_i("New device commissioned or rejoined (short: 0x%04hx)", dev_annce_params-
>device_short_addr);
      esp_zb_zdo_match_desc_req_param_t cmd_req;
      cmd_req.dst_nwk_addr = dev_annce_params->device_short_addr;
      cmd_req.addr_of_interest = dev_annce_params->device_short_addr;
      esp_zb_zdo_find_on_off_light(&cmd_req, user_find_cb, NULL);
      break;
    case ESP_ZB_NWK_SIGNAL_PERMIT_JOIN_STATUS:
      if (err_status == ESP_OK) {
        if (*(uint8_t *)esp_zb_app_signal_get_params(p_sg_p)) {
          log_i("Network(0x%04hx) is open for %d seconds", esp_zb_get_pan_id(), *(uint8_t
*)esp_zb_app_signal_get_params(p_sg_p));
        } else {
          log_w("Network(0x%04hx) closed, devices joining not allowed.", esp_zb_get_pan_id());
        }
      }
      break;
    default: log_i("ZDO signal: %s (0x%x), status: %s", esp_zb_zdo_signal_to_string(sig_type),
sig_type, esp_err_to_name(err_status)); break;
  }
}

static void esp_zb_task(void *pvParameters) {
  esp_zb_cfg_t zb_nwk_cfg = ESP_ZB_ZC_CONFIG();
  esp_zb_init(&zb_nwk_cfg);
  esp_zb_on_off_switch_cfg_t switch_cfg = ESP_ZB_DEFAULT_ON_OFF_SWITCH_CONFIG();
  esp_zb_ep_list_t *esp_zb_on_off_switch_ep =
esp_zb_on_off_switch_ep_create(HA_ONOFF_SWITCH_ENDPOINT, &switch_cfg);
  esp_zb_device_register(esp_zb_on_off_switch_ep);
  esp_zb_set_primary_network_channel_set(ESP_ZB_PRIMARY_CHANNEL_MASK);
  ESP_ERROR_CHECK(esp_zb_start(false));
  esp_zb_main_loop_iteration();
}

/********************* GPIO functions **************************/
static QueueHandle_t gpio_evt_queue = NULL;

static void IRAM_ATTR gpio_isr_handler(void *arg) {
  xQueueSendFromISR(gpio_evt_queue, (switch_func_pair_t *)arg, NULL);
```

```
}

static void switch_gpios_intr_enabled(bool enabled) {
  for (int i = 0; i < PAIR_SIZE(button_func_pair); ++i) {
    if (enabled) {
      enableInterrupt((button_func_pair[i]).pin);
    } else {
      disableInterrupt((button_func_pair[i]).pin);
    }
  }
}

/********************* Arduino functions **************************/
void setup() {
  // Init Zigbee
  esp_zb_platform_config_t config = {
    .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
    .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
  };

  ESP_ERROR_CHECK(esp_zb_platform_config(&config));

  // Init button switch
  for (int i = 0; i < PAIR_SIZE(button_func_pair); i++) {
    pinMode(button_func_pair[i].pin, INPUT_PULLUP);
    /* create a queue to handle gpio event from isr */
    gpio_evt_queue = xQueueCreate(10, sizeof(switch_func_pair_t));
    if (gpio_evt_queue == 0) {
      log_e("Queue was not created and must not be used");
      while (1);
    }
    attachInterruptArg(button_func_pair[i].pin, gpio_isr_handler, (void *)(button_func_pair +
i), FALLING);
  }

  // Start Zigbee task
  xTaskCreate(esp_zb_task, "Zigbee_main", 4096, NULL, 5, NULL);
}

void loop() {
```

```
  // Handle button switch in loop()
  uint8_t pin = 0;
  switch_func_pair_t button_func_pair;
  static switch_state_t switch_state = SWITCH_IDLE;
  bool evt_flag = false;

  /* check if there is any queue received, if yes read out the button_func_pair */
  if (xQueueReceive(gpio_evt_queue, &button_func_pair, portMAX_DELAY)) {
    pin = button_func_pair.pin;
    switch_gpios_intr_enabled(false);
    evt_flag = true;
  }
  while (evt_flag) {
    bool value = digitalRead(pin);
    switch (switch_state) {
      case SWITCH_IDLE:           switch_state = (value == LOW) ? SWITCH_PRESS_DETECTED :
SWITCH_IDLE; break;
      case SWITCH_PRESS_DETECTED: switch_state = (value == LOW) ? SWITCH_PRESS_DETECTED :
SWITCH_RELEASE_DETECTED; break;
      case SWITCH_RELEASE_DETECTED:
        switch_state = SWITCH_IDLE;
        /* callback to button_handler */
        (*esp_zb_buttons_handler)(&button_func_pair);
        break;
      default: break;
    }
    if (switch_state == SWITCH_IDLE) {
      switch_gpios_intr_enabled(true);
      evt_flag = false;
      break;
    }
    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}
```

Exemplo

8_material_list.png

Referência

[1] Espressif. arduino-esp32 Zigbee examples. url: https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples/Zigbee (acedido em 30/08/2024).