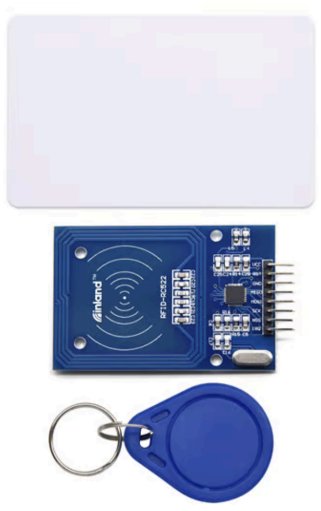
RFID



Source:https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/5531/4411_CN0090%20other%20related%20document%20%281%29.pdf

Library Overview

The RFID Library provides a compact SPI driver for the NXP MFRC522 RFID/NFC reader used in the RFID Security Access System. It handles basic reader initialization, RF field enablement, and a minimal ISO14443A command sequence to detect a card and retrieve its 4-byte UID (plus BCC) using REQA and anticollision.

Key Features

- **SPI Reader Control:** Register read/write helpers over SPI with CS and RST
- **Minimal ISO14443A Flow:** REQA (7-bit) and anticollision (CL1) to read UID
- **Robust Startup:** Hardware reset, timer configuration, antenna enable
- **Simple API:** Initialize the reader and fetch a card UID
- **Portable Helpers:** Uses project SPI exchange function

Use Cases

- Detecting an RFID card presented at the reader
- Reading the card UID to validate access permissions
- Gating other subsystems (lock, LEDs, LCD messages) on successful UID read

Function Summary

Function	Description
rfid_getcard_uid(uint_	Description: Attempts to detect a card and read its 4-byte UID plus BCC

8_t*buffer)	Input: buffer - pointer to a buffer of at least 5 bytes to store UID+BCC
	Return: 0 on success,-1 on error or if no card is detected
	Usage:
	`uint8t card_uid[5] = {0};`
	`rfid_getcard_uid(card_uid);`

rfid_init(void)	Initializes the MFRC522 reader (reset, config, and antenna on)
	Input: none
	Return: none
	Usage:
	`int main(void) {`
	`rfid_init(); // Call once during system initialization`
	`while(1) {`
	`// Main program loop`
	`}`
	`}`

Wiring Diagram

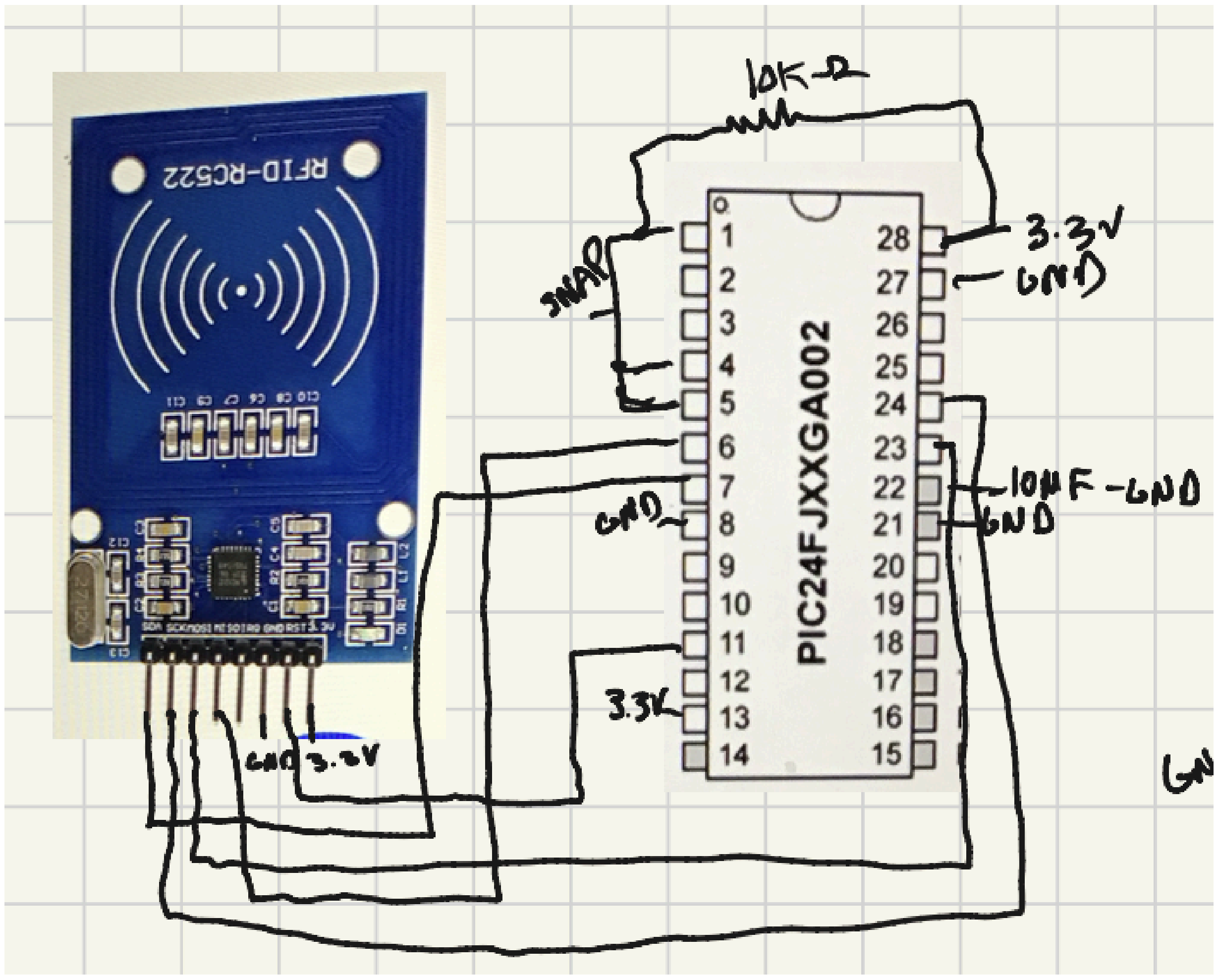


Figure 1: Configuration for the RFID diagram

Notes: RST/SS are connected to pins 11 and7 respectively. MOSI/MISO/CLK are also connected to pins 23, 6 and 24 respectively. Finally, VCC and GND are connected to the PIC24's appropriate pins.

Code

Source file

```
/*
 * File: rfid.h
 * Author: Yusuf Mahamud
 * Project: RFID Security Access System
 * Description: Implementation of the MFRC522 RFID reader driver, over SPI
 */
```

```
#define FCY 16000000UL              // Define instruction cycle frequency
#include <libpic30.h>              // Include for __delay_ms()/__delay_us()
#include "pic24.h"                 // Include for pic24_exchange_spi()
#include "xc.h"
#include "rfid.h"

#define RFID_CS LATBbits.LATB3       // Chip Select line (active LOW)
#define RFID_RST LATBbits.LATB4      // Reset line

// MFRC522 Commands
#define PCD_IDLE 0x00
#define PCD_AUTHENT 0x0E
#define PCD_TRANSCEIVE 0x0C
#define PCD_SOFTRESET 0x0F

// PICC (card) Commands
#define PICC_REQIDL 0x26
#define PICC_ANTICOLL 0x93

// MFRC522 Register addresses
#define CommandReg 0x01
#define ComIEnReg 0x02
#define DivIEnReg 0x03
#define ComIrqReg 0x04
#define DivIrqReg 0x05
#define ErrorReg 0x06
#define Status1Reg 0x07
#define Status2Reg 0x08
#define FIFODataReg 0x09
#define FIFOLevelReg 0x0A
#define WaterLevelReg 0x0B
#define ControlReg 0x0C
#define BitFramingReg 0x0D
#define CollReg 0x0E
#define ModeReg 0x11
#define TxModeReg 0x12
#define RxModeReg 0x13
#define TxControlReg 0x14
#define TxASKReg 0x15
#define TModeReg 0x2A
#define TPrescalerReg 0x2B
#define TReloadRegH 0x2C
#define TReloadRegL 0x2D
#define VersionReg 0x37
```

```
// Function: rfid_write_reg
// Description: Writes a value to an MFRC522 register over SPI
// Input: addr - register address to write
//        val - data byte to write
// Return: none
static void rfid_write_reg(uint8_t addr, uint8_t val)
{
   RFID_CS = 0;                                  // assert CS low
   pic24_exchange_spi((addr << 1) & 0x7E);          // send address (write)
   pic24_exchange_spi(val);                          // send data byte
   RFID_CS = 1;                                  // deassert CS high
}

// Function: rfid_read_reg
// Description: Reads a value from an MFRC522 register over SPI
// Input: addr - register address to read
// Return: 8-bit value read from the register
static uint8_t rfid_read_reg(uint8_t addr)
{
   uint8_t val;

   RFID_CS = 0;                                  // assert CS low
   pic24_exchange_spi(((addr << 1) & 0x7E) | 0x80); // send address (read)
   val = pic24_exchange_spi(0x00);                  // dummy write, read data
   RFID_CS = 1;                                  // deassert CS high

   return val;                                 // return value read
}

// Function: rfid_bitmask_set
// Description: Sets bits in a register per mask
// Input: reg - register address
//        mask - bit mask to set
// Return: none
static void rfid_bitmask_set(uint8_t reg, uint8_t mask)
{
   rfid_write_reg(reg, rfid_read_reg(reg) | mask);
}

// Function: rfid_bitmask_clear
// Description: Clears bits in a register per mask
// Input: reg - register address
//        mask - bit mask to clear
// Return: none
static void rfid_bitmask_clear(uint8_t reg, uint8_t mask)
{
    rfid_write_reg(reg, rfid_read_reg(reg) & (~mask));
}

// Function: rfid_communicate
// Description: Exchanges data with a card via the MFRC522 FIFO using a PCD
//              command (e.g., TRANSCEIVE)
// Input: command  - PCD command to execute (e.g., PCD_TRANSCEIVE)
//        sendData - pointer to bytes to send
//        sendLen  - number of bytes to send
//        backData - pointer to buffer for received data (can be NULL)
//        backLen  - pointer to bit-length of response (can be NULL)
// Return: 0 on success, -1 on failure/timeout
static int rfid_communicate(uint8_t command,
                            uint8_t *sendData,
                            uint8_t sendLen,
                            uint8_t *backData,
                            uint8_t *backLen)
{
    uint8_t status = MI_ERR;                    // default to error
    uint8_t irqEn = 0x00;                       // IRQ enable bits
    uint8_t waitIRq = 0x00;                     // IRQ wait mask
    uint8_t lastBits;                           // last bits count
    uint8_t n;                                  // IRQ/status temp
    uint16_t i;                                 // timeout counter

    // set IRQ enables and the IRQ to wait for based on command type
    switch (command)
    {
        case PCD_AUTHENT:
            irqEn = 0x12;                       // enable AUTH IRQ
            waitIRq = 0x10;                     // wait for AUTH
            break;
        case PCD_TRANSCEIVE:
            irqEn = 0x77;                       // enable RX/TX IRQs
            waitIRq = 0x30;                     // wait for RX done
            break;
        default:
            break;
    }

    rfid_write_reg(ComIEnReg, irqEn | 0x80);          // enable IRQs, global
    rfid_write_reg(ComIrqReg, 0x7F);                  // clear IRQ flags
    rfid_bitmask_set(FIFOLevelReg, 0x80);             // flush FIFO
    rfid_write_reg(CommandReg, PCD_IDLE);             // stop active cmd

    // write outgoing data to the FIFO
```

```
for (i = 0; i < sendLen; i++)
{
   rfid_write_reg(FIFODataReg, sendData[i]);
}

// execute command
rfid_write_reg(CommandReg, command);
if (command == PCD_TRANSCEIVE)
{
   rfid_bitmask_set(BitFramingReg, 0x80);         // StartSend = 1
}

// wait for completion or timeout (~25ms total)
for (i = 2000; i > 0; i--)
{
   n = rfid_read_reg(ComIrqReg);
   if (n & waitIRq)
   {
      break;                                   // operation done
   }
   if (n & 0x01)
   {
      return -1;
   }
   __delay_us(15);                             // small delay step
}

if (i == 0)
{
   return -1;
}

rfid_bitmask_clear(BitFramingReg, 0x80);         // clear StartSend

// check for errors (buffer overflow, collision, parity, protocol)
if (rfid_read_reg(ErrorReg) & 0x1B)
{
   return -1;
}

status = 0;

// read response if buffers provided
if (backData && backLen)
{
   n = rfid_read_reg(FIFOLevelReg);              // bytes in FIFO
```

```
        lastBits = rfid_read_reg(ControlReg) & 0x07;    // valid bits in last

        if (lastBits)
        {
           *backLen = (n - 1) * 8 + lastBits;          // bits count
        }
        else
        {
           *backLen = n * 8;                          // full bytes
        }

        if (n == 0) n = 1;                            // guard minimum
        if (n > 16) n = 16;                           // limit read size

        for (i = 0; i < n; i++)
        {
           backData[i] = rfid_read_reg(FIFODataReg);   // read FIFO
        }
    }

    return status;
}

// Function: rfid_init
// Description: Initializes the MFRC522 reader (reset, config, and antenna on)
// Return: none
void rfid_init(void)
{
    // perform a reset pulse sequence on the MFRC522
    RFID_RST = 1;
    __delay_ms(10);
    RFID_RST = 0;
    __delay_ms(10);
    RFID_RST = 1;
    __delay_ms(50);                                 // wait for boot

    // basic mode configuration
    rfid_write_reg(TxModeReg, 0x00);
    rfid_write_reg(RxModeReg, 0x00);
    rfid_write_reg(TxASKReg, 0x40);                    // 100% ASK
    rfid_write_reg(ModeReg, 0x3D);                     // CRC init 0x6363

    // configure timer (recommended defaults)
    rfid_write_reg(TModeReg, 0x8D);                    // Tauto=1
    rfid_write_reg(TPrescalerReg, 0x3E);               // prescaler
    rfid_write_reg(TReloadRegL, 30);
    rfid_write_reg(TReloadRegH, 0);

    // enable RF field (antenna on)
    rfid_bitmask_set(TxControlReg, 0x03);
}

// Function: rfid_get_card_uid
// Description: Attempts to detect a card and read its 4-byte UID plus BCC
// Input: buffer - pointer to a buffer of at least 5 bytes to store UID+BCC
// Return: 0 on success, -1 on error or if no card is detected
int rfid_get_card_uid(uint8_t *buffer)
{
    uint8_t cmd_buffer[2];                    // command buffer
    uint8_t len;                              // response length

    // step 1: request (find card), expect 2-byte ATQA (16 bits)
    rfid_write_reg(BitFramingReg, 0x07);          // TxLastBits = 7
    cmd_buffer[0] = PICC_REQIDL;                  // REQA (7 bits)

    if (rfid_communicate(PCD_TRANSCEIVE,
                         cmd_buffer,
                         1,
                         cmd_buffer,
                         &len) != 0 ||
        len != 0x10)
    {
        return -1;                            // no card or error
    }

    // step 2: anticollision to read UID (expect 5 bytes: 4 UID + 1 BCC)
    rfid_write_reg(BitFramingReg, 0x00);          // full bytes
    cmd_buffer[0] = PICC_ANTICOLL;                 // anticollision
    cmd_buffer[1] = 0x20;                      // NVB = 0x20

    if (rfid_communicate(PCD_TRANSCEIVE, cmd_buffer, 2, buffer, &len) != 0)
    {
        return -1;                            // failed to get UID
    }

    return 0;
}
```

Header file

```c
/*
 * File: rfid.h
 * Author: Yusuf Mahamud
 * Project: RFID Security Access System
 * Description: Header file for the MFRC522 RFID reader module
 */

#ifndef RFID_H
#define RFID_H

#include <stdint.h>

void rfid_init(void);
int rfid_get_card_uid(uint8_t *buffer);

#endif /* RFID_H */
```