

RFID_ESP32

Generated by Doxygen 1.9.7

1 Module Index	1
1.1 Modules	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Module Documentation	7
4.1 MFRC522 Register Addresses	7
4.1.1 Detailed Description	8
4.2 MFRC522 Commands	8
4.2.1 Detailed Description	9
5 Class Documentation	11
5.1 Mifare1kKey_t Struct Reference	11
5.1.1 Member Data Documentation	11
5.1.1.1 blockKey	11
5.1.1.2 keyData	11
5.1.1.3 uid	11
5.2 UniqueIdentifier_t Struct Reference	12
5.2.1 Member Data Documentation	12
5.2.1.1 bccByte	12
5.2.1.2 doubleSizeUidData	12
5.2.1.3 sakByte	12
5.2.1.4 singleSizeUidData	12
5.2.1.5 trippleSizeUidData	12
5.2.1.6 uidSize	12
6 File Documentation	13
6.1 BLE_Controller.h	13
6.2 Sources/drivers/MFRC522.h File Reference	13
6.2.1 Macro Definition Documentation	17
6.2.1.1 MIFARE_KEY_SIZE	17
6.2.1.2 NUM_BLOCKS_PER_SECTOR	17
6.2.1.3 NUM_SECTORE_MIFARE_1K	17
6.2.2 Enumeration Type Documentation	17
6.2.2.1 bitFraming_t	17
6.2.2.2 uidSize_t	18
6.2.3 Function Documentation	18
6.2.3.1 vMFRC522_GetAndPrintFifoBuf()	18
6.2.3.2 vMifare_PrintKey()	18
6.2.3.3 vMifare_PrintUID()	19

6.2.3.4 xMFRC522_AntennaOn()	19
6.2.3.5 xMFRC522_CalculateCRC()	19
6.2.3.6 xMFRC522_ClrRegBitMask()	20
6.2.3.7 xMFRC522_CommWithMifare()	20
6.2.3.8 xMFRC522_Init()	20
6.2.3.9 xMFRC522_IsCardPresent()	21
6.2.3.10 xMFRC522_MF_Authent()	21
6.2.3.11 xMFRC522_ReadRegister()	22
6.2.3.12 xMFRC522_ReadRegisterArr()	22
6.2.3.13 xMFRC522_Reset()	22
6.2.3.14 xMFRC522_SelfTest()	23
6.2.3.15 xMFRC522_SetRegBitMask()	23
6.2.3.16 xMFRC522_Transcieve()	23
6.2.3.17 xMFRC522_WriteRegister()	24
6.2.3.18 xMFRC522_WriteRegisterArr()	24
6.2.3.19 xMifare_Authenticate()	25
6.2.3.20 xMifare_GetKeyData()	25
6.2.3.21 xMifare_ReadKeyBlock()	26
6.2.3.22 xMifare_ReadUID()	26
6.2.3.23 xMifare_WriteKey()	26
6.2.3.24 xMifare_WriteKeyBlock()	27
6.3 MFRC522.h	27
Index	31

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

MFRC522 Register Addresses	7
MFRC522 Commands	8

Chapter 2

Class Index

2.1 Class List

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

Mifare1kKey_t	11
UniqueIdentifier_t	12

Chapter 3

File Index

3.1 File List

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

Sources/bluetooth_contr/ BLE_Controller.h	13
Sources/drivers/ MFRC522.h	13

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 Mifare1kKey_t Struct Reference

Public Attributes

- [UniqueIdentifier_t](#) uid
- uint8_t [blockKey](#) [6]
- uint8_t [keyData](#) [16][64]

5.1.1 Member Data Documentation

5.1.1.1 blockKey

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

Block key.

5.1.1.2 keyData

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

Key data for 1k card (1024 bytes).

5.1.1.3 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](#)

5.2 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](#)

5.2.1 Member Data Documentation

5.2.1.1 bccByte

[uint8_t UniqueIdentifier_t::bccByte](#)

BCC byte.

5.2.1.2 doubleSizeUidData

[doubleSizeUID_t UniqueIdentifier_t::doubleSizeUidData](#)

Data for a double size UID.

5.2.1.3 sakByte

[uint8_t UniqueIdentifier_t::sakByte](#)

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

5.2.1.4 singleSizeUidData

[singleSizeUID_t UniqueIdentifier_t::singleSizeUidData](#)

Data for a single size UID.

5.2.1.5 trippleSizeUidData

[trippleSizeUID_t UniqueIdentifier_t::trippleSizeUidData](#)

Data for a triple size UID.

5.2.1.6 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 #include <stdio.h>
00011 #include <stdlib.h>
00012 #include <string.h>
00013 #include <inttypes.h>
00014 #include "freertos/FreeRTOS.h"
00015 #include "freertos/task.h"
00016 #include "freertos/event_groups.h"
00017 #include "esp_system.h"
00018 #include "esp_log.h"
00019 #include "nvs_flash.h"
00020 #include "esp_bt.h"
00021
00022 #include "esp_gap_ble_api.h"
00023 #include "esp_gatts_api.h"
00024 #include "esp_bt_defs.h"
00025 #include "esp_bt_main.h"
00026 #include "esp_gatt_common_api.h"
00027
00028 #include "sdkconfig.h"
00029
00040 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);
00041
00051 void example_exec_write_event_env(prepare_type_env_t *prepare_write_env, esp_ble_gatts_cb_param_t
    *param);
00052
00061 static void gap_event_handler(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *param);
00062
00073 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);
00074
00084 void example_exec_write_event_env(prepare_type_env_t *prepare_write_env, esp_ble_gatts_cb_param_t
    *param);
00085
00096 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);
00097
00108 static void gatts_event_handler(esp_gatts_cb_event_t event, esp_gatt_if_t gatts_if,
    esp_ble_gatts_cb_param_t *param);
00109
00115 void BLE_init(void);
```

6.2 Sources/drivers/MFRC522.h File Reference

```
#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>
```

Classes

- struct [UniquelIdentifier_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 registerAddress, 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_Transceive` (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 Macro Definition Documentation

6.2.1.1 MIFARE_KEY_SIZE

```
#define MIFARE_KEY_SIZE 6
```

Size of MIFARE key.

6.2.1.2 NUM_BLOCKS_PER_SECTOR

```
#define NUM_BLOCKS_PER_SECTOR 4
```

Number of blocks per sector.

6.2.1.3 NUM_SECTORE_MIFARE_1K

```
#define NUM_SECTORE_MIFARE_1K 16
```

Number of sectors in MIFARE 1k.

6.2.2 Enumeration Type Documentation

6.2.2.1 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

6.2.2.2 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

6.2.3 Function Documentation

6.2.3.1 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

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

6.2.3.2 vMifare_PrintKey()

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

Prints the key data to the console.

Parameters

<i>key</i>	Pointer to the Mifare1kKey_t structure containing the key data.
------------	---

6.2.3.3 vMifare_PrintUID()

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

Prints the UID information to the console.

Parameters

<i>UID</i>	Pointer to the UniqueIdentifier_t structure containing the UID.
------------	---

6.2.3.4 xMFRC522_AntennaOn()

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

Turns on the antenna of the MFRC522.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
------------------	-----------------------------------

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.5 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.6 xMFRC522_ClrRegBitMask()

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

Clears the specified bits in the register of the MFRC522.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
<i>registerAddress</i>	The address of the register to modify.
<i>mask</i>	The bitmask of the bits to clear.

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.7 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.8 xMFRC522_Init()

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

Initializes the MFRC522 RFID module.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
------------------	-----------------------------------

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.9 xMFRC522_IsCardPresent()

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

Checks if a card is present.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
------------------	-----------------------------------

Returns

True if a card is present, false otherwise.

6.2.3.10 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.11 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.12 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.13 xMFRC522_Reset()

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

Resets the MFRC522.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
------------------	-----------------------------------

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.14 xMFRC522_SelfTest()

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

Performs a self-test of the MFRC522.

Parameters

<i>spiHandle</i>	Pointer to the SPI device handle.
------------------	-----------------------------------

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.15 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.16 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.17 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

<i>spiHandle</i>	Pointer to the SPI device handle.
<i>registerAddress</i>	The address of the register to write.
<i>value</i>	The value to write to the register.

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.18 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.19 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.20 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

<i>spiHandle</i>	Pointer to the SPI device handle.
<i>UID</i>	Pointer to the UniqueIdentifier_t structure containing the UID.

Returns

Pointer to the [Mifare1kKey_t](#) structure containing the key data, or NULL on failure.

6.2.3.21 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.22 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

<i>spiHandle</i>	Pointer to the SPI device handle.
<i>uidSize</i>	The size of the UID.

Returns

Pointer to the [UniqueIdentifier_t](#) structure containing the UID information, or NULL on failure.

6.2.3.23 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

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

Returns

ESP_OK if successful, otherwise an error code.

6.2.3.24 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

<i>spiHandle</i>	Pointer to the SPI device handle.
<i>blockAddress</i>	The block address to write the data to.
<i>UID</i>	Pointer to the UniqueIdentifier_t structure containing the UID.
<i>data</i>	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
00005 #ifndef _MFRC522_H_
00006 #define _MFRC522_H_
00007
00008 #include "driver/spi_master.h"
00009 #include "esp_timer.h"
00010 #include <freertos/FreeRTOS.h>
00011 #include <freertos/task.h>
00012 #include <driver/gpio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include <time.h>
00016
00017 // Number of bytes in the UID. 4, 7 or 10.
00021 typedef uint8_t singleSizeUID_t[4];
00022
00026 typedef uint8_t doubleSizeUID_t[7];
00027
00031 typedef uint8_t trippleSizeUID_t[10];
00032
00036 typedef enum {
00037     sevenBit = 0x07,
00038     eightBit = 0x08,
```

```

00039 } bitFraming_t;
00040
00044 typedef enum {
00045     fourBytesSingle = 4,
00046     sevenBytesDouble = 7,
00047     tenBytesTripple = 10,
00048 } uidSize_t;
00049
00053 typedef struct {
00054     uidSize_t uidSize;
00055     union {
00056         singleSizeUID_t singleSizeUidData;
00057         doubleSizeUID_t doubleSizeUidData;
00058         trippleSizeUID_t trippleSizeUidData;
00059     } uidData;
00060     uint8_t sakByte;
00061     uint8_t bccByte;
00062 } UniqueIdentifier_t;
00063
00067 typedef struct {
00068     UniqueIdentifier_t uid;
00069     uint8_t blockKey[6];
00070     uint8_t keyData[16][64];
00071 } Mifare1kKey_t;
00072
00073 // Definitions for some constants
00074 #define NUM_SECTORE_MIFARE_1K 16
00075 #define NUM_BLOCKS_PER_SECTOR 4
00076 #define MIFARE_KEY_SIZE 6
00082 #define MFRC522_REG_RESERVED00 0x00 « 1
00083 #define MFRC522_REG_COMMAND 0x01 « 1
00084 #define MFRC522_REG_COMIEN 0x02 « 1
00085 #define MFRC522_REG_DIVIEN 0x03 « 1
00086 #define MFRC522_REG_COMIRQ 0x04 « 1
00087 #define MFRC522_REG_DIVIRQ 0x05 « 1
00088 #define MFRC522_REG_ERROR 0x06 « 1
00089 #define MFRC522_REG_STATUS1 0x07 « 1
00090 #define MFRC522_REG_STATUS2 0x08 « 1
00091 #define MFRC522_REG_FIFO_DATA 0x09 « 1
00092 #define MFRC522_REG_FIFO_LEVEL 0x0A « 1
00093 #define MFRC522_REG_WATER_LEVEL 0x0B « 1
00094 #define MFRC522_REG_CONTROL 0x0C « 1
00095 #define MFRC522_REG_BIT_FRAMING 0x0D « 1
00096 #define MFRC522_REG_COLL 0x0E « 1
00097 #define MFRC522_REG_RESERVED01 0x0F « 1
00098
00099 // Page 1: Command
00100 #define MFRC522_REG_RESERVED10 0x10 « 1
00101 #define MFRC522_REG_MODE 0x11 « 1
00102 #define MFRC522_REG_TX_MODE 0x12 « 1
00103 #define MFRC522_REG_RX_MODE 0x13 « 1
00104 #define MFRC522_REG_TX_CONTROL 0x14 « 1
00105 #define MFRC522_REG_TX_AUTO 0x15 « 1
00106 #define MFRC522_REG_TX_SEL 0x16 « 1
00107 #define MFRC522_REG_RX_SEL 0x17 « 1
00108 #define MFRC522_REG_RX_THRESHOLD 0x18 « 1
00109 #define MFRC522_REG_DEMOD 0x19 « 1
00110 #define MFRC522_REG_RESERVED11 0x1A « 1
00111 #define MFRC522_REG_RESERVED12 0x1B « 1
00112 #define MFRC522_REG_MIFARE 0x1C « 1
00113 #define MFRC522_REG_RESERVED13 0x1D « 1
00114 #define MFRC522_REG_RESERVED14 0x1E « 1
00115 #define MFRC522_REG_SERIALSPEED 0x1F « 1
00116
00117 // Page 2: Configuration
00118 #define MFRC522_REG_RESERVED20 0x20 « 1
00119 #define MFRC522_REG_CRC_RESULT_M 0x21 « 1
00120 #define MFRC522_REG_CRC_RESULT_L 0x22 « 1
00121 #define MFRC522_REG_RESERVED21 0x23 « 1
00122 #define MFRC522_REG_MOD_WIDTH 0x24 « 1
00123 #define MFRC522_REG_RESERVED22 0x25 « 1
00124 #define MFRC522_REG_RF_CFG 0x26 « 1
00125 #define MFRC522_REG_GS_N 0x27 « 1
00126 #define MFRC522_REG_CWGS_PREG 0x28 « 1
00127 #define MFRC522_REG_MOD_GS_PREG 0x29 « 1
00128 #define MFRC522_REG_T_MODE 0x2A « 1
00129 #define MFRC522_REG_T_PRESCALER 0x2B « 1
00130 #define MFRC522_REG_T_RELOAD_H 0x2C « 1
00131 #define MFRC522_REG_T_RELOAD_L 0x2D « 1
00132 #define MFRC522_REG_T_COUNTER_VALUE_H 0x2E « 1
00133 #define MFRC522_REG_T_COUNTER_VALUE_L 0x2F « 1
00134
00135 // Page 3: Test
00136 #define MFRC522_REG_RESERVED30 0x30 « 1
00137 #define MFRC522_REG_TEST_SEL1 0x31 « 1
00138 #define MFRC522_REG_TEST_SEL2 0x32 « 1
00139 #define MFRC522_REG_TEST_PIN_EN 0x33 « 1

```



```

00140 #define MFRC522_REG_TEST_PIN_VALUE      0x34 « 1
00141 #define MFRC522_REG_TEST_BUS             0x35 « 1
00142 #define MFRC522_REG_AUTO_TEST            0x36 « 1
00143 #define MFRC522_REG_VERSION               0x37 « 1
00144 #define MFRC522_REG_ANALOG_TEST          0x38 « 1
00145 #define MFRC522_REG_TEST_DAC1            0x39 « 1
00146 #define MFRC522_REG_TEST_DAC2            0x3A « 1
00147 #define MFRC522_REG_TEST_ADC              0x3B « 1
00148 #define MFRC522_REG_RESERVED31           0x3C « 1
00149 #define MFRC522_REG_RESERVED32           0x3D « 1
00150 #define MFRC522_REG_RESERVED33           0x3E « 1
00151 #define MFRC522_REG_RESERVED34           0x3F « 1
00152 // Commands sent to the PICC.
00153 typedef enum {
00154     PICC_CMD_REQA          = 0x26,
00155     PICC_CMD_WUPA          = 0x52,
00156     PICC_CMD_CT            = 0x88,
00157     PICC_CMD_SEL_CL1       = 0x93,
00158     PICC_CMD_SEL_CL2       = 0x95,
00159     PICC_CMD_SEL_CL3       = 0x97,
00160     PICC_CMD_HLTA          = 0x50,
00161     PICC_CMD_RATS          = 0xE0,
00162     PICC_CMD_MF_AUTH_KEY_A = 0x60,
00163     PICC_CMD_MF_AUTH_KEY_B = 0x61,
00164     PICC_CMD_MF_READ        = 0x30,
00165     PICC_CMD_MF_WRITE       = 0xA0,
00166     PICC_CMD_MF_DECREMENT   = 0xC0,
00167     PICC_CMD_MF_INCREMENT   = 0xC1,
00168     PICC_CMD_MF_RESTORE     = 0xC2,
00169     PICC_CMD_MF_TRANSFER    = 0xB0,
00170     PICC_CMD_UL_WRITE       = 0xA2,
00171     MFRC522_MIFARE_ACK     = 0x0A,
00172 } piccCmds_t;
00173
00174 // MFRC522's commands for the PCD.
00175 #define PCD_CMD_IDLE      0x00 // NO action; cancels current command execution
00176 #define PCD_CMD_MEM       0x01 // Stores 25 bytes into the internal buffer
00177 #define PCD_CMD_GEN_RANDOM 0x02 // Generates a 10-byte random ID number
00178 #define PCD_CMD_CALC_CRC  0x03 // Activates the CRC coprocessor or performs a self-test
00179 #define PCD_CMD_TRANSMIT  0x04 // Transmits data from the FIFO buffer
00180 #define PCD_CMD_NO_CMD_CHANGE 0x07 // Can be used to modify the CommandReg register bits
00181 // without affecting the command, if any, currently being executed
00182 #define PCD_CMD_RECEIVE    0x08 // Activates the receiver circuits
00183 #define PCD_CMD_TRANSCEIVE 0x0C // Transmits data from FIFO buffer to antenna and
00184 // automatically activates the receiver after transmission
00185 #define PCD_CMD_MF_AUTHENT 0x0E // Performs the MIFARE standard authentication as a reader
00186 #define PCD_CMD_SOFT_RESET 0x0F // Resets the MFRC522
00187
00188 esp_err_t xMFRC522_WriteRegister(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00189 value);
00190
00191 esp_err_t xMFRC522_ReadRegister(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00192 *data);
00193
00194 esp_err_t xMFRC522_ReadRegisterArr(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00195 *dataArr, uint8_t dataSize);
00196
00197 esp_err_t xMFRC522_WriteRegisterArr(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00198 *dataArr, uint8_t dataSize);
00199
00200 esp_err_t xMFRC522_Init(spi_device_handle_t *spiHandle, uint8_t rstPin);
00201
00202 esp_err_t xMFRC522_ClrRegBitMask(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00203 mask);
00204
00205 bool xMFRC522_IsCardPresent(spi_device_handle_t *spiHandle);
00206
00207 esp_err_t xMFRC522_AntennaOn(spi_device_handle_t *spiHandle);
00208
00209 esp_err_t xMFRC522_SelfTest(spi_device_handle_t *spiHandle, uint8_t rstPin);
00210
00211 esp_err_t xMFRC522_Reset(spi_device_handle_t *spiHandle);
00212
00213 esp_err_t xMFRC522_Transceive(spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf,
00214 uint8_t bufSize, bitFraming_t bitFrame);
00215
00216 esp_err_t xMFRC522_MF_Authent(spi_device_handle_t *spiHandle, uint8_t waitIrq, uint8_t *cmdBuf,
00217 uint8_t bufSize, bitFraming_t bitFrame);
00218
00219 esp_err_t xMFRC522_CommWithMifare(uint8_t cmd, spi_device_handle_t *spiHandle, uint8_t waitIrq,
00220 uint8_t *cmdBuf, uint8_t bufSize, bitFraming_t bitFrame);
00221
00222 esp_err_t xMFRC522_SetRegBitMask(spi_device_handle_t *spiHandle, uint8_t registerAddress, uint8_t
00223 mask);
00224
00225 esp_err_t xMFRC522_CalculateCRC(spi_device_handle_t *spiHandle, uint8_t *buf, uint8_t bufLen, uint8_t
00226 resultBuf[2]);

```

```
00343
00351 UniqueIdentifier_t *xMifare_ReadUID(spi_device_handle_t *spiHandle, uidSize_t uidSize);
00352
00360 Mifare1kKey_t *xMifare_GetKeyData(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID);
00361
00370 esp_err_t xMifare_WriteKey(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID, uint8_t
data[45][16]);
00371
00381 esp_err_t xMifare_WriteKeyBlock(spi_device_handle_t *spiHandle, uint8_t blockAddress,
UniqueIdentifier_t *UID, uint8_t data[16]);
00382
00390 static esp_err_t xMifare_SetSakByte(spi_device_handle_t *spiHandle, UniqueIdentifier_t *UID);
00391
00400 esp_err_t xMifare_ReadKeyBlock(spi_device_handle_t *spiHandle, uint8_t blockAddress,
UniqueIdentifier_t *UID);
00401
00410 static bool xPrv_Mifare_BlockCheckChar(uint8_t *bufData, uint8_t bufSize, UniqueIdentifier_t *UID);
00411
00422 esp_err_t xMifare_Authenticate(spi_device_handle_t *spiHandle, uint8_t cmd, uint8_t blockAddress,
uint8_t *key, UniqueIdentifier_t *UID);
00423
00429 void vMifare_PrintUID(UniqueIdentifier_t *UID);
00430
00438 void vMFRC522_GetAndPrintFifoBuf(spi_device_handle_t *spiHandle, uint8_t *fifoBuf, bool print);
00439
00445 void vMifare_PrintKey(Mifare1kKey_t *key);
00446
00447
00448 #endif // _MFRC522_H_
```

Index

- bccByte
 - UniquelIdentifier_t, [12](#)
- bitFraming_t
 - MFRC522.h, [17](#)
- blockKey
 - Mifare1kKey_t, [11](#)
- doubleSizeUidData
 - UniquelIdentifier_t, [12](#)
- eightBit
 - MFRC522.h, [18](#)
- fourBytesSingle
 - MFRC522.h, [18](#)
- keyData
 - Mifare1kKey_t, [11](#)
- MFRC522 Commands, [8](#)
- MFRC522 Register Addresses, [7](#)
- MFRC522.h
 - bitFraming_t, [17](#)
 - eightBit, [18](#)
 - fourBytesSingle, [18](#)
 - MIFARE_KEY_SIZE, [17](#)
 - NUM_BLOCKS_PER_SECTOR, [17](#)
 - NUM_SECTORE_MIFARE_1K, [17](#)
 - sevenBit, [18](#)
 - sevenBytesDouble, [18](#)
 - tenBytesTripple, [18](#)
 - uidSize_t, [18](#)
 - vMFRC522_GetAndPrintFifoBuf, [18](#)
 - vMifare_PrintKey, [18](#)
 - vMifare_PrintUID, [18](#)
 - xMFRC522_AntennaOn, [19](#)
 - xMFRC522_CalculateCRC, [19](#)
 - xMFRC522_ClrRegBitMask, [19](#)
 - xMFRC522_CommWithMifare, [20](#)
 - xMFRC522_Init, [20](#)
 - xMFRC522_IsCardPresent, [21](#)
 - xMFRC522_MF_Authent, [21](#)
 - xMFRC522_ReadRegister, [21](#)
 - xMFRC522_ReadRegisterArr, [22](#)
 - xMFRC522_Reset, [22](#)
 - xMFRC522_SelfTest, [23](#)
 - xMFRC522_SetRegBitMask, [23](#)
 - xMFRC522_Transcieve, [23](#)
 - xMFRC522_WriteRegister, [24](#)
 - xMFRC522_WriteRegisterArr, [24](#)
 - xMifare_Authenticate, [25](#)
 - xMifare_GetKeyData, [25](#)
 - xMifare_ReadKeyBlock, [26](#)
 - xMifare_ReadUID, [26](#)
 - xMifare_WriteKey, [26](#)
 - xMifare_WriteKeyBlock, [27](#)
- Mifare1kKey_t, [11](#)
 - blockKey, [11](#)
 - keyData, [11](#)
 - uid, [11](#)
- MIFARE_KEY_SIZE
 - MFRC522.h, [17](#)
- NUM_BLOCKS_PER_SECTOR
 - MFRC522.h, [17](#)
- NUM_SECTORE_MIFARE_1K
 - MFRC522.h, [17](#)
- sakByte
 - UniquelIdentifier_t, [12](#)
- sevenBit
 - MFRC522.h, [18](#)
- sevenBytesDouble
 - MFRC522.h, [18](#)
- singleSizeUidData
 - UniquelIdentifier_t, [12](#)
- Sources/bluetooth_contr/BLE_Controller.h, [13](#)
- Sources/drivers/MFRC522.h, [13](#), [27](#)
- tenBytesTripple
 - MFRC522.h, [18](#)
- tripplesizeUidData
 - UniquelIdentifier_t, [12](#)
- uid
 - Mifare1kKey_t, [11](#)
- uidSize
 - UniquelIdentifier_t, [12](#)
- uidSize_t
 - MFRC522.h, [18](#)
- UniquelIdentifier_t, [12](#)
 - bccByte, [12](#)
 - doubleSizeUidData, [12](#)
 - sakByte, [12](#)
 - singleSizeUidData, [12](#)
 - tripplesizeUidData, [12](#)
 - uidSize, [12](#)
- vMFRC522_GetAndPrintFifoBuf
 - MFRC522.h, [18](#)
- vMifare_PrintKey
 - MFRC522.h, [18](#)

vMifare_PrintUID
MFRC522.h, [18](#)

MFRC522_AntennaOn
MFRC522.h, [19](#)

MFRC522_CalculateCRC
MFRC522.h, [19](#)

MFRC522_ClrRegBitMask
MFRC522.h, [19](#)

MFRC522_CommWithMifare
MFRC522.h, [20](#)

MFRC522_Init
MFRC522.h, [20](#)

MFRC522_IsCardPresent
MFRC522.h, [21](#)

MFRC522_MF_Authent
MFRC522.h, [21](#)

MFRC522_ReadRegister
MFRC522.h, [21](#)

MFRC522_ReadRegisterArr
MFRC522.h, [22](#)

MFRC522_Reset
MFRC522.h, [22](#)

MFRC522_SelfTest
MFRC522.h, [23](#)

MFRC522_SetRegBitMask
MFRC522.h, [23](#)

MFRC522_Transcieve
MFRC522.h, [23](#)

MFRC522_WriteRegister
MFRC522.h, [24](#)

MFRC522_WriteRegisterArr
MFRC522.h, [24](#)

Mifare_Authenticate
MFRC522.h, [25](#)

Mifare_GetKeyData
MFRC522.h, [25](#)

Mifare_ReadKeyBlock
MFRC522.h, [26](#)

Mifare_ReadUID
MFRC522.h, [26](#)

Mifare_WriteKey
MFRC522.h, [26](#)

Mifare_WriteKeyBlock
MFRC522.h, [27](#)