LIGA ACADÊMICA DE SISTEMAS EMBARCADOS
CENTRO DE INFORMÁTICA · UFPE

FORMAÇÃO EM ESP32 · MÓDULO 3

A mesma estação, **agora em ESP-IDF**

O mesmo circuito, outro framework.

LUCAS RAYAN GUERRA
CIÊNCIA EMBARCADA · ESPDOCS
LINKEDIN.COM/IN/LUCASRGUERRA

Mesmo projeto, outro caminho

Fica igual

- O circuito e os pinos
- DHT22 + LCD 16×2 I2C
- O resultado no display

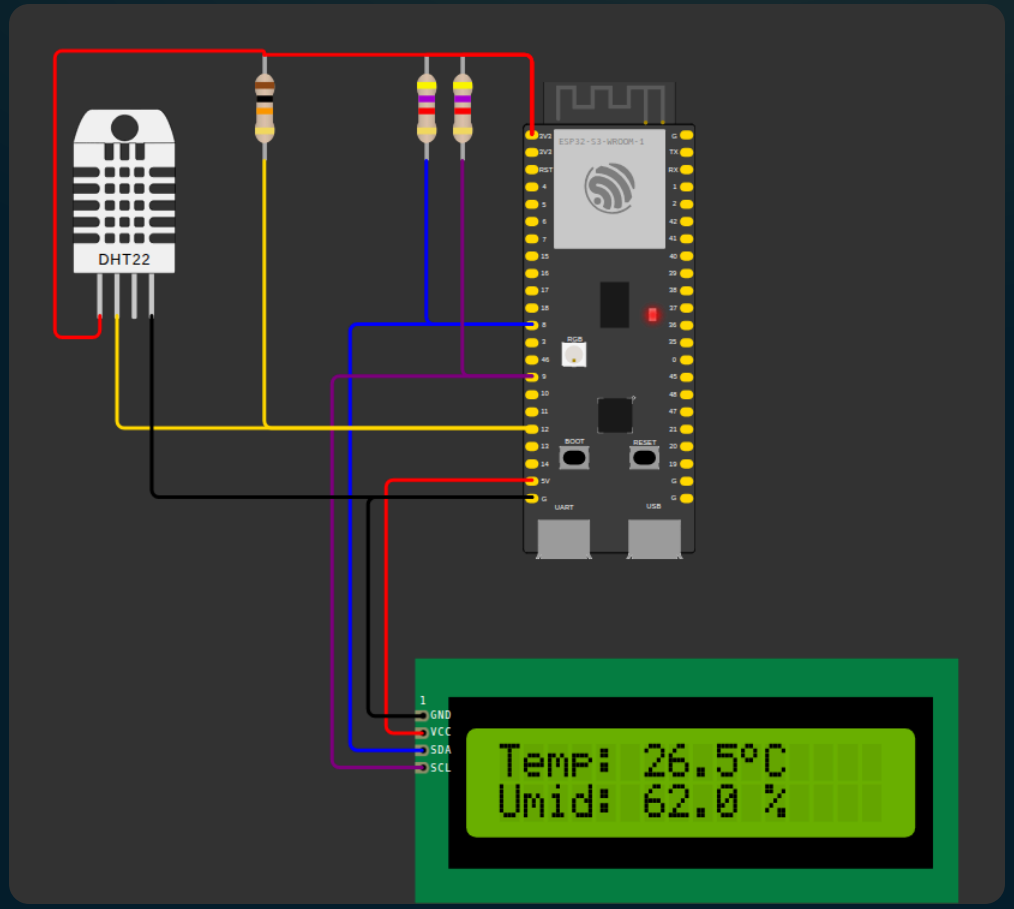
Muda

- Framework: **ESP-IDF**
- Linguagem: **C**
- Compila no seu PC, não no navegador

Isto, rodando no Wokwi

```
Temp: 26.5°C
Umid: 62.0 %
```

```
I (2345) estacao:
Temperatura: 26.5 C
Umidade: 62.0 %
```



PARTE UM

Do Arduino ao ESP-IDF

Você já usava o ESP-IDF. Só não via.

O Arduino roda em cima do ESP-IDF



setup() e loop() viram app_main() e tarefas

Arduino

```
void setup() {
  // uma vez
}

void loop() {
  // para sempre
}
```

ESP-IDF FreeRTOS

```
void app_main(void)
{
  // uma vez, e cria as tarefas
  xTaskCreate(tarefa_estacao, "estacao", 4096, NULL, 5, NULL);
}

static void tarefa(void *arg) {
  while (true) { /* para sempre */ }
}
```

No Arduino, o **loop()** já era uma tarefa do FreeRTOS. Agora você a cria.

O que se ganha, o que custa

Ganha

- Controle de cada periférico
- menuconfig
- Drivers oficiais da Espressif
- Logs e erros explícitos

Custa

- Mais código para o mesmo efeito
- Compilar na sua máquina
- Sem as bibliotecas prontas do Arduino

PARTE DOIS

Anatomia do projeto

Pastas, CMake, menuconfig e quatro comandos.



Uma pasta, um componente

```
idf/
├── CMakeLists.txt       projeto
├── sdkconfig.defaults   ajustes
├── wokwi.toml           simulador
├── diagram.json         circuito
└── main/
    ├── CMakeLists.txt   componente
    ├── Kconfig.projbuild
    ├── estacao.c        app_main
    ├── dht22.c .h       sensor
    └── lcd1602.c .h     display
```

main/

O componente com o **seu** código.

build/

Nasce no `idf.py build`.

Dois CMakeLists, três linhas cada

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(estacao)
```

main/CMakeLists.txt

```
idf_component_register(SRCS "estacao.c" "lcd1602.c" "dht22.c"
                       INCLUDE_DIRS "."
                       REQUIRES esp_driver_i2c esp_driver_gpio esp_timer esp_rom)
```

REQUIRES: os componentes do IDF que o `main` usa.

Kconfig → menuconfig → C

Kconfig.projbuild

```
config PINO_DHT
    int "GPIO do DATA do DHT22"
    default 12
```

você declara

→

idf.py menuconfig

```
Estação de temperatura e umidade
(12) GPIO do DATA do DHT22
(8)  GPIO do SDA (I2C)
(9)  GPIO do SCL (I2C)
(0x27) Endereço I2C do LCD
```

você escolhe

→

sdkconfig.h

```
#define CONFIG_PINO_DHT 12
```

o C lê

Quatro comandos

1

Alvo

`idf.py set-target esp32s3`

2

Compila

`idf.py build`

3

Grava

`idf.py flash`



Monitora

`idf.py monitor`

Na placa de verdade, tudo junto: `idf.py -p /dev/ttyUSB0 flash monitor` (COMx no Windows).

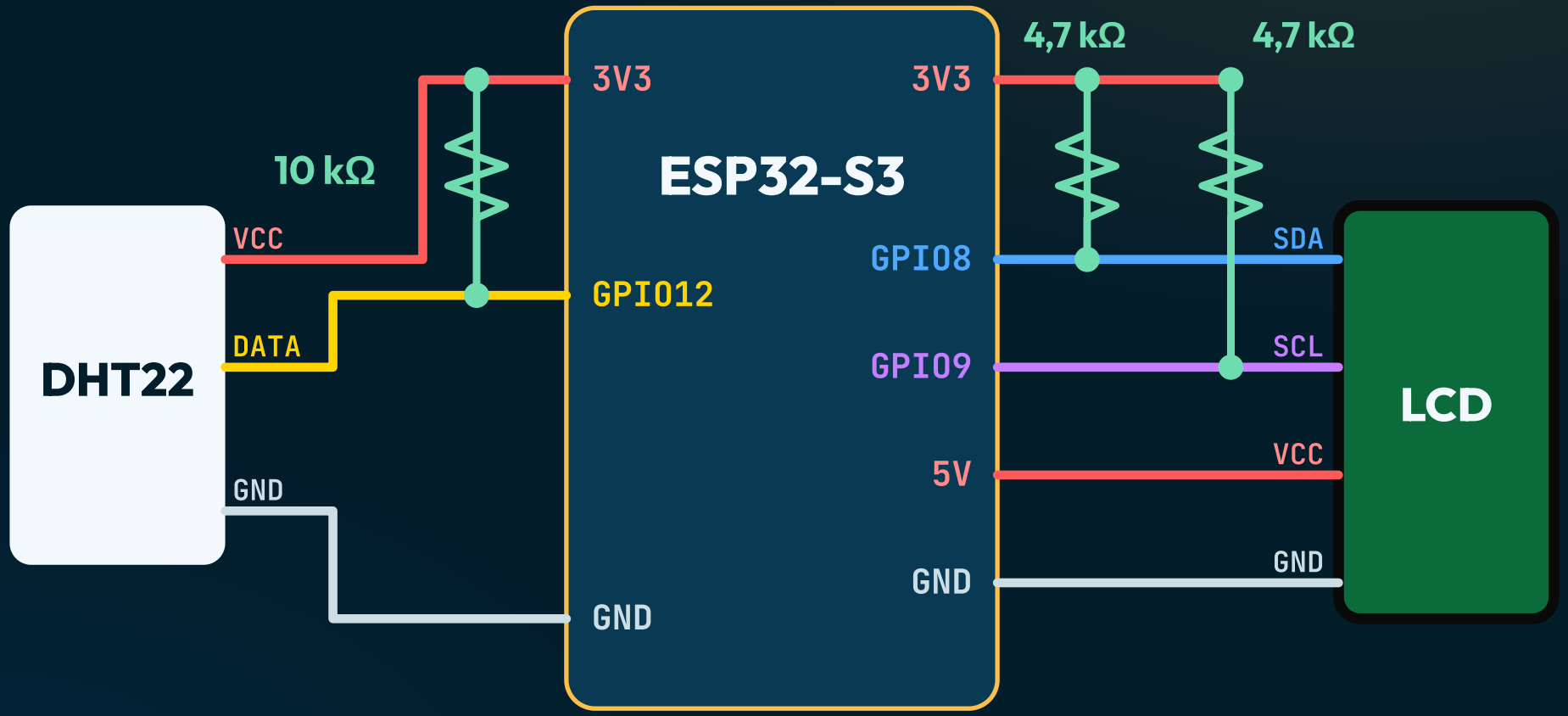
PARTE TRÊS

O circuito

O mesmo do módulo 2, em um slide.

3

Seis fios e três resistores



Pull-ups sempre no **3V3**. Os internos do ESP32-S3 não substituem os externos.

PARTE QUATRO

O código

Do app_main ao pixel do display.



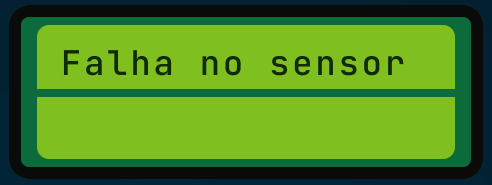
app_main(): preparar e criar a tarefa

```c
void app_main(void)
{
    esp_err_t err = lcd1602_init(&lcd, CONFIG_PINO_SDA, CONFIG_PINO_SCL, CONFIG_LCD_ENDERECO);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "LCD não respondeu (%s). Confira SDA, SCL, alimentação e o endereço 0x%02X",
                 esp_err_to_name(err), CONFIG_LCD_ENDERECO);
    }
    ESP_ERROR_CHECK_WITHOUT_ABORT(lcd1602_create_char(&lcd, 0, GRAU));
    linha(0, "LASER  ESP-IDF");
    linha(1, "Iniciando...");
    vTaskDelay(pdMS_TO_TICKS(2000));       // o DHT22 precisa de ~2 s para a primeira leitura
    lcd1602_clear(&lcd);

    xTaskCreate(tarefa_estacao, "estacao", 4096, NULL, 5, NULL);
}
```

app_main() roda uma vez e termina. Quem continua é a tarefa.

A tarefa: ler, e checar o erro

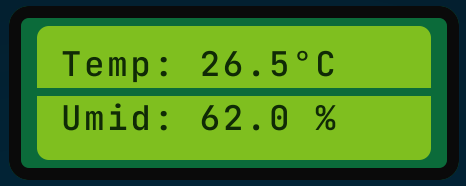
```
static void tarefa_estacao(void *arg)
{
    while (true) {
        float temperatura, umidade;
        esp_err_t err = dht22_ler(CONFIG_PINO_DHT, &temperatura, &umidade);

        if (err != ESP_OK) {
            ESP_LOGE(TAG, "Falha na leitura do DHT22: %s", esp_err_to_name(err));
            linha(0, "Falha no sensor");
            linha(1, "");
        } else {
```



esp_err_t: se não for ESP_OK, avise. Nada de número inventado.

A tarefa: mostrar, e dormir 2 s

```
    char buf[17];
    // \010 (código 8) desenha o slot 0 da CGRAM: o símbolo de grau
    snprintf(buf, sizeof(buf), "Temp: %.1f\010C", temperatura);
    linha(0, buf);
    snprintf(buf, sizeof(buf), "Umid: %.1f %%", umidade);
    linha(1, buf);
    ESP_LOGI(TAG, "Temperatura: %.1f C | Umidade: %.1f %%", temperatura, umidade);
}
vTaskDelay(pdMS_TO_TICKS(2000));   // o DHT22 aceita 1 leitura a cada 2 s
```



vTaskDelay() libera a CPU enquanto espera.

Erros e logs, sem adivinhar

`esp_err_t`
o retorno de quase tudo

`ESP_OK` `ESP_ERR_TIMEOUT` `ESP_ERR_INVALID_CRC`

`ESP_LOGI`
`ESP_LOGE`
com tag e tempo

```
I (2345) estacao: Temperatura: 26.5 C | Umidade: 62.0 %
E (4412) estacao: Falha na leitura do DHT22: ESP_ERR_TIMEOUT
```

`ESP_ERROR_CHECK`
quando não dá para seguir

falhou: **aborta e reinicia**

`_WITHOUT_ABORT`: loga e segue

O driver I2C novo: bus, device, transmit

1

`i2c_new_master_bus()`

```
.sda_io_num
.scl_io_num
.i2c_port
```

`bus`

→

2

`i2c_master_bus_add_device()`

```
.device_address
.scl_speed_hz
```

`dev`

→

3

`i2c_master_transmit()`

```
dev, bytes,
tamanho, timeout
```

`esp_err_t`

Cabeçalho: `driver/i2c_master.h`. Cada camada devolve um handle para a próxima.

Bus e device no lcd1602_init()

```c
// Barramento I2C novo (driver/i2c_master.h)
i2c_master_bus_config_t bus_cfg = {
    .i2c_port = I2C_NUM_0,
    .sda_io_num = sda,
    .scl_io_num = scl,
    .clk_source = I2C_CLK_SRC_DEFAULT,
    .glitch_ignore_cnt = 7,
    // Pull-up interno é fraco (~45 kΩ): NÃO substitui os 4,7 kΩ externos
    .flags.enable_internal_pullup = true,
};
i2c_master_bus_handle_t bus;
ESP_ERROR_CHECK(i2c_new_master_bus(&bus_cfg, &bus));

// O LCD é um dispositivo nesse barramento
i2c_device_config_t dev_cfg = {
    .dev_addr_length = I2C_ADDR_BIT_LEN_7,
    .device_address = endereco,
    .scl_speed_hz = 100000,
};
ESP_ERROR_CHECK(i2c_master_bus_add_device(bus, &dev_cfg, &lcd->dev));
```

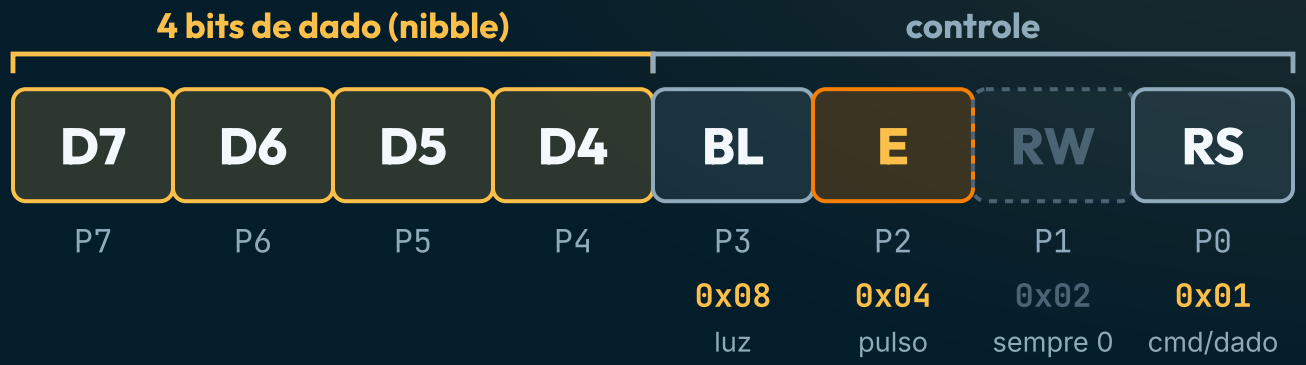
Pull-up interno

Ligado, mas fraco: os **4,7 kΩ** externos continuam.

100 kHz

Velocidade padrão do I2C.

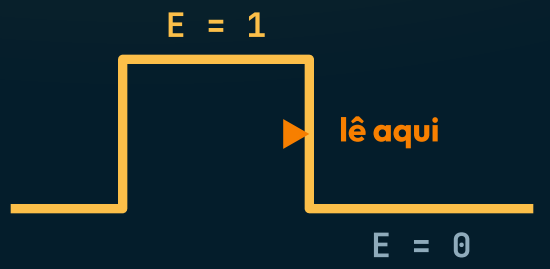
O PCF8574: 1 byte comanda o LCD




```
#define BIT_RS  0x01
#define BIT_E   0x04
#define BIT_BL  0x08
```

Um byte = dois nibbles + um pulso de E

```c
static esp_err_t enviar_nibble(lcd1602_t *lcd, uint8_t nibble, uint8_t rs)
{
    uint8_t base = (nibble << 4) | BIT_BL | rs;
    uint8_t seq[2] = { base | BIT_E, base };
    esp_err_t err = i2c_master_transmit(lcd->dev, seq, sizeof(seq), TIMEOUT_MS);
    esp_rom_delay_us(50);
    return err;
}

static esp_err_t enviar(lcd1602_t *lcd, uint8_t valor, uint8_t rs)
{
    esp_err_t err = enviar_nibble(lcd, valor >> 4, rs);
    return err != ESP_OK ? err : enviar_nibble(lcd, valor & 0x0F, rs);
}
```



O LCD captura o dado na **descida** do E.

DHT22: pedir a leitura, sem interrupções

```
// Open-drain: o ESP só puxa para 0; o resistor de 10 k externo leva a 1
gpio_set_direction(pino, GPIO_MODE_INPUT_OUTPUT_OD);
gpio_set_level(pino, 1);

// Pedido de leitura: linha em 0 por ~2 ms, depois solta
gpio_set_level(pino, 0);
esp_rom_delay_us(2000);

// Os bits duram ~50 us: sem interrupções até o fim da leitura (~4 ms)
static portMUX_TYPE mux = portMUX_INITIALIZER_UNLOCKED;
portENTER_CRITICAL(&mux);
```

Open-drain

O ESP só puxa para 0. O resistor leva a 1.

Seção crítica

Um bit dura ~50 µs. Nada pode atrasar.

DHT22: o tempo em 1 decide o bit

```
// 40 bits: 50 us em 0, depois o tempo em 1 decide o bit (~27 us = 0, ~70 us = 1)
for (int i = 0; i < 40 && erro == ESP_OK; i++) {
    if (esperar_nivel(pino, 1, 100) < 0) { erro = ESP_ERR_TIMEOUT; break; }
    int alto = esperar_nivel(pino, 0, 120);
    if (alto < 0) { erro = ESP_ERR_TIMEOUT; break; }
    dado[i / 8] <<= 1;
    if (alto > 40) {
        dado[i / 8] |= 1;
    }
}
portEXIT_CRITICAL(&mux);
```

~27 µs
bit 0

~70 µs
bit 1

O símbolo de grau na CGRAM



0b01100
0b10010
0b10010
0b01100
0
0
0
0

```
static const uint8_t GRAU[8] = { 0b01100, 0b10010, 0b10010, 0b01100, 0, 0, 0, 0 };

ESP_ERROR_CHECK_WITHOUT_ABORT(lcd1602_create_char(&lcd, 0, GRAU));

snprintf(buf, sizeof(buf), "Temp: %.1f\010C", temperatura);
```

PARTE CINCO

Compilar e simular

O Wokwi online não compila ESP-IDF. Ele simula um firmware pronto.



Compile antes, simule depois

OPÇÃO A

No navegador

1. `idf.py merge-bin -o estacao-esp32s3.bin`
2. Abra `wokwi.com/projects/new/esp32-s3` e cole o `diagram.json`
3. **F1** → Upload Firmware and Start Simulation
4. Escolha o `estacao-esp32s3.bin`

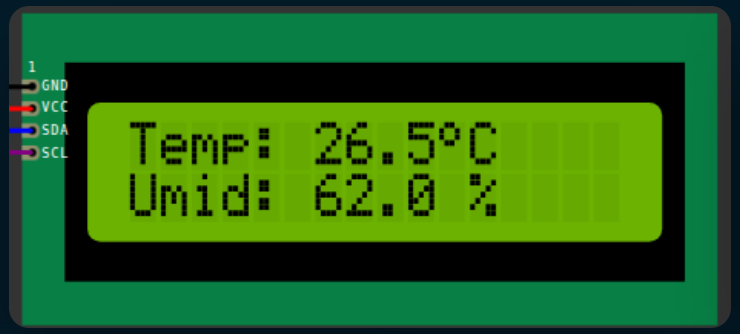
OPÇÃO B

No VS Code

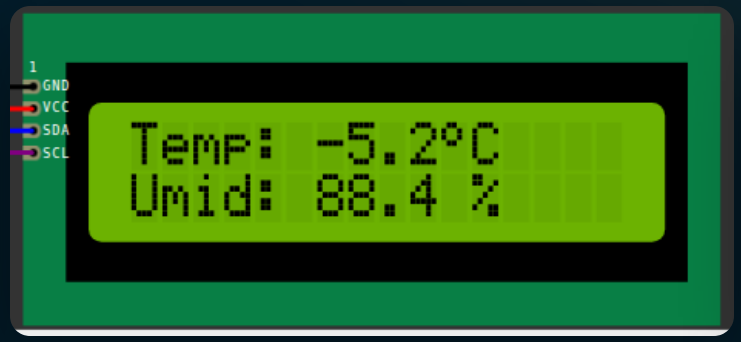
1. `idf.py build`, abra a pasta
2. **F1** → Wokwi: Start Simulator

```
[wokwi]
version = 1
firmware = 'build/flasher_args.json'
elf = 'build/estacao.elf'
```

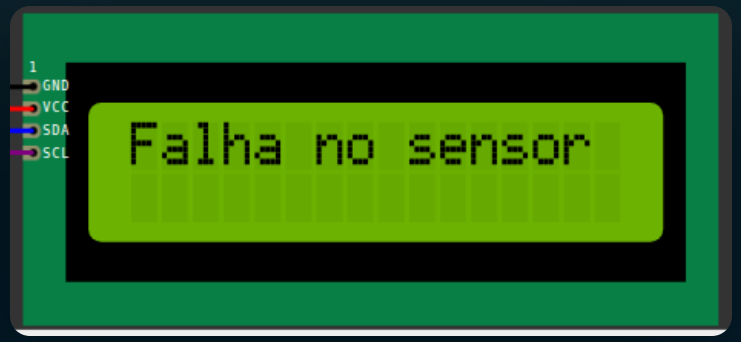
Três testes, três telas



Sala
26,5 °C · 62 %



Frio
−5,2 °C · 88,4 %



Fio solto
erro detectado

Na simulação, clique no DHT22 e mova os controles de temperatura e umidade.

PARTE SEIS

Da tela para a bancada

O que o Wokwi perdoa e o hardware não.

A plaquinha I2C traz pull-ups para 5 V

≈ 4 V

no GPIO, com a plaquinha em 5 V,
mesmo com os 4,7 kΩ para 3V3

✗ fora da especificação

3,3 V

Conversor de nível BSS138
ou remover os pull-ups da plaquinha

✓ seguro

Teste: multímetro entre SDA e VCC da plaquinha. Entre 2 e 10 kΩ, há um pull-up.

Tela vazia? Descubra o endereço

```
for (uint8_t endereco = 1; endereco < 127; endereco++) {
    // i2c_master_probe devolve ESP_OK se alguém responder (ACK)
    if (i2c_master_probe(bus, endereco, 50) == ESP_OK) {
        ESP_LOGI(TAG, "Dispositivo em 0x%02X", endereco);
        encontrados++;
    }
}
```

```
I (305) scanner: Procurando dispositivos I2C...
I (315) scanner: Dispositivo em 0x27
```

Saída esperada do `idf-scanner-i2c`.
Na sua placa pode aparecer **0x3F**.

Quando não funciona



"Falha no sensor"

Pull-up de 10 kΩ, fio no GPIO12, VCC e GND.



Aceso e vazio

Contraste, endereço (menuconfig) ou SDA e SCL trocados.



Linha de blocos

A energia chegou, o I2C não. O monitor diz "LCD não respondeu".



Tudo apagado

Falta o 5V no VCC do LCD.

Desafios



Fila

Leia o sensor em uma tarefa e mostre no LCD em outra, com `xQueueSend()`.



Botão

Troque °C por °F com um botão, por interrupção de GPIO.



NVS

Grave a temperatura máxima na memória, com `nvs_flash`.



menuconfig

Crie um item novo no Kconfig: o limite de alerta.

LIGA ACADÊMICA DE SISTEMAS EMBARCADOS
CENTRO DE INFORMÁTICA · UFPE

MÓDULO 3 · ENCERRAMENTO

Obrigado.
Avalie a formação

Aponte a câmera para o QR code ou abra o link.

qr.codes/YQN2q6

LUCAS RAYAN GUERRA
CONTATO@LUCASRGUERRA.DEV.BR · LINKEDIN.COM/IN/LUCASRGUERRA