[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).

Revision #14
Created 2024-08-30 13:23:15 UTC by João Pedro Monteiro
Updated 2024-09-02 13:34:54 UTC by João Pedro Monteiro