ESP32 RFID Cloner using MFRC522 and BLE

Generated by Doxygen 1.9.7

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 MFRC522 Register Addresses

Macros

- #define **MFRC522_REG_RESERVED00** 0x00 << 1
- #define **MFRC522_REG_COMMAND** 0x01 << 1
- #define **MFRC522_REG_COMIEN** 0x02 << 1
- #define **MFRC522_REG_DIVIEN** 0x03 << 1
- #define **MFRC522_REG_COMIRQ** 0x04 << 1
- #define **MFRC522_REG_DIVIRQ** 0x05 << 1
- #define **MFRC522_REG_ERROR** 0x06 << 1
- #define **MFRC522_REG_STATUS1** 0x07 << 1
- #define **MFRC522_REG_STATUS2** 0x08 << 1
- #define **MFRC522_REG_FIFO_DATA** 0x09 << 1
- #define **MFRC522_REG_FIFO_LEVEL** 0x0A << 1
- #define **MFRC522_REG_WATER_LEVEL** 0x0B << 1
- #define **MFRC522_REG_CONTROL** 0x0C << 1
- #define **MFRC522_REG_BIT_FRAMING** 0x0D << 1
- #define **MFRC522_REG_COLL** 0x0E << 1
- #define **MFRC522_REG_RESERVED01** 0x0F << 1
- #define **MFRC522_REG_RESERVED10** 0x10 << 1
- #define **MFRC522_REG_MODE** 0x11 << 1
- #define **MFRC522_REG_TX_MODE** 0x12 << 1
- #define **MFRC522_REG_RX_MODE** 0x13 << 1
- #define **MFRC522_REG_TX_CONTROL** 0x14 << 1
- #define **MFRC522_REG_TX_AUTO** 0x15 << 1
- #define **MFRC522_REG_TX_SEL** 0x16 << 1
- #define **MFRC522_REG_RX_SEL** 0x17 << 1
- #define **MFRC522_REG_RX_THRESHOLD** 0x18 << 1
- #define **MFRC522_REG_DEMOD** 0x19 << 1
- #define **MFRC522_REG_RESERVED11** 0x1A << 1
- #define **MFRC522_REG_RESERVED12** 0x1B << 1
- #define **MFRC522_REG_MIFARE** 0x1C << 1
- #define **MFRC522_REG_RESERVED13** 0x1D << 1
- #define **MFRC522_REG_RESERVED14** 0x1E << 1
- #define **MFRC522_REG_SERIALSPEED** 0x1F << 1
- #define **MFRC522_REG_RESERVED20** 0x20 << 1

- #define **MFRC522_REG_CRC_RESULT_M** 0x21 << 1
- #define **MFRC522_REG_CRC_RESULT_L** 0x22 << 1
- #define **MFRC522_REG_RESERVED21** 0x23 << 1
- #define **MFRC522_REG_MOD_WIDTH** 0x24 << 1
- #define **MFRC522_REG_RESERVED22** 0x25 << 1
- #define **MFRC522_REG_RF_CFG** 0x26 << 1
- #define **MFRC522_REG_GS_N** 0x27 << 1
- #define **MFRC522_REG_CWGS_PREG** 0x28 << 1
- #define **MFRC522_REG_MOD_GS_PREG** 0x29 << 1
- #define **MFRC522_REG_T_MODE** 0x2A << 1
- #define **MFRC522_REG_T_PRESCALER** 0x2B << 1
- #define **MFRC522_REG_T_RELOAD_H** 0x2C << 1
- #define **MFRC522_REG_T_RELOAD_L** 0x2D << 1
- #define **MFRC522_REG_T_COUNTER_VALUE_H** 0x2E << 1
- #define **MFRC522_REG_T_COUNTER_VALUE_L** 0x2F << 1
- #define **MFRC522_REG_RESERVED30** 0x30 << 1
- #define **MFRC522_REG_TEST_SEL1** 0x31 << 1
- #define **MFRC522_REG_TEST_SEL2** 0x32 << 1
- #define **MFRC522_REG_TEST_PIN_EN** 0x33 << 1
- #define **MFRC522_REG_TEST_PIN_VALUE** 0x34 << 1
- #define **MFRC522_REG_TEST_BUS** 0x35 << 1
- #define **MFRC522_REG_AUTO_TEST** 0x36 << 1
- #define **MFRC522_REG_VERSION** 0x37 << 1
- #define **MFRC522_REG_ANALOG_TEST** 0x38 << 1
- #define **MFRC522_REG_TEST_DAC1** 0x39 << 1
- #define **MFRC522_REG_TEST_DAC2** 0x3A << 1
- #define **MFRC522_REG_TEST_ADC** 0x3B << 1
- #define **MFRC522_REG_RESERVED31** 0x3C << 1
- #define **MFRC522_REG_RESERVED32** 0x3D << 1
- #define **MFRC522_REG_RESERVED33** 0x3E << 1
- #define **MFRC522_REG_RESERVED34** 0x3F << 1

4.1.1 Detailed Description

4.2 MFRC522 Commands

Macros

- #define **PCD_CMD_IDLE** 0x00
- #define **PCD_CMD_MEM** 0x01
- #define **PCD_CMD_GEN_RANDOM_ID** 0x02
- #define **PCD_CMD_CALC_CRC** 0x03
- #define **PCD_CMD_TRANSMIT** 0x04
- #define **PCD_CMD_NO_CMD_CHANGE** 0x07
- #define **PCD_CMD_RECEIVE** 0x08
- #define **PCD_CMD_TRANSCEIVE** 0x0C
- #define **PCD_CMD_MF_AUTHENT** 0x0E
- #define **PCD_CMD_SOFT_RESET** 0x0F

Enumerations

- enum **piccCmds_t** {
  PICC_CMD_REQA = 0x26 , **PICC_CMD_WUPA** = 0x52 , **PICC_CMD_CT** = 0x88 , **PICC_CMD_SEL_CL1** = 0x93 ,
  PICC_CMD_SEL_CL2 = 0x95 , **PICC_CMD_SEL_CL3** = 0x97 , **PICC_CMD_HLTA** = 0x50 , **PICC_CMD_↩ RATS** = 0xE0 ,
  PICC_CMD_MF_AUTH_KEY_A = 0x60 , **PICC_CMD_MF_AUTH_KEY_B** = 0x61 , **PICC_CMD_MF_READ** = 0x30 , **PICC_CMD_MF_WRITE** = 0xA0 ,
  PICC_CMD_MF_DECREMENT = 0xC0 , **PICC_CMD_MF_INCREMENT** = 0xC1 , **PICC_CMD_MF_↩ RESTORE** = 0xC2 , **PICC_CMD_MF_TRANSFER** = 0xB0 ,
  PICC_CMD_UL_WRITE = 0xA2 , **MFRC522_MIFARE_ACK** = 0x0A }

4.2.1 Detailed Description

Chapter 5

Class Documentation

5.1 gatts_profile_inst Struct Reference

Public Attributes

- esp_gatts_cb_t **gatts_cb**
- uint16_t **gatts_if**
- uint16_t **app_id**
- uint16_t **conn_id**
- uint16_t **service_handle**
- esp_gatt_srvc_id_t **service_id**
- uint16_t **char_handle**
- esp_bt_uuid_t **char_uuid**
- esp_gatt_perm_t **perm**
- esp_gatt_char_prop_t **property**
- uint16_t **descr_handle**
- esp_bt_uuid_t **descr_uuid**

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

- Sources/bluetooth_contr/BLE_Controller.h

5.2 Mifare1kKey_t Struct Reference

Public Attributes

- UniqueIdentifier_t uid
- uint8_t blockKey [6]
- uint8_t keyData [16][64]

5.2.1 Member Data Documentation

5.2.1.1 blockKey

```
uint8_t Mifare1kKey_t::blockKey[6]
```

Block key.

5.2.1.2 keyData

```
uint8_t Mifare1kKey_t::keyData[16][64]
```

Key data for 1k card (1024 bytes).

5.2.1.3 uid

```
UniqueIdentifier_t Mifare1kKey_t::uid
```

7 byte uid, the default

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

- Sources/drivers/MFRC522.h

5.3 prepare_type_env_t Struct Reference

Public Attributes

- uint8_t * **prepare_buf**
- int **prepare_len**

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

- Sources/bluetooth_contr/BLE_Controller.h

5.4 UniqueIdentifier_t Struct Reference

Public Attributes

- uidSize_t uidSize
- union {
  singleSizeUID_t singleSizeUidData
  doubleSizeUID_t doubleSizeUidData
  trippleSizeUID_t trippleSizeUidData
  } **uidData**

- uint8_t sakByte
- uint8_t bccByte

5.4.1 Member Data Documentation

5.4.1.1 bccByte

```
uint8_t UniqueIdentifier_t::bccByte
```

BCC byte.

5.4.1.2 doubleSizeUidData

```
doubleSizeUID_t UniqueIdentifier_t::doubleSizeUidData
```

Data for a double size UID.

5.4.1.3 sakByte

```
uint8_t UniqueIdentifier_t::sakByte
```

The SAK (Select acknowledge) byte returned from the PICC after successful selection.

5.4.1.4 singleSizeUidData

```
singleSizeUID_t UniqueIdentifier_t::singleSizeUidData
```

Data for a single size UID.

5.4.1.5 trippleSizeUidData

```
trippleSizeUID_t UniqueIdentifier_t::trippleSizeUidData
```

Data for a triple size UID.

5.4.1.6 uidSize

```
uidSize_t UniqueIdentifier_t::uidSize
```

The size of the UID.

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

- Sources/drivers/MFRC522.h

Chapter 6

File Documentation

6.1 BLE_Controller.h

```
00001 /*****************************************************************************
00002 * File Name: ESP 32 BLE Driver
00003 * Author: Jaime Malone
00004 * File Description:
00005 *
00006 *
00007 *
00008 *****************************************************************************/
00009
00010 #ifndef BLE_CONTROLLER_H
00011 #define BLE_CONTROLLER_H
00012
00013 #include <stdio.h>
00014 #include <stdlib.h>
00015 #include <string.h>
00016 #include <inttypes.h>
00017 #include "freertos/FreeRTOS.h"
00018 #include "freertos/task.h"
00019 #include "freertos/event_groups.h"
00020 #include "esp_system.h"
00021 #include "esp_log.h"
00022 #include "nvs_flash.h"
00023
00024 #include "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\include\esp32\include\esp_bt.h"
00025 #include
      "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\host\bluedroid\api\include\api\esp_gap_ble_api.h"
00026 #include
      "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\host\bluedroid\api\include\api\esp_gatts_api.h"
00027 #include
      "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\host\bluedroid\api\include\api\esp_bt_defs.h"
00028 #include
      "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\host\bluedroid\api\include\api\esp_bt_main.h"
00029 #include
      "C:\Espressif\frameworks\esp-idf-v5.0.2\components\bt\host\bluedroid\api\include\api\esp_gatt_common_api.h"
00030
00031 #include "sdkconfig.h"
00032
00033 typedef struct {
00034     uint8_t                 *prepare_buf;
00035     int                     prepare_len;
00036 } prepare_type_env_t;
00037
00038 struct gatts_profile_inst {
00039     esp_gatts_cb_t gatts_cb;
00040     uint16_t gatts_if;
00041     uint16_t app_id;
00042     uint16_t conn_id;
00043     uint16_t service_handle;
00044     esp_gatt_srvc_id_t service_id;
00045     uint16_t char_handle;
00046     esp_bt_uuid_t char_uuid;
00047     esp_gatt_perm_t perm;
00048     esp_gatt_char_prop_t property;
00049     uint16_t descr_handle;
00050     esp_bt_uuid_t descr_uuid;
00051 };
00052
```

```
00063 void example_write_event_env(esp_gatt_if_t gatts_if, prepare_type_env_t *prepare_write_env,
      esp_ble_gatts_cb_param_t *param);
00064
00074 void example_exec_write_event_env(prepare_type_env_t *prepare_write_env, esp_ble_gatts_cb_param_t
      *param);
00075
00084 static void gap_event_handler(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *param);
00085
00096 void example_write_event_env(esp_gatt_if_t gatts_if, prepare_type_env_t *prepare_write_env,
      esp_ble_gatts_cb_param_t *param);
00097
00107 void example_exec_write_event_env(prepare_type_env_t *prepare_write_env, esp_ble_gatts_cb_param_t
      *param);
00108
00119 static void gatts_profile_a_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if,
      esp_ble_gatts_cb_param_t *param);
00120
00131 static void gatts_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if,
      esp_ble_gatts_cb_param_t *param);
00132
00138 void BLE_init(void);
00139
00140 #endif /* BLE_CONTROLLER_H */
```

6.2 Sources/drivers/MFRC522.h File Reference

MFRC522 RFID Module.

```
#include ¨../unit_tests/unit_tests.h¨
#include ¨driver/spi_master.h¨
#include ¨esp_timer.h¨
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <driver/gpio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <assert.h>
```

Classes

- struct UniqueIdentifier_t
- struct Mifare1kKey_t

Macros

- #define NUM_SECTORE_MIFARE_1K 16
- #define NUM_BLOCKS_PER_SECTOR 4
- #define MIFARE_KEY_SIZE 6
- #define **MFRC522_REG_RESERVED00** 0x00 << 1
- #define **MFRC522_REG_COMMAND** 0x01 << 1
- #define **MFRC522_REG_COMIEN** 0x02 << 1
- #define **MFRC522_REG_DIVIEN** 0x03 << 1
- #define **MFRC522_REG_COMIRQ** 0x04 << 1
- #define **MFRC522_REG_DIVIRQ** 0x05 << 1
- #define **MFRC522_REG_ERROR** 0x06 << 1
- #define **MFRC522_REG_STATUS1** 0x07 << 1
- #define **MFRC522_REG_STATUS2** 0x08 << 1
- #define **MFRC522_REG_FIFO_DATA** 0x09 << 1

- #define **MFRC522_REG_FIFO_LEVEL** 0x0A << 1
- #define **MFRC522_REG_WATER_LEVEL** 0x0B << 1
- #define **MFRC522_REG_CONTROL** 0x0C << 1
- #define **MFRC522_REG_BIT_FRAMING** 0x0D << 1
- #define **MFRC522_REG_COLL** 0x0E << 1
- #define **MFRC522_REG_RESERVED01** 0x0F << 1
- #define **MFRC522_REG_RESERVED10** 0x10 << 1
- #define **MFRC522_REG_MODE** 0x11 << 1
- #define **MFRC522_REG_TX_MODE** 0x12 << 1
- #define **MFRC522_REG_RX_MODE** 0x13 << 1
- #define **MFRC522_REG_TX_CONTROL** 0x14 << 1
- #define **MFRC522_REG_TX_AUTO** 0x15 << 1
- #define **MFRC522_REG_TX_SEL** 0x16 << 1
- #define **MFRC522_REG_RX_SEL** 0x17 << 1
- #define **MFRC522_REG_RX_THRESHOLD** 0x18 << 1
- #define **MFRC522_REG_DEMOD** 0x19 << 1
- #define **MFRC522_REG_RESERVED11** 0x1A << 1
- #define **MFRC522_REG_RESERVED12** 0x1B << 1
- #define **MFRC522_REG_MIFARE** 0x1C << 1
- #define **MFRC522_REG_RESERVED13** 0x1D << 1
- #define **MFRC522_REG_RESERVED14** 0x1E << 1
- #define **MFRC522_REG_SERIALSPEED** 0x1F << 1
- #define **MFRC522_REG_RESERVED20** 0x20 << 1
- #define **MFRC522_REG_CRC_RESULT_M** 0x21 << 1
- #define **MFRC522_REG_CRC_RESULT_L** 0x22 << 1
- #define **MFRC522_REG_RESERVED21** 0x23 << 1
- #define **MFRC522_REG_MOD_WIDTH** 0x24 << 1
- #define **MFRC522_REG_RESERVED22** 0x25 << 1
- #define **MFRC522_REG_RF_CFG** 0x26 << 1
- #define **MFRC522_REG_GS_N** 0x27 << 1
- #define **MFRC522_REG_CWGS_PREG** 0x28 << 1
- #define **MFRC522_REG_MOD_GS_PREG** 0x29 << 1
- #define **MFRC522_REG_T_MODE** 0x2A << 1
- #define **MFRC522_REG_T_PRESCALER** 0x2B << 1
- #define **MFRC522_REG_T_RELOAD_H** 0x2C << 1
- #define **MFRC522_REG_T_RELOAD_L** 0x2D << 1
- #define **MFRC522_REG_T_COUNTER_VALUE_H** 0x2E << 1
- #define **MFRC522_REG_T_COUNTER_VALUE_L** 0x2F << 1
- #define **MFRC522_REG_RESERVED30** 0x30 << 1
- #define **MFRC522_REG_TEST_SEL1** 0x31 << 1
- #define **MFRC522_REG_TEST_SEL2** 0x32 << 1
- #define **MFRC522_REG_TEST_PIN_EN** 0x33 << 1
- #define **MFRC522_REG_TEST_PIN_VALUE** 0x34 << 1
- #define **MFRC522_REG_TEST_BUS** 0x35 << 1
- #define **MFRC522_REG_AUTO_TEST** 0x36 << 1
- #define **MFRC522_REG_VERSION** 0x37 << 1
- #define **MFRC522_REG_ANALOG_TEST** 0x38 << 1
- #define **MFRC522_REG_TEST_DAC1** 0x39 << 1
- #define **MFRC522_REG_TEST_DAC2** 0x3A << 1
- #define **MFRC522_REG_TEST_ADC** 0x3B << 1
- #define **MFRC522_REG_RESERVED31** 0x3C << 1
- #define **MFRC522_REG_RESERVED32** 0x3D << 1
- #define **MFRC522_REG_RESERVED33** 0x3E << 1
- #define **MFRC522_REG_RESERVED34** 0x3F << 1
- #define **PCD_CMD_IDLE** 0x00

- #define **PCD_CMD_MEM** 0x01
- #define **PCD_CMD_GEN_RANDOM_ID** 0x02
- #define **PCD_CMD_CALC_CRC** 0x03
- #define **PCD_CMD_TRANSMIT** 0x04
- #define **PCD_CMD_NO_CMD_CHANGE** 0x07
- #define **PCD_CMD_RECEIVE** 0x08
- #define **PCD_CMD_TRANSCEIVE** 0x0C
- #define **PCD_CMD_MF_AUTHENT** 0x0E
- #define **PCD_CMD_SOFT_RESET** 0x0F

Typedefs

- typedef uint8_t **singleSizeUID_t**[4]

  Typedef for single size UID. It consists of 4 bytes.
- typedef uint8_t **doubleSizeUID_t**[7]

  Typedef for double size UID. It consists of 7 bytes.
- typedef uint8_t **trippleSizeUID_t**[10]

  Typedef for triple size UID. It consists of 10 bytes.

Enumerations

- enum bitFraming_t { sevenBit = 0x07 , eightBit = 0x08 }

  Enum typedef for bit framing. It can either be sevenBit or eightBit.
- enum uidSize_t { fourBytesSingle = 4 , sevenBytesDouble = 7 , tenBytesTripple = 10 }

  Enum typedef for UID size. It can be either fourBytesSingle, sevenBytesDouble or tenBytesTripple.
- enum **piccCmds_t** {
  PICC_CMD_REQA = 0x26 , **PICC_CMD_WUPA** = 0x52 , **PICC_CMD_CT** = 0x88 , **PICC_CMD_SEL_CL1** = 0x93 ,
  PICC_CMD_SEL_CL2 = 0x95 , **PICC_CMD_SEL_CL3** = 0x97 , **PICC_CMD_HLTA** = 0x50 , **PICC_CMD_RATS** = 0xE0 ,
  PICC_CMD_MF_AUTH_KEY_A = 0x60 , **PICC_CMD_MF_AUTH_KEY_B** = 0x61 , **PICC_CMD_MF_READ** = 0x30 , **PICC_CMD_MF_WRITE** = 0xA0 ,
  PICC_CMD_MF_DECREMENT = 0xC0 , **PICC_CMD_MF_INCREMENT** = 0xC1 , **PICC_CMD_MF_RESTORE** = 0xC2 , **PICC_CMD_MF_TRANSFER** = 0xB0 ,
  PICC_CMD_UL_WRITE = 0xA2 , **MFRC522_MIFARE_ACK** = 0x0A }

Functions

- esp_err_t xMFRC522_WriteRegister (spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t value)

  Writes a value to the specified register in the MFRC522.
- esp_err_t xMFRC522_ReadRegister (spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t *data)

  Reads the value from the specified register in the MFRC522.
- esp_err_t xMFRC522_ReadRegisterArr (spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t *dataArr, uint8_t dataSize)

  Reads multiple consecutive registers in the MFRC522.
- esp_err_t xMFRC522_WriteRegisterArr (spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t *dataArr, uint8_t dataSize)

  Writes multiple consecutive registers in the MFRC522.
- esp_err_t xMFRC522_Init (spi_device_handle_t *spiHandle, uint8_t rstPin)

Initializes the MFRC522 RFID module.

- esp_err_t xMFRC522_ClrRegBitMask (spi_device_handle_t *spiHandle, uint8_t registerAdress, uint8_t mask)

  Clears the specified bits in the register of the MFRC522.
- bool xMFRC522_IsCardPresent (spi_device_handle_t *spiHandle)

  Checks if a card is present.
- esp_err_t xMFRC522_AntennaOn (spi_device_handle_t *spiHandle)

  Turns on the antenna of the MFRC522.
- esp_err_t xMFRC522_SelfTest (spi_device_handle_t *spiHandle, uint8_t rstPin)

  Performs a self-test of the MFRC522.
- esp_err_t xMFRC522_Reset (spi_device_handle_t *spiHandle)

  Resets the MFRC522.
- esp_err_t xMFRC522_Transcieve (spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf, uint8_t bufSize, bitFraming_t bitFrame)

  Transmits data to the MFRC522 and receives the response.
- esp_err_t xMFRC522_MF_Authent (spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf, uint8_t bufSize, bitFraming_t bitFrame)

  Performs MIFARE authentication with the MFRC522.
- esp_err_t xMFRC522_CommWithMifare (uint8_t cmd, spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf, uint8_t bufSize, bitFraming_t bitFrame)

  Performs communication with a MIFARE card using the MFRC522.
- esp_err_t xMFRC522_SetRegBitMask (spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t mask)

  Sets the specified bits in the register of the MFRC522.
- esp_err_t xMFRC522_CalculateCRC (spi_device_handle_t *spiHandle, uint8_t *buf, uint8_t bufLen, uint8_t resultBuf[2])

  Calculates the CRC value for the given buffer.
- UniqueIdentifier_t * xMifare_ReadUID (spi_device_handle_t *spiHandle, uidSize_t uidSize)

  Reads the UID from the MFRC522 for the specified UID size.
- Mifare1kKey_t * xMifare_GetKeyData (spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID)

  Retrieves the key data for the specified UID from the MFRC522.
- esp_err_t xMifare_WriteKey (spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID, uint8_t data[45][16])

  Writes the provided key data to the specified UID in the MFRC522.
- esp_err_t xMifare_WriteKeyBlock (spi_device_handle_t *spiHandle, uint8_t blockAddress, UniqueIdentifier_t *UID, uint8_t data[16])

  Writes the provided data to the specified block address for the specified UID in the MFRC522.
- esp_err_t xMifare_ReadKeyBlock (spi_device_handle_t *spiHandle, uint8_t blockAddress, UniqueIdentifier_t *UID)

  Reads the key block at the specified block address for the specified UID in the MFRC522.
- esp_err_t xMifare_Authenticate (spi_device_handle_t *spiHandle, uint8_t cmd, uint8_t blockAddress, uint8_t *key, UniqueIdentifier_t *UID)

  Performs MIFARE authentication for the specified block address and key with the MFRC522.
- void vMifare_PrintUID (UniqueIdentifier_t *UID)

  Prints the UID information to the console.
- void vMFRC522_GetAndPrintFifoBuf (spi_device_handle_t *spiHandle, uint8_t *fifoBuf, bool print)

  Retrieves and prints the FIFO buffer data from the MFRC522.
- void vMifare_PrintKey (Mifare1kKey_t *key)

  Prints the key data to the console.

6.2.1 Detailed Description

MFRC522 RFID Module.

Author

Jack Lukomski

Date

Date: 2023-06-02

This file contains the declarations for the MFRC522 RFID module. It provides functions for initializing, communicating, and authenticating with MIFARE RFID cards using the MFRC522 chip.

6.2.2 Macro Definition Documentation

6.2.2.1 MIFARE_KEY_SIZE

```
#define MIFARE_KEY_SIZE 6
```

Size of MIFARE key.

6.2.2.2 NUM_BLOCKS_PER_SECTOR

```
#define NUM_BLOCKS_PER_SECTOR 4
```

Number of blocks per sector.

6.2.2.3 NUM_SECTORE_MIFARE_1K

```
#define NUM_SECTORE_MIFARE_1K 16
```

Number of sectors in MIFARE 1k.

6.2.3 Enumeration Type Documentation

6.2.3.1 bitFraming_t

```
enum bitFraming_t
```

Enum typedef for bit framing. It can either be sevenBit or eightBit.

Enumerator

sevenBit	7 bit framing
eightBit	8 bit framing

6.2.3.2 uidSize_t

```
enum uidSize_t
```

Enum typedef for UID size. It can be either fourBytesSingle, sevenBytesDouble or tenBytesTripple.

Enumerator

fourBytesSingle	UID of 4 bytes
sevenBytesDouble	UID of 7 bytes
tenBytesTripple	UID of 10 bytes

6.2.4 Function Documentation

6.2.4.1 vMFRC522_GetAndPrintFifoBuf()

```
void vMFRC522_GetAndPrintFifoBuf (
            spi_device_handle_t * spiHandle,
            uint8_t * fifoBuf,
            bool print )
```

Retrieves and prints the FIFO buffer data from the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
fifoBuf	Pointer to the buffer to store the FIFO data.
print	Flag indicating whether to print the FIFO data or not.

6.2.4.2 vMifare_PrintKey()

```
void vMifare_PrintKey (
            Mifare1kKey_t * key )
```

Prints the key data to the console.

Parameters

key	Pointer to the Mifare1kKey_t structure containing the key data.

6.2.4.3 vMifare_PrintUID()

```
void vMifare_PrintUID (
            UniqueIdentifier_t * UID )
```

Prints the UID information to the console.

Parameters

UID	Pointer to the UniqueIdentifier_t structure containing the UID.

6.2.4.4 xMFRC522_AntennaOn()

```
esp_err_t xMFRC522_AntennaOn (
            spi_device_handle_t * spiHandle )
```

Turns on the antenna of the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.5 xMFRC522_CalculateCRC()

```
esp_err_t xMFRC522_CalculateCRC (
            spi_device_handle_t * spiHandle,
            uint8_t * buf,
            uint8_t bufLen,
            uint8_t resultBuf[2] )
```

Calculates the CRC value for the given buffer.

Parameters

spiHandle	Pointer to the SPI device handle.
buf	Pointer to the buffer for which the CRC is to be calculated.
bufLen	Length of the buffer.
resultBuf	Pointer to the buffer to store the calculated CRC value (2 bytes).

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.6 xMFRC522_ClrRegBitMask()

```
esp_err_t xMFRC522_ClrRegBitMask (
            spi_device_handle_t * spiHandle,
            uint8_t registerAdress,
            uint8_t mask )
```

Clears the specified bits in the register of the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAdress	The address of the register to modify.
mask	The bitmask of the bits to clear.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.7 xMFRC522_CommWithMifare()

```
esp_err_t xMFRC522_CommWithMifare (
            uint8_t cmd,
            spi_device_handle_t * spiHandle,
            uint8_t waitIrq,
            uint8_t * cmdBuf,
            uint8_t bufSize,
            bitFraming_t bitFrame )
```

Performs communication with a MIFARE card using the MFRC522.

Parameters

cmd	Command to be sent to the MIFARE card.
spiHandle	Pointer to the SPI device handle.
waitIrq	Wait for the command to complete (1) or not (0).
cmdBuf	Pointer to the command buffer to be transmitted.
bufSize	Size of the command buffer.
bitFrame	The bit framing type.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.8 xMFRC522_Init()

```
esp_err_t xMFRC522_Init (
            spi_device_handle_t * spiHandle,
            uint8_t rstPin )
```

Initializes the MFRC522 RFID module.

Parameters

spiHandle	Pointer to the SPI device handle.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.9 xMFRC522_IsCardPresent()

```
bool xMFRC522_IsCardPresent (
            spi_device_handle_t * spiHandle )
```

Checks if a card is present.

Parameters

spiHandle	Pointer to the SPI device handle.

Returns

True if a card is present, false otherwise.

6.2.4.10 xMFRC522_MF_Authent()

```
esp_err_t xMFRC522_MF_Authent (
            spi_device_handle_t * spiHandle,
            uint8_t waitIrq,
            uint8_t * cmdBuf,
            uint8_t bufSize,
            bitFraming_t bitFrame )
```

Performs MIFARE authentication with the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
waitIrq	Wait for the command to complete (1) or not (0).
cmdBuf	Pointer to the command buffer to be transmitted.
bufSize	Size of the command buffer.
bitFrame	The bit framing type.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.11 xMFRC522_ReadRegister()

```
esp_err_t xMFRC522_ReadRegister (
            spi_device_handle_t * spiHandle,
            uint8_t registerAddress,
            uint8_t * data )
```

Reads the value from the specified register in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAddress	The address of the register to read.
data	Pointer to store the read value.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.12 xMFRC522_ReadRegisterArr()

```
esp_err_t xMFRC522_ReadRegisterArr (
            spi_device_handle_t * spiHandle,
            uint8_t registerAddress,
            uint8_t * dataArr,
            uint8_t dataSize )
```

Reads multiple consecutive registers in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAddress	The address of the first register to read.
dataArr	Pointer to the array to store the read values.
dataSize	The number of registers to read.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.13 xMFRC522_Reset()

```
esp_err_t xMFRC522_Reset (
            spi_device_handle_t * spiHandle )
```

Resets the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.14 xMFRC522_SelfTest()

```
esp_err_t xMFRC522_SelfTest (
            spi_device_handle_t * spiHandle,
            uint8_t rstPin )
```

Performs a self-test of the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.15 xMFRC522_SetRegBitMask()

```
esp_err_t xMFRC522_SetRegBitMask (
            spi_device_handle_t * spiHandle,
            uint8_t registerAddress,
            uint8_t mask )
```

Sets the specified bits in the register of the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAddress	The address of the register to modify.
mask	The bitmask of the bits to set.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.16 xMFRC522_Transcieve()

```
esp_err_t xMFRC522_Transcieve (
            spi_device_handle_t * spiHandle,
            uint8_t waitIrq,
            uint8_t * cmdBuf,
            uint8_t bufSize,
            bitFraming_t bitFrame )
```

Transmits data to the MFRC522 and receives the response.

Parameters

spiHandle	Pointer to the SPI device handle.
waitIrq	Wait for the command to complete (1) or not (0).
cmdBuf	Pointer to the command buffer to be transmitted.
bufSize	Size of the command buffer.
bitFrame	The bit framing type.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.17 xMFRC522_WriteRegister()

```
esp_err_t xMFRC522_WriteRegister (
            spi_device_handle_t * spiHandle,
            uint8_t registerAddress,
            uint8_t value )
```

Writes a value to the specified register in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAddress	The address of the register to write.
value	The value to write to the register.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.18 xMFRC522_WriteRegisterArr()

```
esp_err_t xMFRC522_WriteRegisterArr (
            spi_device_handle_t * spiHandle,
            uint8_t registerAddress,
            uint8_t * dataArr,
            uint8_t dataSize )
```

Writes multiple consecutive registers in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
registerAddress	The address of the first register to write.
dataArr	Pointer to the array of values to write.
dataSize	The number of registers to write.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.19 xMifare_Authenticate()

```
esp_err_t xMifare_Authenticate (
            spi_device_handle_t * spiHandle,
```

```
            uint8_t cmd,
            uint8_t blockAddress,
            uint8_t * key,
            UniqueIdentifier_t * UID )
```

Performs MIFARE authentication for the specified block address and key with the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
cmd	The command for the authentication (PICC_CMD_MF_AUTH_KEY_A or PICC_CMD_MF_AUTH_KEY_B).
blockAddress	The block address to authenticate.
key	The key for authentication.
UID	Pointer to the UniqueIdentifier_t structure containing the UID.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.20 xMifare_GetKeyData()

```
Mifare1kKey_t * xMifare_GetKeyData (
            spi_device_handle_t * spiHandle,
            UniqueIdentifier_t * UID )
```

Retrieves the key data for the specified UID from the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
UID	Pointer to the UniqueIdentifier_t structure containing the UID.

Returns

Pointer to the Mifare1kKey_t structure containing the key data, or NULL on failure.

6.2.4.21 xMifare_ReadKeyBlock()

```
esp_err_t xMifare_ReadKeyBlock (
            spi_device_handle_t * spiHandle,
            uint8_t blockAddress,
            UniqueIdentifier_t * UID )
```

Reads the key block at the specified block address for the specified UID in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
blockAddress	The block address to read from.
UID	Pointer to the UniqueIdentifier_t structure containing the UID.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.22 xMifare_ReadUID()

```
UniqueIdentifier_t * xMifare_ReadUID (
            spi_device_handle_t * spiHandle,
            uidSize_t uidSize )
```

Reads the UID from the MFRC522 for the specified UID size.

Parameters

spiHandle	Pointer to the SPI device handle.
uidSize	The size of the UID.

Returns

Pointer to the UniqueIdentifier_t structure containing the UID information, or NULL on failure.

6.2.4.23 xMifare_WriteKey()

```
esp_err_t xMifare_WriteKey (
            spi_device_handle_t * spiHandle,
            UniqueIdentifier_t * UID,
            uint8_t data[45][16] )
```

Writes the provided key data to the specified UID in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
UID	Pointer to the UniqueIdentifier_t structure containing the UID.
data	The key data to be written.

Returns

ESP_OK if successful, otherwise an error code.

6.2.4.24 xMifare_WriteKeyBlock()

```
esp_err_t xMifare_WriteKeyBlock (
            spi_device_handle_t * spiHandle,
            uint8_t blockAddress,
            UniqueIdentifier_t * UID,
            uint8_t data[16] )
```

Writes the provided data to the specified block address for the specified UID in the MFRC522.

Parameters

spiHandle	Pointer to the SPI device handle.
blockAddress	The block address to write the data to.
UID	Pointer to the UniqueIdentifier_t structure containing the UID.
data	The data to be written.

Returns

ESP_OK if successful, otherwise an error code.

6.3 MFRC522.h

Go to the documentation of this file.

```
00001
00011 #ifndef _MFRC522_H_
00012 #define _MFRC522_H_
00013
00014 #include "../unit_tests/unit_tests.h"
00015 #include "driver/spi_master.h"
00016 #include "esp_timer.h"
00017 #include <freertos/FreeRTOS.h>
00018 #include <freertos/task.h>
00019 #include <driver/gpio.h>
00020 #include <stdlib.h>
00021 #include <string.h>
00022 #include <time.h>
00023 #include <assert.h>
00024
00025 // Number of bytes in the UID. 4, 7 or 10.
00029 typedef uint8_t singleSizeUID_t[4];
00030
00034 typedef uint8_t doubleSizeUID_t[7];
00035
00039 typedef uint8_t trippleSizeUID_t[10];
00040
00044 typedef enum {
00045     sevenBit = 0x07,
00046     eightBit = 0x08,
00047 } bitFraming_t;
00048
00052 typedef enum {
00053     fourBytesSingle = 4,
00054     sevenBytesDouble = 7,
00055     tenBytesTripple = 10,
00056 } uidSize_t;
00057
00061 typedef struct {
00062     uidSize_t uidSize;
00063     union {
00064         singleSizeUID_t singleSizeUidData;
00065         doubleSizeUID_t doubleSizeUidData;
00066         trippleSizeUID_t trippleSizeUidData;
00067     } uidData;
00068     uint8_t sakByte;
00069     uint8_t bccByte;
00070 } UniqueIdentifier_t;
00071
00075 typedef struct {
00076     UniqueIdentifier_t uid;
00077     uint8_t blockKey[6];
00078     uint8_t keyData[16][64];
00079 } Mifare1kKey_t;
00080
00081 // Definitions for some constants
00082 #define NUM_SECTORE_MIFARE_1K 16
00083 #define NUM_BLOCKS_PER_SECTOR 4
00084 #define MIFARE_KEY_SIZE 6
00090 #define MFRC522_REG_RESERVED00          0x00 << 1
00091 #define MFRC522_REG_COMMAND             0x01 << 1
00092 #define MFRC522_REG_COMIEN              0x02 << 1
00093 #define MFRC522_REG_DIVIEN              0x03 << 1
00094 #define MFRC522_REG_COMIRQ              0x04 << 1
00095 #define MFRC522_REG_DIVIRQ              0x05 << 1
```

```
00096 #define MFRC522_REG_ERROR               0x06 << 1
00097 #define MFRC522_REG_STATUS1             0x07 << 1
00098 #define MFRC522_REG_STATUS2             0x08 << 1
00099 #define MFRC522_REG_FIFO_DATA           0x09 << 1
00100 #define MFRC522_REG_FIFO_LEVEL          0x0A << 1
00101 #define MFRC522_REG_WATER_LEVEL         0x0B << 1
00102 #define MFRC522_REG_CONTROL             0x0C << 1
00103 #define MFRC522_REG_BIT_FRAMING         0x0D << 1
00104 #define MFRC522_REG_COLL                0x0E << 1
00105 #define MFRC522_REG_RESERVED01          0x0F << 1
00106
00107 // Page 1: Command
00108 #define MFRC522_REG_RESERVED10          0x10 << 1
00109 #define MFRC522_REG_MODE                0x11 << 1
00110 #define MFRC522_REG_TX_MODE             0x12 << 1
00111 #define MFRC522_REG_RX_MODE             0x13 << 1
00112 #define MFRC522_REG_TX_CONTROL          0x14 << 1
00113 #define MFRC522_REG_TX_AUTO             0x15 << 1
00114 #define MFRC522_REG_TX_SEL              0x16 << 1
00115 #define MFRC522_REG_RX_SEL              0x17 << 1
00116 #define MFRC522_REG_RX_THRESHOLD        0x18 << 1
00117 #define MFRC522_REG_DEMOD               0x19 << 1
00118 #define MFRC522_REG_RESERVED11          0x1A << 1
00119 #define MFRC522_REG_RESERVED12          0x1B << 1
00120 #define MFRC522_REG_MIFARE              0x1C << 1
00121 #define MFRC522_REG_RESERVED13          0x1D << 1
00122 #define MFRC522_REG_RESERVED14          0x1E << 1
00123 #define MFRC522_REG_SERIALSPEED         0x1F << 1
00124
00125 // Page 2: Configuration
00126 #define MFRC522_REG_RESERVED20          0x20 << 1
00127 #define MFRC522_REG_CRC_RESULT_M        0x21 << 1
00128 #define MFRC522_REG_CRC_RESULT_L        0x22 << 1
00129 #define MFRC522_REG_RESERVED21          0x23 << 1
00130 #define MFRC522_REG_MOD_WIDTH           0x24 << 1
00131 #define MFRC522_REG_RESERVED22          0x25 << 1
00132 #define MFRC522_REG_RF_CFG              0x26 << 1
00133 #define MFRC522_REG_GS_N                0x27 << 1
00134 #define MFRC522_REG_CWGS_PREG           0x28 << 1
00135 #define MFRC522_REG_MOD_GS_PREG         0x29 << 1
00136 #define MFRC522_REG_T_MODE              0x2A << 1
00137 #define MFRC522_REG_T_PRESCALER         0x2B << 1
00138 #define MFRC522_REG_T_RELOAD_H          0x2C << 1
00139 #define MFRC522_REG_T_RELOAD_L          0x2D << 1
00140 #define MFRC522_REG_T_COUNTER_VALUE_H   0x2E << 1
00141 #define MFRC522_REG_T_COUNTER_VALUE_L   0x2F << 1
00142
00143 // Page 3: Test
00144 #define MFRC522_REG_RESERVED30          0x30 << 1
00145 #define MFRC522_REG_TEST_SEL1           0x31 << 1
00146 #define MFRC522_REG_TEST_SEL2           0x32 << 1
00147 #define MFRC522_REG_TEST_PIN_EN         0x33 << 1
00148 #define MFRC522_REG_TEST_PIN_VALUE      0x34 << 1
00149 #define MFRC522_REG_TEST_BUS            0x35 << 1
00150 #define MFRC522_REG_AUTO_TEST           0x36 << 1
00151 #define MFRC522_REG_VERSION             0x37 << 1
00152 #define MFRC522_REG_ANALOG_TEST         0x38 << 1
00153 #define MFRC522_REG_TEST_DAC1           0x39 << 1
00154 #define MFRC522_REG_TEST_DAC2           0x3A << 1
00155 #define MFRC522_REG_TEST_ADC            0x3B << 1
00156 #define MFRC522_REG_RESERVED31          0x3C << 1
00157 #define MFRC522_REG_RESERVED32          0x3D << 1
00158 #define MFRC522_REG_RESERVED33          0x3E << 1
00159 #define MFRC522_REG_RESERVED34          0x3F << 1
00166 // Commands sent to the PICC.
00167 typedef enum {
00168     PICC_CMD_REQA         = 0x26,
00169     PICC_CMD_WUPA         = 0x52,
00170     PICC_CMD_CT           = 0x88,
00171     PICC_CMD_SEL_CL1      = 0x93,
00172     PICC_CMD_SEL_CL2      = 0x95,
00173     PICC_CMD_SEL_CL3      = 0x97,
00174     PICC_CMD_HLTA         = 0x50,
00175     PICC_CMD_RATS         = 0xE0,
00176     PICC_CMD_MF_AUTH_KEY_A = 0x60,
00177     PICC_CMD_MF_AUTH_KEY_B = 0x61,
00178     PICC_CMD_MF_READ      = 0x30,
00179     PICC_CMD_MF_WRITE     = 0xA0,
00180     PICC_CMD_MF_DECREMENT = 0xC0,
00181     PICC_CMD_MF_INCREMENT = 0xC1,
00182     PICC_CMD_MF_RESTORE   = 0xC2,
00183     PICC_CMD_MF_TRANSFER  = 0xB0,
00184     PICC_CMD_UL_WRITE     = 0xA2,
00185     MFRC522_MIFARE_ACK    = 0x0A,
00186 } piccCmds_t;
00187
00188
```

```
00189 // MFRC522's commands for the PCD.
00190 #define PCD_CMD_IDLE                0x00  // NO action; cancels current command execution
00191 #define PCD_CMD_MEM                 0x01  // Stores 25 bytes into the internal buffer
00192 #define PCD_CMD_GEN_RANDOM_ID       0x02  // Generates a 10-byte random ID number
00193 #define PCD_CMD_CALC_CRC            0x03  // Activates the CRC coprocessor or performs a self-test
00194 #define PCD_CMD_TRANSMIT            0x04  // Transmits data from the FIFO buffer
00195 #define PCD_CMD_NO_CMD_CHANGE       0x07  // Can be used to modify the CommandReg register bits
      without affecting the command, if any, currently being executed
00196 #define PCD_CMD_RECEIVE             0x08  // Activates the receiver circuits
00197 #define PCD_CMD_TRANSCEIVE          0x0C  // Transmits data from FIFO buffer to antenna and
      automatically activates the receiver after transmission
00198 #define PCD_CMD_MF_AUTHENT          0x0E  // Performs the MIFARE standard authentication as a reader
00199 #define PCD_CMD_SOFT_RESET          0x0F  // Resets the MFRC522
00210 esp_err_t xMFRC522_WriteRegister(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
      value);
00211
00220 esp_err_t xMFRC522_ReadRegister(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
      *data);
00221
00231 esp_err_t xMFRC522_ReadRegisterArr(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
      *dataArr, uint8_t dataSize);
00232
00242 esp_err_t xMFRC522_WriteRegisterArr(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
      *dataArr, uint8_t dataSize);
00243
00250 esp_err_t xMFRC522_Init(spi_device_handle_t *spiHandle, uint8_t rstPin);
00251
00260 esp_err_t xMFRC522_ClrRegBitMask(spi_device_handle_t *spiHandle, uint8_t registerAdress, uint8_t
      mask);
00261
00268 bool xMFRC522_IsCardPresent(spi_device_handle_t *spiHandle);
00269
00276 esp_err_t xMFRC522_AntennaOn(spi_device_handle_t *spiHandle);
00277
00284 esp_err_t xMFRC522_SelfTest(spi_device_handle_t *spiHandle, uint8_t rstPin);
00285
00292 esp_err_t xMFRC522_Reset(spi_device_handle_t *spiHandle);
00293
00304 esp_err_t xMFRC522_Transcieve(spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf,
      uint8_t bufSize, bitFraming_t bitFrame);
00305
00316 esp_err_t xMFRC522_MF_Authent(spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf,
      uint8_t bufSize, bitFraming_t bitFrame);
00317
00329 esp_err_t xMFRC522_CommWithMifare(uint8_t cmd, spi_device_handle_t *spiHandle, uint8_t waitIrq,
      uint8_t *cmdBuf, uint8_t bufSize, bitFraming_t bitFrame);
00330
00339 esp_err_t xMFRC522_SetRegBitMask(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
      mask);
00340
00350 esp_err_t xMFRC522_CalculateCRC(spi_device_handle_t *spiHandle, uint8_t *buf, uint8_t bufLen, uint8_t
      resultBuf[2]);
00351
00359 UniqueIdentifier_t *xMifare_ReadUID(spi_device_handle_t *spiHandle, uidSize_t uidSize);
00360
00368 Mifare1kKey_t *xMifare_GetKeyData(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID);
00369
00378 esp_err_t xMifare_WriteKey(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID, uint8_t
      data[45][16]);
00379
00389 esp_err_t xMifare_WriteKeyBlock(spi_device_handle_t *spiHandle, uint8_t blockAddress,
      UniqueIdentifier_t *UID, uint8_t data[16]);
00390
00398 static esp_err_t xMifare_SetSakByte(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID);
00399
00408 esp_err_t xMifare_ReadKeyBlock(spi_device_handle_t *spiHandle, uint8_t blockAddress,
      UniqueIdentifier_t *UID);
00409
00418 static bool xPrv_Mifare_BlockCheckChar(uint8_t *bufData, uint8_t bufSize, UniqueIdentifier_t *UID);
00419
00430 esp_err_t xMifare_Authenticate(spi_device_handle_t *spiHandle, uint8_t cmd, uint8_t blockAddress,
      uint8_t *key, UniqueIdentifier_t *UID);
00431
00437 void vMifare_PrintUID(UniqueIdentifier_t *UID);
00438
00446 void vMFRC522_GetAndPrintFifoBuf(spi_device_handle_t *spiHandle, uint8_t *fifoBuf, bool print);
00447
00453 void vMifare_PrintKey(Mifare1kKey_t *key);
00454
00455
00456 #endif // _MFRC522_H_
```

6.4 Sources/unit_tests/unit_tests.h File Reference

Unit Test Framework.

```
#include <stdio.h>
```

Macros

- #define **UNIT_TESTS** 0
- #define ESP_ASSERT(name, condition, message)

  Macro for performing unit test assertions.

6.4.1 Detailed Description

Unit Test Framework.

This file contains macros and utilities for performing unit tests.

Author

Jack Lukomski

6.4.2 Macro Definition Documentation

6.4.2.1 ESP_ASSERT

```
#define ESP_ASSERT(
                name,
                condition,
                message )
```

Value:

```
    { \
    printf("Unit Test Name: %s, Test Result: ", name); \
    if (!(condition)) { \
        printf("Test Failed: %s\n", message); \
    } else { \
        printf("Test Passed!\n"); \
    } \
}
```

Macro for performing unit test assertions.

This macro checks the condition and prints the test result. If the condition is false, it prints the test failure message.

Parameters

name	The name of the unit test.
condition	The condition to be checked.
message	The failure message to be printed.

6.5 unit_tests.h

Go to the documentation of this file.

```
00001
00010 #ifndef UNIT_TESTS_H
00011 #define UNIT_TESTS_H
00012
00013 #include <stdio.h>
00014
00015 #define UNIT_TESTS 0
00016
00028 #define ESP_ASSERT(name, condition, message) { \
00029     printf("Unit Test Name: %s, Test Result: ", name); \
00030     if (!(condition)) { \
00031         printf("Test Failed: %s\n", message); \
00032     } else { \
00033         printf("Test Passed!\n"); \
00034     } \
00035 }
00036
00037 #endif /* UNIT_TESTS_H */
```

Index