README: RC522 RFID Driver for STM32F446RETx

Sara F. Munoz[1] and Iker G. Barrera[2]

[3]Department of Electronic Systems, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, Mexico.

Contributing authors: {al301618,al307630}@edu.uaa.mx;

Abstract

This document describes the usage of the RC522 RFID driver implemented on the STM32F446RETx microcontroller via SPI, programmed in the Keil IDE. The driver provides initialization routines, authentication, data reading and writing functions, and a self-test mechanism.

Keywords: RC522, RFID, STM32F446RETx, Usage.

1 Introduction

The RC522 RFID driver for the STM32F446RETx microcontroller provides a structured interface for communication with RFID cards using the SPI protocol. Developed in the Keil IDE, the driver includes initialization routines, authentication mechanisms, and functions for reading and writing data blocks. Its design emphasizes reliability, modularity, and ease of integration with higher-level applications, such as mobile interfaces for card management. This documentation outlines the usage of the driver, including initialization steps, available functions, expected parameters, return values, error conditions, and example code for practical implementation.

2 Initialization

Before using the driver, the following steps must be performed:

1. Configure the system clock using `SystemClock_Config()`.
2. Initialize GPIO, SPI, and USART peripherals with `confGPIO()`, `confSPI()`, and `confUSART()`.
3. Reset and initialize the RC522 module:

```
RC522_ResetLow();
delay_ms(50);
```

```
    RC522_ResetHigh();
    delay_ms(100);
    RC522_Init();
    RC522_SetBitMask(TxControlReg, 0x03); // Enable antenna
```

3 Driver Functions

- **RC522_Init(void)** Initializes the RC522 registers. **Return:** None. **Errors:** Initialization may fail if SPI communication is not established.
- **RC522_RequestA(uint8_t *atqa, uint8_t *atqaLen)** Detects the presence of a card. **Return:** 0 on success, -1 on failure. **Errors:** Timeout or no card detected.
- **RC522_AnticollCL1(uint8_t *uid, uint8_t *uidLen)** Retrieves the UID of the card. **Return:** 0 on success, -1 on failure. **Errors:** Collision or invalid response.
- **RC522_Select(uint8_t *uid)** Selects a card with the given UID. **Return:** 0 on success, -1 on failure. **Errors:** Invalid UID or communication error.
- **MIFARE_Auth(uint8_t authMode, uint8_t blockAddr, uint8_t *key, uint8_t *uid)** Authenticates access to a block using Key A or Key B. **Return:** 0 on success, -1 on failure. **Errors:** Authentication failed, invalid key.
- **MIFARE_Read(uint8_t blockAddr, uint8_t *recvData)** Reads 16 bytes from a block. **Return:** 0 on success, -1 on failure. **Errors:** CRC error, timeout, or invalid block.
- **MIFARE_Write(uint8_t blockAddr, uint8_t *writeData)** Writes 16 bytes to a block. **Return:** 0 on success, -1 on failure. **Errors:** Write failed, timeout, or invalid block.

4 Error Conditions

- Timeout during communication.
- Authentication failure due to invalid key or UID mismatch.
- CRC errors indicating data integrity issues.
- Collision errors when multiple cards are detected simultaneously.
- Bad configuration between timers.

5 Example Usage

```
int main(void) {
    uint8_t atqa[2], atqaLen, uid[10], uidLen;
    uint8_t keyA[6] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
    uint8_t blockData[16];
    uint8_t writeData[16] = "Hello RFID World";

    SystemClock_Config();
    confGPIO();
    confUSART();
    confSPI();
    RC522_Init();
    RC522_SetBitMask(TxControlReg, 0x03);

    if(RC522_RequestA(atqa, &atqaLen) == 0) {
        if(RC522_AnticollCL1(uid, &uidLen) == 0) {
            if(RC522_Select(uid) == 0) {
                if(MIFARE_Auth(PICC_AUTHENT1A, 4, keyA, uid) == 0) {
                    if(MIFARE_Read(4, blockData) == 0) {
                        USART_SendString("Block 4 read successfully.\r\n");
                    }
                    if(MIFARE_Write(4, writeData) == 0) {
                        USART_SendString("Block 4 written successfully.\r\n");
                    }
                }
            }
        }
    }
}
```

6 Conclusion

Practical testing confirmed that the driver correctly handled authentication, reading, and writing of MIFARE Classic cards, with UART debugging.

The notes highlighted during implementation are: ensuring proper power supply and antenna connection for the RC522, using UART output for monitoring and debugging, and stopping the crypto session (`RC522_StopCrypto1()`) before re-authentication to maintain system stability.

In alignment with the references consulted, including the NXP RC522 datasheet, the STMicroelectronics STM32F446RETx reference manual, and Keil IDE documentation, the project validated the importance of structured driver design and adherence to hardware specifications.