SmartBMS_2209A

AUTHOR
Version v1.0.0

Table of Contents

Table of contents

SmartBMS_2209A Documentation

Main Page

Project Name: SmartBMS_2209A v1.0.0

Brief Description: Battery Management System for Tubitak 2209A

Navigation

- Home

Search

Use the search box to find specific content within the documentation.

Footer

Generated by Doxygen 1.13.2.

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Class Documentation

DbcStruct Struct Reference

`#include <d2cc_lib.h>`

Public Attributes

- struct {
- uint32_t **ID**
- uint8_t **DLC**
- union {
- struct {
- } **Signal**
- uint8_t **Data** [8]
- }
- } **Battery_Messages**
- struct {
- uint32_t **ID**
- uint8_t **DLC**
- union {
- struct {
- } **Signal**
- uint8_t **Data** [8]
- }
- } **Battery_Temperatures**
- struct {
- uint32_t **ID**
- uint8_t **DLC**
- union {
- struct {
- } **Signal**
- uint8_t **Data** [8]
- }
- } **Battery_Voltages**
- struct {
- uint32_t **ID**
- uint8_t **DLC**
- union {
- struct {
- uint8_t **AliveCounter**:7
- **BatteryChemistry_enum BatteryChemistry**:1
- uint8_t **BatteryBalance_MaxVoltage**:8
- uint8_t **BatteryBalance_MinVoltage**:8
- uint8_t **BatteryBalance_MaxTemp**:8
- uint8_t **BatteryBalance_MinTemp**:8
- uint8_t **SwVersionMajor**:2
- uint8_t **SwVersionMinor**:2
- uint8_t **SwVersionBugfix**:4
- **CanBusEnable_enum CanBusEnable**:1
- **WiFiEnable_enum WiFiEnable**:1
- **TcpEnable_enum TcpEnable**:2
- **VpnEnable_enum VpnEnable**:1
- **WiFi_AP_Status_enum WiFi_AP_Status**:2
- uint8_t **TcpClientCount**:8
- } **Signal**
- uint8_t **Data** [8]
- }

- struct {
- struct {
- float **factor**
- int **offset**
- float **value**
- } **Phys_Value**
- } **BatteryBalance_MaxVoltage**
- struct {
- struct {
- float **factor**
- int **offset**
- float **value**
- } **Phys_Value**
- } **BatteryBalance_MinVoltage**
- struct {
- struct {
- float **factor**
- int **offset**
- float **value**
- } **Phys_Value**
- } **BatteryBalance_MinTemp**
- } **Can_Main**

Member Data Documentation

uint8_t DbcStruct::AliveCounter

struct { ... } DbcStruct::Battery_Messages

struct { ... } DbcStruct::Battery_Temperatures

struct { ... } DbcStruct::Battery_Voltages

uint8_t DbcStruct::BatteryBalance_MaxTemp

uint8_t DbcStruct::BatteryBalance_MaxVoltage

struct { ... } DbcStruct::BatteryBalance_MaxVoltage

uint8_t DbcStruct::BatteryBalance_MinTemp

struct { ... } DbcStruct::BatteryBalance_MinTemp

uint8_t DbcStruct::BatteryBalance_MinVoltage

struct { ... } DbcStruct::BatteryBalance_MinVoltage

BatteryChemistry_enum DbcStruct::BatteryChemistry

struct { ... } DbcStruct::Can_Main

CanBusEnable_enum DbcStruct::CanBusEnable

uint8_t DbcStruct::Data[8]

uint8_t DbcStruct::DLC

float DbcStruct::factor

uint32_t DbcStruct::ID

int DbcStruct::offset

struct { ... } DbcStruct::Phys_Value

struct { ... } DbcStruct::Phys_Value

struct { ... } DbcStruct::Phys_Value

struct { ... } DbcStruct::Signal

struct { ... } DbcStruct::Signal

struct { ... } DbcStruct::Signal

struct { ... } DbcStruct::Signal

uint8_t DbcStruct::SwVersionBugfix

uint8_t DbcStruct::SwVersionMajor

uint8_t DbcStruct::SwVersionMinor

uint8_t DbcStruct::TcpClientCount

TcpEnable_enum DbcStruct::TcpEnable

float DbcStruct::value

VpnEnable_enum DbcStruct::VpnEnable

WiFi_AP_Status_enum DbcStruct::WiFi_AP_Status

WiFiEnable_enum DbcStruct::WiFiEnable

The documentation for this struct was generated from the following file:

- include/**d2cc_lib.h**

File Documentation

Doxygen_Markdown.md File Reference

include/Bq76pl455.h File Reference

Header file for interfacing with the BQ76PL455A-Q1 battery management IC.

```
#include "Uart.h"
```

Macros

- #define **BQ_UART_PORT** UART_NUM_1
  UART port used for communication with the BQ76PL455A-Q1.

- #define **BQ_TX** 12
  GPIO pin number for UART TX (transmit) connected to the BQ76PL455A-Q1.

- #define **BQ_RX** 11
  GPIO pin number for UART RX (receive) connected to the BQ76PL455A-Q1.

Functions

- void **Bq_Init** ()
  Initializes the BQ76PL455A-Q1 interface.

- void **BQ_Test** ()
  Tests the communication with the BQ76PL455A-Q1.

- uint8_t * **CRC16** (uint8_t *data, uint8_t data_length)
  Calculates the CRC-16 for a given data buffer.

- void **BQ_Start** (void *args)
  Starts the BQ76PL455A-Q1 task.

- void **BQ_Uart_Init** (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)
  Initializes the UART interface for the BQ76PL455A-Q1.

- uint8_t **BQ_Uart_Transmit** (uint8_t uart_pin, uint8_t *data, uint8_t data_length)
  Transmits data to the BQ76PL455A-Q1 over UART.

Detailed Description

Header file for interfacing with the BQ76PL455A-Q1 battery management IC.

This file provides declarations for initializing, communicating, and managing the BQ76PL455A-Q1 IC using UART. It includes CRC-16 calculation and message structure details.

Created on: 18 Dec 2024

Author

hakimmc

Macro Definition Documentation

#define BQ_RX 11

GPIO pin number for UART RX (receive) connected to the BQ76PL455A-Q1.

#define BQ_TX 12

GPIO pin number for UART TX (transmit) connected to the BQ76PL455A-Q1.

#define BQ_UART_PORT UART_NUM_1

UART port used for communication with the BQ76PL455A-Q1.

Function Documentation

void Bq_Init ()

Initializes the BQ76PL455A-Q1 interface.

Sets up the UART interface and any necessary configurations for communication with the BQ76PL455A-Q1 device.

void BQ_Start (void * args)

Starts the BQ76PL455A-Q1 task.

A FreeRTOS task responsible for managing communication with the BQ76PL455A-Q1.

Parameters

args	Pointer to task-specific arguments (optional).

Starts the BQ76PL455A-Q1 task.

Continuously transmits a test message via UART at regular intervals.

Parameters

args	Task-specific arguments (optional).

void BQ_Test ()

Tests the communication with the BQ76PL455A-Q1.

Sends a test command or sequence to verify the functionality of the BQ76PL455A-Q1.

Tests the communication with the BQ76PL455A-Q1.

Sends a test message to read the device ID.

void BQ_Uart_Init (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)

Initializes the UART interface for the BQ76PL455A-Q1.

Configures the UART parameters for communicating with the BQ76PL455A-Q1, including pin assignments, baud rate, and RX buffer size.

Parameters

uart_pin	The UART port number to initialize.
baudrate	The desired baud rate for communication.
rx_buffsize	The size of the RX buffer.
TXD_PIN	The GPIO pin number for UART TX.
RXD_PIN	The GPIO pin number for UART RX.

Initializes the UART interface for the BQ76PL455A-Q1.

Parameters

uart_pin	UART port number.
baudrate	Baud rate for UART communication.
rx_buffsize	RX buffer size.
TXD_PIN	GPIO pin for UART TX.
RXD_PIN	GPIO pin for UART RX.

uint8_t BQ_Uart_Transmit (uint8_t uart_pin, uint8_t * data, uint8_t data_length)

Transmits data to the BQ76PL455A-Q1 over UART.

Sends a sequence of bytes to the BQ76PL455A-Q1 via UART.

Parameters

uart_pin	The UART port number to use for transmission.
data	Pointer to the data buffer to be transmitted.
data_length	The length of the data buffer in bytes.

Returns

Returns `0` on success or a non-zero value on failure.

Transmits data to the BQ76PL455A-Q1 over UART.

Parameters

uart_pin	UART port number.
data	Pointer to the data buffer to be transmitted.
data_length	Length of the data buffer in bytes.

Returns

1 on success, 0 on failure.

uint8_t * CRC16 (uint8_t * data, uint8_t data_length)

Calculates the CRC-16 for a given data buffer.

Computes the CRC-16 using the IBM standard with reversed MSB and LSB for compatibility with the BQ76PL455A-Q1 message format.

Parameters

data	Pointer to the data buffer for which the CRC needs to be calculated.
data_length	The length of the data buffer in bytes.

Returns

Pointer to a 2-byte array containing the CRC-16 result.

Calculates the CRC-16 for a given data buffer.

Parameters

data	Pointer to the data buffer.
data_length	Length of the data buffer in bytes.

Returns

Pointer to a 2-byte array containing the CRC checksum.

Bq76pl455.h

Go to the documentation of this file.

```
1
11
12 #ifndef BQ76PL455 H
13 #define BQ76PL455 H
14
15 #include "Uart.h"
16
21 #define BQ_UART_PORT    UART_NUM_1
22
27 #define BQ TX           12
28
33 #define BQ RX           11
34
41 void Bq_Init();
42
48 void BQ Test();
49
60 uint8_t* CRC16(uint8_t* data, uint8_t data_length);
61
69 void BQ Start(void *args);
70
83 void BQ Uart Init(uint8 t uart pin, int baudrate, uint32 t rx buffsize, uint8 t
TXD PIN, uint8 t RXD PIN);
84
95 uint8_t BQ_Uart_Transmit(uint8_t uart_pin, uint8_t *data, uint8_t data_length);
96
97 #endif /* BQ76PL455_H_ */
```

include/Can.h File Reference

CAN communication interface for ESP32.

```
#include "stdint.h"
#include <stdio.h>
#include <stdlib.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "esp_err.h"
#include "esp_log.h"
#include "driver/twai.h"
#include "d2cc_lib.h"
```

Macros

- #define **TX_GPIO_NUM** 5
  GPIO pin number for CAN TX.

- #define **RX_GPIO_NUM** 4
  GPIO pin number for CAN RX.

- #define **CAN_TAG** "CAN_MASTER"
  Tag for logging CAN-related messages.

- #define **Can_Main_ID** 0x100
  CAN ID for main messages.

- #define **Battery_Messages_ID** 0x101
  CAN ID for battery messages.

- #define **Can_Battery_Voltages_ID** 0x102
  CAN ID for battery voltage messages.

- #define **Can_Battery_Temperatures_ID** 0x103
  CAN ID for battery temperature messages.

- #define **CAN_DELAY** 250
  Delay in milliseconds for CAN tasks.

Functions

- uint8_t **Can_Init** (twai_general_config_t can_gpio_config, twai_timing_config_t can_time_config, twai_filter_config_t can_filter_config)
  Initializes the CAN interface.

- uint8_t **Can_Transmit** (twai_message_t message, uint8_t data[])
  Transmits a CAN message.

- void **CanReporter** (void *pvParameter)
  Task to report CAN messages.

Detailed Description

CAN communication interface for ESP32.

This header file provides declarations for initializing and managing CAN (Controller Area Network) communication on the ESP32 using the TWAI (Two-Wire Automotive Interface) driver.

Created on: 18 Dec 2024

Author

hakimmc

Macro Definition Documentation

#define Battery_Messages_ID 0x101

CAN ID for battery messages.

#define Can_Battery_Temperatures_ID 0x103

CAN ID for battery temperature messages.

#define Can_Battery_Voltages_ID 0x102

CAN ID for battery voltage messages.

#define CAN_DELAY 250

Delay in milliseconds for CAN tasks.

#define Can_Main_ID 0x100

CAN ID for main messages.

#define CAN_TAG "CAN_MASTER"

Tag for logging CAN-related messages.

#define RX_GPIO_NUM 4

GPIO pin number for CAN RX.

#define TX_GPIO_NUM 5

GPIO pin number for CAN TX.

Function Documentation

uint8_t Can_Init (twai_general_config_t can_gpio_config, twai_timing_config_t can_time_config, twai_filter_config_t can_filter_config)

Initializes the CAN interface.

Configures the CAN interface with the specified general, timing, and filter configurations.

Parameters

can_gpio_config	Configuration for CAN GPIO pins.
can_time_config	Timing configuration for CAN communication.
can_filter_config	Filter configuration for CAN messages.

Returns

0 on success or a non-zero value if initialization fails.

Initializes the CAN interface.

Parameters

can_gpio_config	General configuration for the CAN GPIO pins and mode.
can_time_config	Timing configuration for the CAN bus (e.g., bitrate).
can_filter_config	Filter configuration for message acceptance.

Returns

uint8_t Returns 1 if the driver is successfully initialized.

uint8_t Can_Transmit (twai_message_t message, uint8_t data[])

Transmits a CAN message.

Sends a CAN message with the specified data through the CAN interface.

Parameters

message	The TWAI message structure to transmit.
data	Pointer to the data array to be transmitted.

Returns

0 on success or a non-zero value if transmission fails.

Transmits a CAN message.

Parameters

message	The CAN message structure to transmit.
data	The data array to be sent with the message.

Returns

uint8_t Returns 1 if the transmission is successful.

void CanReporter (void * pvParameter)

Task to report CAN messages.

A FreeRTOS task that handles CAN communication and reporting.

Parameters

pvParameter	Pointer to task-specific parameters (optional).

Task to report CAN messages.

This function initializes the CAN driver and repeatedly transmits preconfigured CAN messages, using data from the **`DbcStruct`** .

Parameters

pvParameter	A pointer to any parameters passed to the task (unused).

Can.h

Go to the documentation of this file.

```
1
11
12 #ifndef INCLUDE CAN H
13 #define INCLUDE CAN H
14
15 #include "stdint.h"
16 #include <stdio.h>
17 #include <stdlib.h>
18 #include "freertos/FreeRTOS.h"
19 #include "freertos/task.h"
20 #include "freertos/queue.h"
21 #include "freertos/semphr.h"
22 #include "esp_err.h"
23 #include "esp_log.h"
24 #include "driver/twai.h"
25 #include "d2cc lib.h"
26
31 #define TX_GPIO_NUM             5
32
37 #define RX GPIO NUM             4
38
43 #define CAN TAG "CAN MASTER"
44
49 #define Can_Main_ID                             0x100
50
55 #define Battery Messages ID                     0x101
56
61 #define Can Battery Voltages ID                 0x102
62
67 #define Can_Battery_Temperatures_ID             0x103
68
73 #define CAN DELAY                               250 // ms
74
85 uint8 t Can Init(twai general config t can gpio config, twai timing config t
can time config, twai filter config t can filter config);
86
96 uint8_t Can_Transmit(twai_message_t message, uint8_t data[]);
97
105 void CanReporter(void* pvParameter);
106
107 #endif /* INCLUDE_CAN_H_ */
```

include/d2cc_lib.h File Reference

```
#include <stdint.h>
#include <stdbool.h>
```

Classes

struct DbcStruct**Enumerations**

- enum **BatteryChemistry_enum** { **BatteryChemistry_LFP**, **BatteryChemistry_NMC** }
- enum **CanBusEnable_enum** { **CanBusEnable_Enable**, **CanBusEnable_Disable** }
- enum **WiFiEnable_enum** { **WiFiEnable_Enable**, **WiFiEnable_Disable** }
- enum **TcpEnable_enum** { **TcpEnable_Offline**, **TcpEnable_Online**, **TcpEnable_Connected** }
- enum **VpnEnable_enum** { **VpnEnable_Enable**, **VpnEnable_Disable** }
- enum **WiFi_AP_Status_enum** { **WiFi_AP_Status_Enable**, **WiFi_AP_Status_Disable** }

Functions

- void **D2cc_Lib_Init** (**DbcStruct** *st)
- void **ReadParse** (uint8_t *rx_data, uint32_t id, **DbcStruct** *st)
- void **CreateTable_Battery_Messages** (**DbcStruct** *dbc)
- void **CreateTable_Battery_Temperatures** (**DbcStruct** *dbc)
- void **CreateTable_Battery_Voltages** (**DbcStruct** *dbc)
- void **CreateTable_Can_Main** (**DbcStruct** *dbc)

Enumeration Type Documentation

enum BatteryChemistry_enum

Enumerator:

BatteryChemistry_LFP	
BatteryChemistry_NMC	

enum CanBusEnable_enum

Enumerator:

CanBusEnable_Enable	
CanBusEnable_Disable	

enum TcpEnable_enum

Enumerator:

TcpEnable_Offline	
TcpEnable_Online	
TcpEnable_Conn	

ected	

enum VpnEnable_enum

Enumerator:

VpnEnable_Enable	
VpnEnable_Disable	

enum WiFi_AP_Status_enum

Enumerator:

WiFi_AP_Status_Enable	
WiFi_AP_Status_Disable	

enum WiFiEnable_enum

Enumerator:

WiFiEnable_Enable	
WiFiEnable_Disable	

Function Documentation

void CreateTable_Battery_Messages (DbcStruct * dbc)

void CreateTable_Battery_Temperatures (DbcStruct * dbc)

void CreateTable_Battery_Voltages (DbcStruct * dbc)

void CreateTable_Can_Main (DbcStruct * dbc)

void D2cc_Lib_Init (DbcStruct * st)

void ReadParse (uint8_t * rx_data, uint32_t id, DbcStruct * st)

d2cc_lib.h

Go to the documentation of this file.

```
1 /*
2  *  d2cc_lib.h
3  *
4  *  Created on: 5.01.2025
5  *  Author: hakimmc
6  *
7  *  https://www.linkedin.com/in/abdulhakim-calgin/
8  *
9  */
10
11 #ifndef LIB
12 #define LIB
13
14 #include <stdint.h>
15 #include <stdbool.h>
16
17
18 typedef enum{
19     BatteryChemistry_LFP,   BatteryChemistry_NMC
20 }BatteryChemistry enum;
21
22
23 typedef enum{
24     CanBusEnable_Enable, CanBusEnable_Disable
25 }CanBusEnable_enum;
26
27
28 typedef enum{
29     WiFiEnable Enable, WiFiEnable Disable
30 }WiFiEnable_enum;
31
32
33 typedef enum{
34     TcpEnable Offline, TcpEnable Online,   TcpEnable Connected
35 }TcpEnable enum;
36
37
38 typedef enum{
39     VpnEnable Enable,   VpnEnable Disable
40 }VpnEnable enum;
41
42
43 typedef enum{
44     WiFi AP Status Enable, WiFi AP Status Disable
45 }WiFi AP Status enum;
46
47 typedef struct{
48 /* Battery_Messages Line Start */
49     struct{
50         uint32 t ID;
51         uint8 t DLC;
52         union{
53             struct{
54             }Signal;
55             uint8 t Data[8];
56         };
57     }Battery Messages;
58
59 /* Battery_Messages Line End */
60
61 /* Battery Temperatures Line Start */
62     struct{
63         uint32 t ID;
64         uint8 t DLC;
65         union{
66             struct{
67             }Signal;
68             uint8 t Data[8];
69         };
70     }Battery_Temperatures;
```

```
71
72 /* Battery Temperatures Line End */
73
74 /* Battery Voltages Line Start */
75     struct{
76         uint32_t ID;
77         uint8 t DLC;
78         union{
79             struct{
80             }Signal;
81             uint8_t Data[8];
82         };
83     }Battery Voltages;
84
85 /* Battery Voltages Line End */
86
87 /* Can_Main Line Start */
88     struct{
89         uint32 t ID;
90         uint8 t DLC;
91         union{
92             struct{
93                 uint8_t AliveCounter:7; //7 bit
94                 BatteryChemistry enum BatteryChemistry:1; //1 bit
95                 uint8 t BatteryBalance MaxVoltage:8; //8 bit
96                 uint8 t BatteryBalance MinVoltage:8; //8 bit
97                 uint8 t BatteryBalance MaxTemp:8; //8 bit
98                 uint8_t BatteryBalance_MinTemp:8; //8 bit
99                 uint8_t SwVersionMajor:2; //2 bit
100                 uint8 t SwVersionMinor:2; //2 bit
101                 uint8 t SwVersionBugfix:4; //4 bit
102                 CanBusEnable enum CanBusEnable:1; //1 bit
103                 WiFiEnable enum WiFiEnable:1; //1 bit
104                 TcpEnable_enum TcpEnable:2; //2 bit
105                 VpnEnable_enum VpnEnable:1; //1 bit
106                 WiFi AP Status enum WiFi AP Status:2; //2 bit
107                 uint8 t TcpClientCount:8; //8 bit
108             }Signal;
109             uint8_t Data[8];
110         };
111         struct{
112             struct{
113                 float factor;
114                 int offset;
115                 float value;
116             }Phys_Value;
117         }BatteryBalance_MaxVoltage;
118         struct{
119             struct{
120                 float factor;
121                 int offset;
122                 float value;
123             }Phys Value;
124         }BatteryBalance MinVoltage;
125         struct{
126             struct{
127                 float factor;
128                 int offset;
129                 float value;
130             }Phys Value;
131         }BatteryBalance MinTemp;
132     }Can_Main;
133
134 /* Can_Main Line End */
135
136 }DbcStruct;
137 /*     USER CODE FUNCTION BLOCK START       */
138
139 void D2cc_Lib_Init(DbcStruct *st); //Init Function (Must Be Run)
140
141 void ReadParse(uint8 t* rx data, uint32 t id, DbcStruct *st); //Can Read & Parse
Function
142
143 void CreateTable_Battery_Messages(DbcStruct *dbc);
144
145 void CreateTable Battery Temperatures(DbcStruct *dbc);
146
```

```
147 void CreateTable Battery Voltages(DbcStruct *dbc);
148
149 void CreateTable Can Main(DbcStruct *dbc);
150
151 /*     USER CODE FUNCTION BLOCK STOP        */
152
153 #endif
```

include/LedTaskInit.h File Reference

LED initialization and control tasks for ESP32.

```
#include "stdint.h"
```

Functions

- void **gpio_init** (uint8_t gpio_pin)
  Initializes a GPIO pin for LED control.

- void **led_init** (void *pvParameter)
  Task to initialize and control LED behavior.

Detailed Description

LED initialization and control tasks for ESP32.

This header file provides declarations for initializing GPIO pins and creating LED-related tasks for the ESP32.

Created on: 18 Dec 2024

Author

hakimmc

Function Documentation

void gpio_init (uint8_t gpio_pin)

Initializes a GPIO pin for LED control.

Configures the specified GPIO pin as an output to control an LED.

Parameters

gpio_pin	The GPIO pin number to initialize.

Initializes a GPIO pin for LED control.

This function configures the given GPIO pin for output mode with no pull-up or pull-down resistors.

Parameters

gpio_pin	The GPIO pin number to initialize.

< No interrupt triggered.

< Set GPIO mode to output.

< Pin mask for the specified GPIO pin.

< Disable pull-down resistor.

< Disable pull-up resistor.

void led_init (void * pvParameter)

Task to initialize and control LED behavior.

This function serves as an entry point for a FreeRTOS task, handling LED initialization and operation based on the provided parameters.

Parameters

pvParameter	Pointer to task parameters (optional).

Task to initialize and control LED behavior.

This task initializes the LED GPIO pin and continuously toggles the LED state with a 500 ms delay.

Parameters

pvParameter	A pointer to task-specific parameters (unused).

< Initialize the LED GPIO pin.

< Turn the LED on.

< Delay for 500 ms.

< Turn the LED off.

< Delay for 500 ms.

LedTaskInit.h

Go to the documentation of this file.

```
1
11
12 #ifndef INCLUDE_LEDTASKINIT_H
13 #define INCLUDE_LEDTASKINIT_H
14
15 #include "stdint.h"
16
24 void gpio_init(uint8_t gpio_pin);
25
34 void led_init(void* pvParameter);
35
36 #endif /* INCLUDE_LEDTASKINIT_H_ */
```

include/TcpServer.h File Reference

TCP server implementation for ESP32.

```
#include <lwip/sockets.h>
#include <esp_log.h>
#include <string.h>
#include <errno.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "Wifi.h"
#include "esp_ping.h"
#include "ping/ping_sock.h"
#include "nvs_flash.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "d2cc_lib.h"
```

Macros

- #define **WIFI_CONNECTED_BIT** BIT0
  Uncomment the following line to enable Wi-Fi connection functionality.

- #define **WIFI_FAIL_BIT** BIT1
  Event bit indicating a Wi-Fi connection failure.

- #define **TCP_TAG** "TCP_SERVER"
  Tag for logging TCP server messages.

- #define **SERVER_PORT** 5166
  Port number for the TCP server.

Functions

- void **Handle_Client** (void *args)
  Handles client connections to the TCP server.

- void **Create_Server** (void *pvParameter)
  Creates and starts the TCP server.

- int **receive_data** (int sock, char *buffer, size_t size, uint8_t *timeout_counter, uint8_t max_timeout)
  Receives data from a connected client.

- void **Tcp_Init** ()
  Initializes the TCP server.

Detailed Description

TCP server implementation for ESP32.

This header file provides declarations for functions and macros used in implementing a TCP server on the ESP32. It supports handling client connections and data transfer over Wi-Fi.

Created on: 18 Dec 2024

Author

hakimmc

Macro Definition Documentation

#define SERVER_PORT 5166

Port number for the TCP server.

#define TCP_TAG "TCP_SERVER"

Tag for logging TCP server messages.

#define WIFI_CONNECTED_BIT BIT0

Uncomment the following line to enable Wi-Fi connection functionality.

Event bit indicating a successful Wi-Fi connection.

#define WIFI_FAIL_BIT BIT1

Event bit indicating a Wi-Fi connection failure.

Function Documentation

void Create_Server (void * pvParameter)

Creates and starts the TCP server.

Sets up the TCP server to listen for incoming client connections and handle them using the specified parameters.

Parameters

pvParameter	Pointer to additional parameters for server setup (optional).

Creates and starts the TCP server.

This function sets up a TCP socket, binds it to a specified address and port, and listens for incoming connections. For each connection, a new task is created to handle the client.

Parameters

pvParameter	Unused parameter for FreeRTOS task.

void Handle_Client (void * args)

Handles client connections to the TCP server.

This function processes communication with a connected client, handling data transmission and reception as required.

Parameters

args	Pointer to additional arguments (optional).

Handles client connections to the TCP server.

This function receives and processes commands from the client, authenticates the user, and manages communication based on different states. It handles login, data requests, and exit commands.

Parameters

args	Socket descriptor for the client.

int receive_data (int sock, char * buffer, size_t size, uint8_t * timeout_counter, uint8_t max_timeout)

Receives data from a connected client.

Reads data from a socket connection into the provided buffer, with support for timeout and retry mechanisms.

Parameters

sock	The socket descriptor for the client connection.
buffer	Pointer to the buffer where the received data will be stored.
size	The maximum number of bytes to read.
timeout_counter	Pointer to a counter tracking timeout occurrences.
max_timeout	The maximum number of allowable timeouts before returning an error.

Returns

The number of bytes received, or a negative value if an error occurs.

Receives data from a connected client.

This function receives data from the client and handles timeouts. If no data is received within a specified timeout period, it returns a timeout error.

Parameters

sock	Socket descriptor for the client.
buffer	Buffer to store received data.
size	Size of the buffer.
timeout_counter	Counter to track consecutive timeouts.
max_timeout	Maximum allowed consecutive timeouts before returning an error.

Returns

Length of received data, -1 if timeout occurs, or -2 if maximum timeouts reached.

void Tcp_Init ()

Initializes the TCP server.

Configures the necessary components and prepares the ESP32 for running a TCP server.

Initializes the TCP server.

< Initialize Wi-Fi in access point and station mode.

< Start the TCP server task.

TcpServer.h

Go to the documentation of this file.

```
1
12
13 #ifndef INCLUDE TCPSERVER H
14 #define INCLUDE TCPSERVER H
15
19 //#define WIFI CONNECT
20
21 #include <lwip/sockets.h>
22 #include <esp log.h>
23 #include <string.h>
24 #include <errno.h>
25 #include "freertos/FreeRTOS.h"
26 #include "freertos/task.h"
27 #include "esp_system.h"
28 #include "Wifi.h"
29 #include "string.h"
30 #include "esp ping.h"
31 #include "ping/ping_sock.h"
32 #include "nvs_flash.h"
33 #include "freertos/event groups.h"
34 #include "esp wifi.h"
35 #include "esp event.h"
36 #include "esp log.h"
37 #include "d2cc_lib.h"
38
43 #define WIFI CONNECTED BIT BIT0
44
49 #define WIFI FAIL BIT      BIT1
50
55 #define TCP_TAG         "TCP_SERVER"
56
61 #define SERVER PORT     5166
62
71 void Handle Client(void* args);
72
81 void Create_Server(void* pvParameter);
82
96 int receive data(int sock, char* buffer, size t size, uint8 t* timeout counter, uint8 t
max timeout);
97
103 void Tcp Init();
104
105 #endif /* INCLUDE_TCPSERVER_H_ */
```

include/Uart.h File Reference

UART driver interface for ESP32.

```
#include "stdint.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include <stdio.h>
#include <stdlib.h>
```

Functions

- void **Uart_Init** (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)
  Initializes the UART interface.

- uint8_t **Uart_Transmit** (uint8_t uart_pin, uint8_t *data, uint8_t data_length)
  Transmits data over UART.

- int **Uart_Receive** (uint8_t uart_pin, uint8_t *data, uint8_t data_length, uint32_t timeout)
  Receives data from UART.

- uint8_t **IsTimeout** (uint32_t max_reach_time)
  Checks for a timeout condition.

Detailed Description

UART driver interface for ESP32.

This header file provides the function declarations for initializing and using UART communication on the ESP32.

Created on: 18 Dec 2024

Author

hakimmc

Function Documentation

uint8_t IsTimeout (uint32_t max_reach_time)

Checks for a timeout condition.

Determines if a given time has exceeded the specified maximum time limit.

Parameters

max_reach_time	The maximum allowable time in milliseconds.

Returns

Returns `1` if the timeout condition is met, or `0` otherwise.

Checks for a timeout condition.

This function checks if the specified maximum time has passed since the start of the function call.

Parameters

max_reach_time	Maximum time in ticks to wait before timing out.

Returns

0 if timeout occurred, 1 if still within the allowed time.

< Get current tick count.

< Wait for the timeout period to pass.

< Timeout reached.

< Timeout not reached.

void Uart_Init (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)

Initializes the UART interface.

Configures the UART interface with the specified parameters, including pin assignments, baud rate, and RX buffer size.

Parameters

uart_pin	The UART port number to initialize (e.g., UART_NUM_0, UART_NUM_1).
baudrate	The desired baud rate for communication.
rx_buffsize	The size of the RX buffer.
TXD_PIN	The GPIO pin number assigned for UART TX (transmit).
RXD_PIN	The GPIO pin number assigned for UART RX (receive).

Initializes the UART interface.

This function configures the UART with the specified baud rate, data bits, stop bits, and other settings. It also assigns pins for transmission and reception and installs the UART driver.

Parameters

uart_pin	UART port to configure (typically 0, 1, or 2).
baudrate	The baud rate for the UART communication.
rx_buffsize	The size of the receive buffer.
TXD_PIN	The GPIO pin for UART transmission (TX).
RXD_PIN	The GPIO pin for UART reception (RX).

< Configures the UART with the given settings.

< Assigns pins to UART.

< Installs the UART driver with specified buffer size.

int Uart_Receive (uint8_t uart_pin, uint8_t * data, uint8_t data_length, uint32_t timeout)

Receives data from UART.

Reads data from the specified UART port into a buffer, with an optional timeout.

Parameters

uart_pin	The UART port number to use for reception.
data	Pointer to the buffer where received data will be stored.
data_length	The maximum number of bytes to read.
timeout	The maximum time to wait for data in milliseconds.

Returns

The number of bytes successfully received, or a negative value on error.

Receives data from UART.

This function receives data from the UART port and stores it in the provided buffer.

Parameters

uart_pin	UART port to use for reception (typically 0, 1, or 2).
data	Pointer to the buffer to store received data.
data_length	Length of the data buffer.
timeout	Timeout for receiving data in milliseconds.

Returns

The number of bytes received or an error code.

< Reads data from UART into buffer.

uint8_t Uart_Transmit (uint8_t uart_pin, uint8_t * data, uint8_t data_length)

Transmits data over UART.

Sends a sequence of bytes through the specified UART port.

Parameters

uart_pin	The UART port number to use for transmission.
data	Pointer to the data buffer containing the bytes to transmit.
data_length	The number of bytes to transmit.

Returns

Returns `0` on success or a non-zero value if the transmission fails.

Transmits data over UART.

This function sends a specified amount of data through the UART transmission port.

Parameters

uart_pin	UART port to use for transmission (typically 0, 1, or 2).
data	Pointer to the data to be transmitted.
data_length	Length of the data to be transmitted.

Returns

1 if transmission is successful, 0 if failed.

< Writes data to UART.

< Transmission failure.

< Transmission success.

Uart.h

Go to the documentation of this file.

```
1
11
12 #ifndef INCLUDE UART H
13 #define INCLUDE UART H
14
15 #include "stdint.h"
16 #include "driver/uart.h"
17 #include "driver/gpio.h"
18 #include <stdio.h>
19 #include <stdlib.h>
20
33 void Uart Init(uint8 t uart pin, int baudrate, uint32 t rx buffsize, uint8 t TXD PIN,
uint8_t RXD_PIN);
34
45 uint8 t Uart Transmit(uint8 t uart pin, uint8 t *data, uint8 t data length);
46
58 int Uart Receive(uint8 t uart pin, uint8 t* data, uint8 t data length, uint32 t
timeout);
59
68 uint8 t IsTimeout(uint32 t max reach time);
69
70 #endif // INCLUDE_UART_H_
```

include/Wifi.h File Reference

Wi-Fi configuration for ESP32 in AP+STA mode.

```
#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"
#include "esp_wifi.h"
#include "freertos/task.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_netif.h"
#include "string.h"
#include <inttypes.h>
#include <time.h>
#include <sys/time.h>
#include <esp_system.h>
#include <lwip/netdb.h>
#include <ping/ping_sock.h>
```

Macros

- #define **WIFI_CONNECTED_BIT** BIT0
  Event bit indicating a successful Wi-Fi connection.

- #define **WIFI_FAIL_BIT** BIT1
  Event bit indicating a Wi-Fi connection failure.

- #define **WIFI_CONNECT**
  Define this macro to enable Wi-Fi station connection. Comment it out to disable the Wi-Fi station functionality.

- #define **AP_SSID** "Ottomotive_BMS"
  SSID for the Access Point.

- #define **AP_PASS** "Ottomotive22*"
  Password for the Access Point.

- #define **MAX_STA_CONN** 5
  Maximum number of stations that can connect to the Access Point.

- #define **MAX_RETRY** 5
  Maximum number of retry attempts for Wi-Fi connection.

- #define **WIFI_SSID** "Poyrazwifi_Calgin"
  SSID for the Wi-Fi station connection.

- #define **WIFI_PASS** "Ah487602"
  Password for the Wi-Fi station connection.

- #define **WIFI_TIMEOUT_MS** 10000

Timeout for Wi-Fi connection in milliseconds.

- #define **ESP32_AP_CHANNEL** 3
  Channel for the Access Point.

- #define **ESP32_MAX_CONN** 3
  Maximum number of simultaneous connections to the Access Point.

Functions

- static void **event_handler** (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
  Handles Wi-Fi events.

- void **wifi_init_ap_sta** (void)
  Initializes the Wi-Fi in AP+STA mode.

Detailed Description

Wi-Fi configuration for ESP32 in AP+STA mode.

This header file provides the necessary definitions and function declarations for configuring the ESP32 to operate in both Access Point (AP) and Station (STA) modes.

Created on: 18 Dec 2024

Author

hakimmc

Macro Definition Documentation

#define AP_PASS "Ottomotive22*"

Password for the Access Point.

#define AP_SSID "Ottomotive_BMS"

SSID for the Access Point.

#define ESP32_AP_CHANNEL 3

Channel for the Access Point.

#define ESP32_MAX_CONN 3

Maximum number of simultaneous connections to the Access Point.

#define MAX_RETRY 5

Maximum number of retry attempts for Wi-Fi connection.

#define MAX_STA_CONN 5

Maximum number of stations that can connect to the Access Point.

#define WIFI_CONNECT

Define this macro to enable Wi-Fi station connection. Comment it out to disable the Wi-Fi station functionality.

#define WIFI_CONNECTED_BIT BIT0

Event bit indicating a successful Wi-Fi connection.

#define WIFI_FAIL_BIT BIT1

Event bit indicating a Wi-Fi connection failure.

#define WIFI_PASS "Ah487602"

Password for the Wi-Fi station connection.

#define WIFI_SSID "Poyrazwifi_Calgin"

SSID for the Wi-Fi station connection.

#define WIFI_TIMEOUT_MS 10000

Timeout for Wi-Fi connection in milliseconds.

Function Documentation

static void event_handler (void * arg, esp_event_base_t event_base, int32_t event_id, void * event_data) `[static]`

Handles Wi-Fi events.

This function is called to process various Wi-Fi events, such as connection establishment and disconnection.

Parameters

arg	User-provided argument (optional).
event_base	Event base that identifies the event type.
event_id	Event ID specifying the exact event.
event_data	Additional data associated with the event (optional).

void wifi_init_ap_sta (void)

Initializes the Wi-Fi in AP+STA mode.

This function configures the ESP32 to operate in both Access Point (AP) and Station (STA) modes. The configuration is based on the defined macros.

Initializes the Wi-Fi in AP+STA mode.

This function initializes the Wi-Fi stack, configures the Wi-Fi interfaces, and starts the Wi-Fi service. It also registers event handlers for Wi-Fi and IP events, and connects to a Wi-Fi network in STA mode, if configured.

Note

This function configures the ESP32 in dual mode (AP and STA) and connects to a Wi-Fi network if `WIFI_CONNECT` is defined.

< Initialize network interface.

< Create the default event loop.

< Create the default Wi-Fi AP (Access Point) interface.

< Default Wi-Fi configuration.

< Initialize Wi-Fi driver with the default configuration.

< Register Wi-Fi event handler.

< Register IP event handler.

< AP SSID.

< AP password.

< Length of SSID.

< AP channel.

< Max connections to the AP.

< WPA/WPA2 PSK authentication.

< Set Wi-Fi mode to AP only.

< Set AP configuration.

< Start the Wi-Fi driver.

< Log completion of Wi-Fi initialization.

Wifi.h

Go to the documentation of this file.

```
1
12
13 #ifndef WIFI AP STA H
14 #define WIFI AP STA H
15
16 #include "freertos/FreeRTOS.h"
17 #include "freertos/event groups.h"
18 #include "esp_wifi.h"
19 #include "freertos/task.h"
20 #include "esp event.h"
21 #include "esp log.h"
22 #include "nvs flash.h"
23 #include "esp_netif.h"
24 #include "string.h"
25 #include <string.h>
26 #include <inttypes.h>
27 #include <time.h>
28 #include <sys/time.h>
29 #include <esp_system.h>
30 #include <lwip/netdb.h>
31 #include <ping/ping sock.h>
32
37 #define WIFI CONNECTED BIT BIT0
38
43 #define WIFI_FAIL_BIT      BIT1
44
49 #define WIFI CONNECT
50
55 #define AP SSID         "Ottomotive BMS"
56
61 #define AP_PASS         "Ottomotive22*"
62
67 #define MAX STA CONN   5
68
73 #define MAX RETRY      5
74
75 #ifdef WIFI_CONNECT
80     #define WIFI SSID   "Poyrazwifi Calgin"
81
86     #define WIFI PASS   "Ah487602"
87
92     #define WIFI_TIMEOUT_MS 10000
93
98     #define ESP32 AP CHANNEL        3
99
104     #define ESP32 MAX CONN          3
105 #endif
106
118 static void event handler(void *arg, esp event base t event base, int32 t event id,
void *event data);
119
126 void wifi init ap sta(void);
127
128 #endif // WIFI_AP_STA_H
```

src/Bq76pl455.c File Reference

Implementation file for interfacing with the BQ76PL455A-Q1 battery management IC.
`#include "Bq76pl455.h"`

Functions

- void **BQ_Init** ()
  Initialize the BQ76PL455A-Q1 communication interface.

- void **BQ_Test** ()
  Test communication with the BQ76PL455A-Q1 device.

- void **BQ_Start** (void *args)
  Starts the main task for BQ76PL455A-Q1.

- uint8_t * **CRC16** (uint8_t *data, uint8_t data_length)
  Calculate the CRC-16 checksum for a given data buffer.

- void **BQ_Uart_Init** (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)
  Initialize UART for communication with the BQ76PL455A-Q1.

- uint8_t **BQ_Uart_Transmit** (uint8_t uart_pin, uint8_t *data, uint8_t data_length)
  Transmit data via UART to the BQ76PL455A-Q1.

Variables

- const uint16_t **crc16_table_bq** [256]

Detailed Description

Implementation file for interfacing with the BQ76PL455A-Q1 battery management IC.

Provides functionality to initialize, communicate, and calculate CRC for the BQ76PL455A-Q1 IC.

Created on: 18 Dec 2024

Author

Function Documentation

void BQ_Init ()

Initialize the BQ76PL455A-Q1 communication interface.

Configures the UART for the BQ76PL455A-Q1.

void BQ_Start (void * args)

Starts the main task for BQ76PL455A-Q1.

Starts the BQ76PL455A-Q1 task.

Continuously transmits a test message via UART at regular intervals.

Parameters

args	Task-specific arguments (optional).

void BQ_Test ()

Test communication with the BQ76PL455A-Q1 device.

Tests the communication with the BQ76PL455A-Q1.

Sends a test message to read the device ID.

void BQ_Uart_Init (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)

Initialize UART for communication with the BQ76PL455A-Q1.

Initializes the UART interface for the BQ76PL455A-Q1.

Parameters

uart_pin	UART port number.
baudrate	Baud rate for UART communication.
rx_buffsize	RX buffer size.
TXD_PIN	GPIO pin for UART TX.
RXD_PIN	GPIO pin for UART RX.

uint8_t BQ_Uart_Transmit (uint8_t uart_pin, uint8_t * data, uint8_t data_length)

Transmit data via UART to the BQ76PL455A-Q1.

Transmits data to the BQ76PL455A-Q1 over UART.

Parameters

uart_pin	UART port number.
data	Pointer to the data buffer to be transmitted.
data_length	Length of the data buffer in bytes.

Returns

1 on success, 0 on failure.

uint8_t * CRC16 (uint8_t * data, uint8_t data_length)

Calculate the CRC-16 checksum for a given data buffer.

Calculates the CRC-16 for a given data buffer.

Parameters

data	Pointer to the data buffer.
data_length	Length of the data buffer in bytes.

Returns

Pointer to a 2-byte array containing the CRC checksum.

Variable Documentation

const uint16_t crc16_table_bq[256]

Initial value:

```
= {



}
```

src/Can.c File Reference

CAN driver implementation for the ESP32 using the TWAI driver.

```
#include "Can.h"
#include "d2cc_lib.h"
```

Functions

- uint8_t **Can_Init** (twai_general_config_t can_gpio_config, twai_timing_config_t can_time_config, twai_filter_config_t can_filter_config)
  Initializes the CAN (TWAI) driver with the specified configurations.

- uint8_t **Can_Transmit** (twai_message_t message, uint8_t data[])
  Transmits a CAN message with the specified data.

- void **CanReporter** (void *pvParameter)
  Task function to periodically report CAN messages.

Variables

- **DbcStruct maindbc_struct**
  Structure to hold CAN message data.

- twai_message_t **Can_Main** = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = **Can_Main_ID**, .data_length_code = 8, .data = {0},}
- twai_message_t **Battery_Messages** = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = **Battery_Messages_ID**, .data_length_code = 8, .data = {0},}
- twai_message_t **Battery_Voltages** = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = **Can_Battery_Voltages_ID**, .data_length_code = 8, .data = {0},}
- twai_message_t **Battery_Temperatures** = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = **Can_Battery_Temperatures_ID**, .data_length_code = 8, .data = {0},}
- uint8_t **Data_Of_Can_Main** [8] = {0, 1, 2, 3, 4, 5, 6, 7}
- uint8_t **Data_Of_Battery_Messages** [8] = {8, 9, 10, 11, 12, 13, 14, 15}
- uint8_t **Data_Of_Battery_Voltages** [8] = {16, 17, 22, 32, 42, 52, 62, 72}
- uint8_t **Data_Of_Battery_Temperatures** [8] = {35, 13, 23, 33, 43, 53, 63, 73}

Detailed Description

CAN driver implementation for the ESP32 using the TWAI driver.

Date

18 December 2024

Author

hakimmc

Function Documentation

uint8_t Can_Init (twai_general_config_t can_gpio_config, twai_timing_config_t can_time_config, twai_filter_config_t can_filter_config)

Initializes the CAN (TWAI) driver with the specified configurations.

Initializes the CAN interface.

Parameters

can_gpio_config	General configuration for the CAN GPIO pins and mode.
can_time_config	Timing configuration for the CAN bus (e.g., bitrate).
can_filter_config	Filter configuration for message acceptance.

Returns

uint8_t Returns 1 if the driver is successfully initialized.

uint8_t Can_Transmit (twai_message_t message, uint8_t data[])

Transmits a CAN message with the specified data.

Transmits a CAN message.

Parameters

message	The CAN message structure to transmit.
data	The data array to be sent with the message.

Returns

uint8_t Returns 1 if the transmission is successful.

void CanReporter (void * pvParameter)

Task function to periodically report CAN messages.

Task to report CAN messages.

This function initializes the CAN driver and repeatedly transmits preconfigured CAN messages, using data from the **`DbcStruct`** .

Parameters

pvParameter	A pointer to any parameters passed to the task (unused).

Variable Documentation

twai_message_t Battery_Messages = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = Battery_Messages_ID, .data_length_code = 8, .data = {0},}

twai_message_t Battery_Temperatures = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = Can_Battery_Temperatures_ID, .data_length_code = 8, .data = {0},}

twai_message_t Battery_Voltages = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = Can_Battery_Voltages_ID, .data_length_code = 8, .data = {0},}

twai_message_t Can_Main = {.extd = 0, .rtr = 0, .ss = 0, .self = 0, .dlc_non_comp = 0, .identifier = Can_Main_ID, .data_length_code = 8, .data = {0},}

uint8_t Data_Of_Battery_Messages[8] = {8, 9, 10, 11, 12, 13, 14, 15}

uint8_t Data_Of_Battery_Temperatures[8] = {35, 13, 23, 33, 43, 53, 63, 73}

uint8_t Data_Of_Battery_Voltages[8] = {16, 17, 22, 32, 42, 52, 62, 72}

uint8_t Data_Of_Can_Main[8] = {0, 1, 2, 3, 4, 5, 6, 7}

DbcStruct maindbc_struct `[extern]`

Structure to hold CAN message data.

src/d2cc_lib.c File Reference

```
#include "d2cc_lib.h"
```

Functions

- void **D2cc_Lib_Init** (**DbcStruct** *dbc)
- void **ReadParse** (uint8_t *rx_data, uint32_t id, **DbcStruct** *dbc)
- void **CreateTable_Battery_Messages** (**DbcStruct** *dbc)
- void **CreateTable_Battery_Temperatures** (**DbcStruct** *dbc)
- void **CreateTable_Battery_Voltages** (**DbcStruct** *dbc)
- void **CreateTable_Can_Main** (**DbcStruct** *dbc)

Function Documentation

void CreateTable_Battery_Messages (DbcStruct * dbc)

void CreateTable_Battery_Temperatures (DbcStruct * dbc)

void CreateTable_Battery_Voltages (DbcStruct * dbc)

void CreateTable_Can_Main (DbcStruct * dbc)

void D2cc_Lib_Init (DbcStruct * dbc)

void ReadParse (uint8_t * rx_data, uint32_t id, DbcStruct * dbc)

src/LedTaskInit.c File Reference

Implementation of LED blinking task using ESP32 GPIO.

```
#include "LedTaskInit.h"
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
```

Macros

- #define **LED_PIN** 13
  GPIO pin used for the LED.

Functions

- void **gpio_init** (uint8_t gpio_pin)
  Initializes the specified GPIO pin as an output.

- void **led_init** (void *pvParameter)
  LED blinking task.

Detailed Description

Implementation of LED blinking task using ESP32 GPIO.

Date

18 December 2024

Author

hakimmc

Macro Definition Documentation

#define LED_PIN 13

GPIO pin used for the LED.

Function Documentation

void gpio_init (uint8_t gpio_pin)

Initializes the specified GPIO pin as an output.

Initializes a GPIO pin for LED control.

This function configures the given GPIO pin for output mode with no pull-up or pull-down resistors.

Parameters

gpio_pin	The GPIO pin number to initialize.

< No interrupt triggered.

< Set GPIO mode to output.

< Pin mask for the specified GPIO pin.

< Disable pull-down resistor.

< Disable pull-up resistor.

void led_init (void * pvParameter)

LED blinking task.

Task to initialize and control LED behavior.

This task initializes the LED GPIO pin and continuously toggles the LED state with a 500 ms delay.

Parameters

pvParameter	A pointer to task-specific parameters (unused).

< Initialize the LED GPIO pin.

< Turn the LED on.

< Delay for 500 ms.

< Turn the LED off.

< Delay for 500 ms.

src/main.c File Reference

Main application entry point for ESP32, initializing various tasks.

```
#include "LedTaskInit.h"
#include "Can.h"
#include "Wifi.h"
#include "TcpServer.h"
#include "d2cc_lib.h"
#include "Bq76pl455.h"
```

Functions

- void **app_main** ()
  Application entry point.

Variables

- **DbcStruct maindbc_struct**
  Structure to hold CAN message data.

Detailed Description

Main application entry point for ESP32, initializing various tasks.

Date

18 December 2024

Author

hakimmc

Function Documentation

void app_main ()

Application entry point.

This function initializes non-volatile storage, sets up the TCP server, and creates tasks for CAN communication, LED toggling, and BQ76PL455 functionality.

< Print "helloworld" for testing.

Variable Documentation

DbcStruct maindbc_struct

Structure to hold CAN message data.

src/README.md File Reference

src/TcpServer.c File Reference

TCP server implementation to handle incoming client connections, authenticate, and exchange data.

```
#include "TcpServer.h"
#include "Wifi.h"
```

Macros

- #define **SERVER_PORT** 8080
- #define **TCP_TAG** "TCP_SERVER"

Functions

- void **Create_Server** (void *pvParameter)
  Creates and starts a TCP server to listen for client connections.

- void **Handle_Client** (void *args)
  Handles communication with a single client.

- int **receive_data** (int sock, char *buffer, size_t size, uint8_t *timeout_counter, uint8_t max_timeout)
  Receives data from the client socket.

- void **Tcp_Init** ()
  Initializes the TCP server and starts listening for client connections.

Variables

- char * **GUI_USER** = "root"
  Username for GUI authentication.

- char * **GUI_PASS** = "otto"
  Password for GUI authentication.

Detailed Description

TCP server implementation to handle incoming client connections, authenticate, and exchange data.

Date

18 December 2024

Author

hakimmc

Macro Definition Documentation

#define SERVER_PORT 8080

Port number for the TCP server

#define TCP_TAG "TCP_SERVER"

Tag for logging

Function Documentation

void Create_Server (void * pvParameter)

Creates and starts a TCP server to listen for client connections.

Creates and starts the TCP server.

This function sets up a TCP socket, binds it to a specified address and port, and listens for incoming connections. For each connection, a new task is created to handle the client.

Parameters

pvParameter	Unused parameter for FreeRTOS task.

void Handle_Client (void * args)

Handles communication with a single client.

Handles client connections to the TCP server.

This function receives and processes commands from the client, authenticates the user, and manages communication based on different states. It handles login, data requests, and exit commands.

Parameters

args	Socket descriptor for the client.

int receive_data (int sock, char * buffer, size_t size, uint8_t * timeout_counter, uint8_t max_timeout)

Receives data from the client socket.

Receives data from a connected client.

This function receives data from the client and handles timeouts. If no data is received within a specified timeout period, it returns a timeout error.

Parameters

sock	Socket descriptor for the client.
buffer	Buffer to store received data.
size	Size of the buffer.
timeout_counter	Counter to track consecutive timeouts.
max_timeout	Maximum allowed consecutive timeouts before returning an error.

Returns

Length of received data, -1 if timeout occurs, or -2 if maximum timeouts reached.

void Tcp_Init ()

Initializes the TCP server and starts listening for client connections.

Initializes the TCP server.

< Initialize Wi-Fi in access point and station mode.

< Start the TCP server task.

Variable Documentation

char* GUI_PASS = "otto"

Password for GUI authentication.

char* GUI_USER = "root"

Username for GUI authentication.

src/Uart.c File Reference

UART initialization, transmission, and reception functions for UART communication.

```
#include "LedTaskInit.h"
#include "Uart.h"
#include "Can.h"
#include "Wifi.h"
#include "TcpServer.h"
#include "d2cc_lib.h"
#include "Bq76pl455.h"
```

Functions

- void **Uart_Init** (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)
  Initializes the UART peripheral with specified parameters.

- uint8_t **Uart_Transmit** (uint8_t uart_pin, uint8_t *data, uint8_t data_length)
  Transmits data via UART.

- int **Uart_Receive** (uint8_t uart_pin, uint8_t *data, uint8_t data_length, uint32_t timeout)
  Receives data via UART.

- uint8_t **IsTimeout** (uint32_t max_reach_time)
  Checks if a specified timeout duration has been reached.

Detailed Description

UART initialization, transmission, and reception functions for UART communication.

Date

18 December 2024

Author

hakimmc

Function Documentation

uint8_t IsTimeout (uint32_t max_reach_time)

Checks if a specified timeout duration has been reached.

Checks for a timeout condition.

This function checks if the specified maximum time has passed since the start of the function call.

Parameters

max_reach_time	Maximum time in ticks to wait before timing out.

Returns

0 if timeout occurred, 1 if still within the allowed time.

< Get current tick count.

< Wait for the timeout period to pass.

< Timeout reached.

< Timeout not reached.

void Uart_Init (uint8_t uart_pin, int baudrate, uint32_t rx_buffsize, uint8_t TXD_PIN, uint8_t RXD_PIN)

Initializes the UART peripheral with specified parameters.

Initializes the UART interface.

This function configures the UART with the specified baud rate, data bits, stop bits, and other settings. It also assigns pins for transmission and reception and installs the UART driver.

Parameters

uart_pin	UART port to configure (typically 0, 1, or 2).
baudrate	The baud rate for the UART communication.
rx_buffsize	The size of the receive buffer.
TXD_PIN	The GPIO pin for UART transmission (TX).
RXD_PIN	The GPIO pin for UART reception (RX).

< Configures the UART with the given settings.

< Assigns pins to UART.

< Installs the UART driver with specified buffer size.

int Uart_Receive (uint8_t uart_pin, uint8_t * data, uint8_t data_length, uint32_t timeout)

Receives data via UART.

Receives data from UART.

This function receives data from the UART port and stores it in the provided buffer.

Parameters

uart_pin	UART port to use for reception (typically 0, 1, or 2).
data	Pointer to the buffer to store received data.
data_length	Length of the data buffer.
timeout	Timeout for receiving data in milliseconds.

Returns

The number of bytes received or an error code.

< Reads data from UART into buffer.

uint8_t Uart_Transmit (uint8_t uart_pin, uint8_t * data, uint8_t data_length)

Transmits data via UART.

Transmits data over UART.

This function sends a specified amount of data through the UART transmission port.

Parameters

uart_pin	UART port to use for transmission (typically 0, 1, or 2).
data	Pointer to the data to be transmitted.

data_length	Length of the data to be transmitted.

Returns

1 if transmission is successful, 0 if failed.

< Writes data to UART.

< Transmission failure.

< Transmission success.

src/Wifi.c File Reference

Functions for initializing and managing Wi-Fi connectivity in AP (Access Point) and STA (Station) modes.

```
#include "Wifi.h"
```

Functions

- static void **event_handler** (void *arg, esp_event_base_t event_base, int32_t event_id, void *event_data)
  Event handler for Wi-Fi and IP events.

- void **wifi_init_ap_sta** (void)
  Initializes Wi-Fi in both Access Point (AP) and Station (STA) modes.

Detailed Description

Functions for initializing and managing Wi-Fi connectivity in AP (Access Point) and STA (Station) modes.

Date

18 December 2024

Author

hakimmc

Function Documentation

static void event_handler (void * arg, esp_event_base_t event_base, int32_t event_id, void * event_data) `[static]`

Event handler for Wi-Fi and IP events.

This function handles Wi-Fi and IP events, such as Wi-Fi connection status changes and obtaining an IP address. It also manages retry logic for Wi-Fi connection attempts.

Parameters

arg	Pointer to additional arguments (unused).
event_base	The event base (WIFI_EVENT or IP_EVENT).
event_id	The event ID (such as WIFI_EVENT_STA_START).
event_data	Event data associated with the event.

< Start the Wi-Fi connection process.

< Retry the connection.

< Log obtained IP address.

< Reset retry count.

void wifi_init_ap_sta (void)

Initializes Wi-Fi in both Access Point (AP) and Station (STA) modes.

Initializes the Wi-Fi in AP+STA mode.

This function initializes the Wi-Fi stack, configures the Wi-Fi interfaces, and starts the Wi-Fi service. It also registers event handlers for Wi-Fi and IP events, and connects to a Wi-Fi network in STA mode, if configured.

Note

This function configures the ESP32 in dual mode (AP and STA) and connects to a Wi-Fi network if `WIFI_CONNECT` is defined.

< Initialize network interface.

< Create the default event loop.

< Create the default Wi-Fi AP (Access Point) interface.

< Default Wi-Fi configuration.

< Initialize Wi-Fi driver with the default configuration.

< Register Wi-Fi event handler.

< Register IP event handler.

< AP SSID.

< AP password.

< Length of SSID.

< AP channel.

< Max connections to the AP.

< WPA/WPA2 PSK authentication.

< Set Wi-Fi mode to AP only.

< Set AP configuration.

< Start the Wi-Fi driver.

< Log completion of Wi-Fi initialization.

Index

INDEX