Controle de acesso

1.0

Generated by Doxygen 1.14.0

Chapter 1

File Documentation

1.1 main.c File Reference

Sistema de leitor de SmartCard com comunicação Wi-Fi e LCD.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include "driver/i2c_master.h"
#include "driver/gpio.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_sntp.h"
#include "nvs_flash.h"
#include "esp_http_client.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/event_groups.h"
#include "display_lcd.h"
```

Macros

- #define BFC_PIN 2

  < Cabeçalho para funções de controle do display LCD
- #define **I2C_SCL_PIN** 3

  Pino SCL (Serial Clock) do barramento I2C.
- #define **I2C_SDA_PIN** 4

  Pino SDA (Serial Data) do barramento I2C.
- #define **EEPROM_ADDRESS** 0x50

  Endereço I2C do dispositivo EEPROM (SmartCard)
- #define **DEBOUNCE_DELAY_MS** 100

  Tempo de debounce do botão em milissegundos, para evitar múltiplas leituras por um único clique.
- #define **LED_VERMELHO_PIN** 9

Pino do LED vermelho.

- #define **LED_VERDE_PIN** 0

  Pino do LED verde.
- #define **SERVER_URL** "http://eel-7030-server.azurewebsites.net/"

  URL do servidor para comunicação HTTP.
- #define **AUTH_TOKEN** "Q1w2E3r4T5y6U7i8"

  Token de autenticação para as requisições ao servidor.
- #define **ROOM_NUMBER** "101"

  Número do quarto ao qual o dispositivo ESP32-C3 está associado.
- #define **TAG_HTTP** "HTTP_CLIENT"

  Tag para logs relacionados ao cliente HTTP.
- #define **TAG_SNTP** "SNTP"

  Tag para logs relacionados ao SNTP.
- #define **TAG_WIFI** "Wi-Fi STA"

  Tag para logs relacionados ao Wi-Fi (modo estação)
- #define **TAG_MAIN** "SmartCard Reader"

  Tag para logs principais do sistema leitor de SmartCard.
- #define **TAG_LCD** "LCD"

  Tag para logs do display LCD.
- #define **TAG_LED_VERDE** "LED_VERDE"

  Tag para logs do LED verde.
- #define **WIFI_SSID** "Wokwi-GUEST"

  SSID (nome) da rede Wi-Fi a ser conectada.
- #define **WIFI_PASS** ""

  Senha da rede Wi-Fi (vazia para redes abertas)
- #define **WIFI_CONNECTED_BIT** BIT0

  Bit que será setado no grupo de eventos quando o Wi-Fi estiver conectado.

Enumerations

- enum system_state_t { STATE_CONNECTING_WIFI , STATE_WAITING_CARD , STATE_VERIFYING_CARD , STATE_ACCESS_GRANTED , STATE_ACCESS_DENIED }

Functions

- void configura_barramento_mestre (i2c_master_bus_handle_t *ptr_bus_handle)

  Configura o barramento I2C mestre.
- void configura_dispositivo_smartcard (i2c_master_bus_handle_t bus_handle, i2c_master_dev_handle_t *ptr_dev_handle)

  Configura o dispositivo SmartCard no barramento I2C.
- void ler_dados_cartao (i2c_master_dev_handle_t dev_handle, uint8_t endereco_inicial, uint8_t *buffer, size_t tamanho)

  Lê dados do cartão SmartCard.
- void escrever_timestamp_cartao (i2c_master_dev_handle_t dev_handle, uint64_t timestamp)

  Escreve timestamp no cartão SmartCard.
- void gpio_isr_handler (void *arg)

  Handler de interrupção GPIO.
- void init_gpio ()

Inicializa os GPIOs.

- void cartao_task (void ∗arg)

  Tarefa para processamento do cartão.
- void sntp_sync_task (void ∗pvParameter)

  Tarefa para sincronização SNTP.
- void leds_status_task (void ∗pvParameter)

  Tarefa para gerenciamento dos LEDs e display LCD.
- void nvs_init ()

  Inicializa o NVS (Non-Volatile Storage)
- void event_loop_init ()

  Inicializa o loop de eventos.
- void wifi_init ()

  Inicializa a conexão Wi-Fi.
- esp_err_t http_post_request_event_handler (esp_http_client_event_t ∗evt)

  Handler de eventos HTTP.
- void sntp_callback_sync_time (struct timeval ∗tv)

  Callback de sincronização de tempo SNTP.
- void update_lcd_display (system_state_t state)

  Atualiza o display LCD conforme o estado do sistema.
- void app_main (void)

  Função principal do sistema, ponto de entrada do programa.
- void verify_card_http_post (char ∗card_id_str)

  Realiza requisição HTTP POST para verificar o cartão.

Variables

- i2c_master_bus_handle_t **bus_handle**

  Handle para o barramento I2C mestre.
- i2c_master_dev_handle_t **dev_handle**

  Handle para o dispositivo I2C (EEPROM do SmartCard)

1.1.1 Detailed Description

Sistema de leitor de SmartCard com comunicação Wi-Fi e LCD.

Author

Rebecca Quintino Do Ó

1.1.2 Macro Definition Documentation

1.1.2.1 BFC_PIN

```
#define BFC_PIN 2
```

< Cabeçalho para funções de controle do display LCD

< Inclui funções padrão de entrada e saída Inclui funções para manipulação de strings Inclui funções de uso geral, como alocação de memória e conversão de tipos Para manipulação de tempo (timestamp) Para a função gettimeofday, usada para obter o tempo atual com alta precisão < Driver para comunicação I2C mestre Driver para controle de pinos GPIO API do ESP-IDF para controle de Wi-Fi API do ESP-IDF para gerenciamento de eventos API do ESP-IDF para logging (mensagens de depuração) API do ESP-IDF para protocolo SNTP (sincronização de tempo de rede) API do ESP-IDF para NVS (Non-Volatile Storage), usada para armazenar dados de forma persistente API do ESP-IDF para cliente HTTP Sistema operacional em tempo real FreeRTOS Funções para gerenciamento de tarefas no FreeRTOS Funções para gerenciamento de filas no FreeRTOS (comunicação entre tarefas) Funções para gerenciamento de grupos de eventos no FreeRTOS (sinalização entre tarefas) Pino do botão de fim de curso

1.1.3 Enumeration Type Documentation

1.1.3.1 system_state_t

```
enum system_state_t
```

Enumerator

STATE_CONNECTING_WIFI	Estado: Conectando ao Wi-Fi.
STATE_WAITING_CARD	Estado: Aguardando inserção do cartão.
STATE_VERIFYING_CARD	Estado: Verificando o cartão.
STATE_ACCESS_GRANTED	Estado: Acesso liberado.
STATE_ACCESS_DENIED	Estado: Acesso negado.

1.1.4 Function Documentation

1.1.4.1 app_main()

```
void app_main (
            void )
```

Função principal do sistema, ponto de entrada do programa.

< Criação do Grupo de Eventos Wi-Fi para gerenciar o estado da conexão

< Cria fila para eventos do GPIO (BFC), com capacidade para 10 itens

< Cria fila para estados do sistema, com capacidade para 5 estados

< Define o estado inicial como "Conectando ao Wi-Fi"

< Envia o estado inicial para a fila de estados do sistema (bloqueia indefinidamente se a fila estiver cheia)

< Inicializa o NVS (Non-Volatile Storage) para armazenamento persistente

< Inicializa o loop de eventos do ESP-IDF para eventos de rede

< Inicializa e tenta conectar ao Wi-Fi

< Inicializa o barramento I2C para comunicação com SmartCard e LCD

< Configura o dispositivo SmartCard no barramento I2C

< Configura o display LCD via I2C

< Tenta configurar o display LCD

< Inicializa os pinos GPIO e configura a interrupção para o botão de fim de curso (BFC)

< Cria tarefas FreeRTOS

< Cria a tarefa para processamento do cartão (stack de 4KB, prioridade 10)

< Cria a tarefa para sincronização SNTP (stack de 4KB, prioridade 5)

< Cria a tarefa para gerenciamento dos LEDs e LCD (stack de 2KB, prioridade 5)

1.1.4.2 cartao_task()

```
void cartao_task (
            void * arg)
```

Tarefa para processamento do cartão.

Parameters

arg	Argumento da tarefa (não utilizado)

< Variável para armazenar o número do pino GPIO recebido da fila

< Buffer para a primeira parte do ID do cartão

< Buffer para a segunda parte do ID do cartão

< String para armazenar o ID completo do cartão (16 caracteres + null terminator)

< Aguarda indefinidamente por um evento na fila do GPIO

< Atraso para debounce do botão

< Verifica se o botão ainda está ativado após o debounce

< Define o estado como "Verificando Cartão"

< Envia o estado para a fila do sistema (não bloqueante)

< Lê a primeira parte do ID do cartão (endereço 0x00)

< Lê a segunda parte do ID do cartão (endereço 0x08)

< Copia a primeira parte do ID para a string

< Copia a segunda parte do ID para a string

< Adiciona o terminador nulo à string

< Verifica se o tempo foi sincronizado via SNTP

< Obtém o tempo atual do sistema

< Converte o tempo para um timestamp de 64 bits (segundos)

< Escreve o timestamp no cartão SmartCard

< Converte o timestamp para uma estrutura de tempo legível

< Formata o tempo para uma string legível

< Aguarda indefinidamente até que o Wi-Fi esteja conectado

< Formata os dados JSON para a requisição POST

< URL do servidor

< Método HTTP (POST)

< Handler de eventos HTTP para processar a resposta

< Tempo limite da requisição em milissegundos

< User-Agent para a requisição

< Inicializa o cliente HTTP

< Define o cabeçalho Content-Type como JSON

< Define os dados a serem enviados no corpo da requisição POST

< Realiza a requisição HTTP

< Obtém o código de status HTTP da resposta

< Define o estado como "Acesso Liberado"

< Envia o estado para a fila

< Define o estado como "Acesso Negado"

< Envia o estado para a fila

< Define o estado como "Acesso Negado"

< Envia o estado para a fila

< Limpa os recursos do cliente HTTP

< Reabilita a interrupção para o pino do BFC

< Define o estado como "Aguardando Cartão"

< Envia o estado para a fila

1.1.4.3 configura_barramento_mestre()

```
void configura_barramento_mestre (
            i2c_master_bus_handle_t * ptr_bus_handle)
```

Configura o barramento I2C mestre.

Parameters

ptr_bus_handle	Ponteiro para o handle do barramento I2C

< Fonte do clock do I2C (padrão)

< Porta I2C a ser usada (I2C_NUM_0)

< Pino GPIO para SCL

< Pino GPIO para SDA

< Número de pulsos de glitch para ignorar

< Habilita resistores de pull-up internos

< Cria um novo barramento I2C mestre

1.1.4.4 configura_dispositivo_smartcard()

```
void configura_dispositivo_smartcard (
            i2c_master_bus_handle_t bus_handle,
            i2c_master_dev_handle_t * ptr_dev_handle)
```

Configura o dispositivo SmartCard no barramento I2C.

Parameters

bus_handle	Handle do barramento I2C
ptr_dev_handle	Ponteiro para o handle do dispositivo

< Endereço do dispositivo tem 7 bits

< Endereço I2C da EEPROM do SmartCard

< Velocidade do clock SCL em Hz (100 KHz)

< Adiciona o dispositivo SmartCard ao barramento I2C

1.1.4.5 escrever_timestamp_cartao()

```
void escrever_timestamp_cartao (
            i2c_master_dev_handle_t dev_handle,
            uint64_t timestamp)
```

Escreve timestamp no cartão SmartCard.

Parameters

dev_handle	Handle do dispositivo I2C
timestamp	Timestamp a ser escrito

< Buffer para escrita, 1 byte para endereço + 8 bytes para timestamp

< Endereço inicial na EEPROM para escrita do timestamp

< Buffer para leitura de verificação após a escrita

< Copia o timestamp para o buffer de escrita, após o byte de endereço

< Realiza a transmissão dos dados (endereço + timestamp) via I2C

< Lê os dados recém-escritos para verificação

< Copia os dados lidos de volta para uma variável uint64_t

< Converte o timestamp para time_t

< Converte o timestamp para uma estrutura de tempo legível

< Formata o tempo para uma string legível

< Pequeno atraso para estabilização da EEPROM

1.1.4.6 event_loop_init()

```
void event_loop_init ()
```

Inicializa o loop de eventos.

< Inicializa a camada de interface de rede (netif)

< Cria o loop de eventos padrão do ESP-IDF

< Cria a interface de rede padrão para o modo Wi-Fi estação (STA)

< Registra o handler para todos os eventos Wi-Fi

< Registra o handler para o evento de IP obtido no modo STA

1.1.4.7 gpio_isr_handler()

```
void gpio_isr_handler (
            void * arg)
```

Handler de interrupção GPIO.

Parameters

arg	Argumento passado para o handler (geralmente o número do pino que gerou a interrupção)
arg	Argumento passado para o handler

< Obtém o número do pino GPIO que gerou a interrupção

< Desabilita temporariamente a interrupção para o pino para evitar múltiplos eventos

< Variável para verificar se uma tarefa de maior prioridade foi "acordada"

< Envia o número do pino para a fila de eventos GPIO a partir da ISR

< Força uma troca de contexto se uma tarefa de maior prioridade foi acordada

1.1.4.8 http_post_request_event_handler()

```
esp_err_t http_post_request_event_handler (
            esp_http_client_event_t * evt)
```

Handler de eventos HTTP.

Parameters

evt	Estrutura com dados do evento HTTP

Returns

Código de erro ESP

< Evento de recebimento de dados da resposta HTTP

< Realoca o buffer para acomodar os novos dados

< Libera o buffer antigo em caso de falha na realocação

< Retorna falha

< Copia os novos dados para o buffer

< Atualiza o tamanho da resposta

< Adiciona o terminador nulo

< Evento de finalização da requisição HTTP

< Libera o buffer de resposta

< Evento de desconexão HTTP

< Evento de erro HTTP

< Libera o buffer em caso de desconexão ou erro

< Retorna sucesso

1.1.4.9 init_gpio()

```
void init_gpio ()
```

Inicializa os GPIOs.

< Máscara de bits para o pino do BFC

< Configura o pino como entrada

< Desabilita resistor de pull-up

< Habilita resistor de pull-down (garante estado baixo quando não ativado)

< Configura interrupção para borda de subida (quando o botão é pressionado e o pino vai de baixo para alto)

< Aplica a configuração ao pino do BFC

< Máscara de bits para o pino do LED vermelho

< Configura o pino como saída

< Desabilita pull-up

< Desabilita pull-down

< Desabilita interrupções

< Aplica a configuração ao pino do LED vermelho

< Desliga o LED vermelho inicialmente

< Máscara de bits para o pino do LED verde

< Configura o pino como saída

< Desabilita pull-up

< Desabilita pull-down

< Desabilita interrupções

< Aplica a configuração ao pino do LED verde

< Desliga o LED verde inicialmente

< Instala o serviço de interrupção GPIO

< Adiciona o handler de interrupção para o pino do BFC

1.1.4.10 leds_status_task()

```
void leds_status_task (
            void * pvParameter)
```

Tarefa para gerenciamento dos LEDs e display LCD.

Parameters

pvParameter	Argumento da tarefa (não utilizado)

< Define o estado inicial da tarefa

< Variável para armazenar o estado recebido da fila

< Atualiza o LCD com o estado inicial

< Tenta receber um novo estado da fila com um timeout

< Atualiza o estado atual da tarefa

< Atualiza o display LCD com o novo estado

< Se está conectando ao Wi-Fi

< Liga o LED verde

< Atraso de 1 segundo

< Desliga o LED verde

< Atraso de 1 segundo (efeito de pisca-pisca)

< Se está aguardando o cartão

< Mantém o LED verde ligado

< Se está verificando o cartão

< Desliga o LED verde

< Atraso curto

< Liga o LED verde brevemente

< Retorna ao estado de aguardando cartão

< Se o acesso foi liberado

< Pisca o LED verde duas vezes

< Liga o LED vermelho (indicando acesso liberado)

< Atraso de 2 segundos

< Desliga o LED vermelho

< Retorna ao estado de aguardando cartão

< Envia o novo estado para a fila (para atualizar o LCD)

< Se o acesso foi negado

< Pisca o LED verde seis vezes

< Garante que o LED vermelho esteja desligado (ou o LED vermelho poderia ser usado para indicar negação)

< Retorna ao estado de aguardando cartão

< Envia o novo estado para a fila (para atualizar o LCD)

< Para qualquer outro estado

< Desliga o LED verde

< Desliga o LED vermelho

1.1.4.11 ler_dados_cartao()

```
void ler_dados_cartao (
            i2c_master_dev_handle_t dev_handle,
            uint8_t endereco_inicial,
            uint8_t * buffer,
            size_t tamanho)
```

Lê dados do cartão SmartCard.

Parameters

dev_handle	Handle do dispositivo I2C
endereco_inicial	Endereço inicial de leitura na EEPROM do cartão
buffer	Buffer para armazenar os dados lidos
tamanho	Tamanho dos dados a serem lidos
dev_handle	Handle do dispositivo I2C
endereco_inicial	Endereço inicial de leitura
buffer	Buffer para armazenar os dados lidos
tamanho	Tamanho dos dados a serem lidos

< Realiza a operação de transmissão (endereço) e recepção (dados) via I2C

1.1.4.12 nvs_init()

```
void nvs_init ()
```

Inicializa o NVS (Non-Volatile Storage)

< Tenta inicializar o NVS

< Se não há páginas livres ou uma nova versão é encontrada

< Apaga a flash NVS

< Tenta inicializar novamente

1.1.4.13 sntp_callback_sync_time()

```
void sntp_callback_sync_time (
            struct timeval * tv)
```

Callback de sincronização de tempo SNTP.

Parameters

tv	Estrutura com dados de tempo (struct timeval)
tv	Estrutura com dados de tempo

< Define o fuso horário para GMT-3 (Brasília)

< Atualiza as informações de fuso horário

< Obtém o tempo atual (após a sincronização)

< Converte o tempo para a estrutura tm (tempo local)

< Seta a flag para indicar que o tempo foi sincronizado

1.1.4.14 sntp_sync_task()

```
void sntp_sync_task (
            void * pvParameter)
```

Tarefa para sincronização SNTP.

Parameters

pvParameter	Argumento da tarefa (não utilizado)

< Aguarda indefinidamente até que o Wi-Fi esteja conectado para iniciar a sincronização SNTP

< Define o modo de operação SNTP como "polling"

< Define o servidor SNTP (servidor de tempo do Google)

< Registra a função de callback para notificação de sincronização de tempo

< Define o intervalo de sincronização para 6 horas (em milissegundos)

< Inicializa o cliente SNTP

< Deleta a própria tarefa após a inicialização do SNTP, pois a sincronização ocorrerá em segundo plano e através do callback

1.1.4.15 update_lcd_display()

```
void update_lcd_display (
            system_state_t state)
```

Atualiza o display LCD conforme o estado do sistema.

Parameters

state	Estado do sistema a ser exibido

< Limpa o conteúdo do display LCD

< Posiciona o cursor na primeira coluna, primeira linha

< Caso o estado seja "Conectando ao Wi-Fi"

< Escreve "Conectando"

< Posiciona o cursor na primeira coluna, segunda linha

< Escreve "ao Wi-Fi..."

< Caso o estado seja "Aguardando Cartão"

< Escreve "Aguardando"

< Posiciona o cursor

< Escreve "Cartao..."

< Caso o estado seja "Verificando Cartão"

< Escreve "Verificando"

< Posiciona o cursor

< Escreve "Cartao..."

< Caso o estado seja "Acesso Liberado"

< Escreve "Acesso"

< Posiciona o cursor

< Escreve "Liberado!"

< Caso o estado seja "Acesso Negado"

< Escreve "Acesso"

< Posiciona o cursor

< Escreve "Negado!"

< Para qualquer outro estado não previsto

< Exibe uma mensagem de erro

1.1.4.16 verify_card_http_post()

```
void verify_card_http_post (
            char * card_id_str)
```

Realiza requisição HTTP POST para verificar o cartão.

Parameters

card_id_str	ID do cartão a ser verificado

Note

Esta função parece ser uma duplicação do trecho de código dentro de `cartao_task` e pode ser refatorada.

< Formata os dados JSON para a requisição POST

< URL do servidor

< Método HTTP (POST)

< Handler de eventos HTTP para processar a resposta

< Tempo limite da requisição em milissegundos

< User-Agent para a requisição

< Inicializa o cliente HTTP

< Define o cabeçalho Content-Type como JSON

< Define os dados a serem enviados no corpo da requisição POST

< Realiza a requisição HTTP

< Obtém o código de status HTTP da resposta

< Define o estado como "Acesso Liberado"

< Envia o estado para a fila

< Define o estado como "Acesso Negado"

< Envia o estado para a fila

< Define o estado como "Acesso Negado" (por erro de comunicação)

< Envia o estado para a fila

< Limpa os recursos do cliente HTTP

1.1.4.17 wifi_init()

```
void wifi_init ()
```

Inicializa a conexão Wi-Fi.

< Obtém a configuração padrão de inicialização do Wi-Fi

< Inicializa o subsistema Wi-Fi com as configurações

< Define o modo de operação do Wi-Fi como estação (STA)

< Define o SSID da rede Wi-Fi

< Define a senha da rede Wi-Fi

< Aplica as configurações da rede Wi-Fi

< Inicia o Wi-Fi

Index