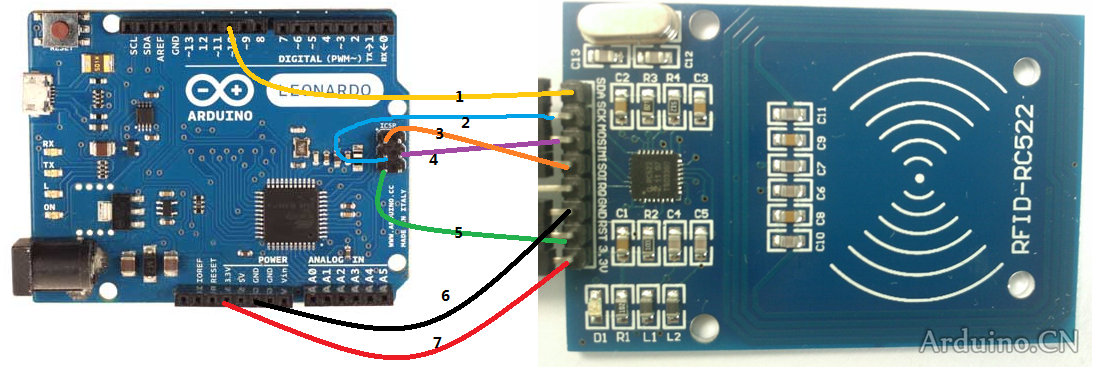
# RFID-RC522 与Arduino的连接

# 通过Leonardo来控制

BY 箱子 交大创客空间



连接图

连接方式

RST 🡨------------------------------🡪 芯片13脚，即图中5号线

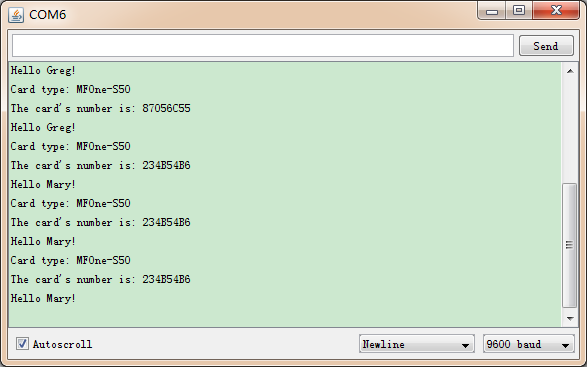
SDA 🡨------------------------------🡪 图中1号线，连接10号端口，控制片选

SCK 🡨------------------------------🡪 图中2号线，连接左中插针，时钟信号

MOSI 🡨------------------------------🡪 图中4号线，连接右中插针，信号脚

MISO 🡨------------------------------🡪 图中3号线，连接左上插针，信号脚

简单分析，当正常工作时，把卡放在射频区域，就能读卡，识别到卡上的卡号(ID)。与程序中的数据库进行对比就能够判断该卡能否识别，进而做出其他控制。



相关程序

#include <SPI.h>

#define uchar unsigned char

#define uint unsigned int

//data array maxium length

#define MAX\_LEN 16

/////////////////////////////////////////////////////////////////////

//set the pin

/////////////////////////////////////////////////////////////////////

const int chipSelectPin = 10;//cs control

const int NRSTPD = 13;//Reset

//MF522 command bits

#define PCD\_IDLE 0x00 //NO action; cancel current commands

#define PCD\_AUTHENT 0x0E //verify password key

#define PCD\_RECEIVE 0x08 //receive data

#define PCD\_TRANSMIT 0x04 //send data

#define PCD\_TRANSCEIVE 0x0C //send and receive data

#define PCD\_RESETPHASE 0x0F //reset

#define PCD\_CALCCRC 0x03 //CRC check and caculation

//Mifare\_One card command bits

#define PICC\_REQIDL 0x26 //Search the cards that not into sleep mode in the antenna area

#define PICC\_REQALL 0x52 //Search all the cards in the antenna area

#define PICC\_ANTICOLL 0x93 //prevent conflict

#define PICC\_SElECTTAG 0x93 //select card

#define PICC\_AUTHENT1A 0x60 //verify A password key

#define PICC\_AUTHENT1B 0x61 //verify B password key

#define PICC\_READ 0x30 //read

#define PICC\_WRITE 0xA0 //write

#define PICC\_DECREMENT 0xC0 //deduct value

#define PICC\_INCREMENT 0xC1 //charge up value

#define PICC\_RESTORE 0xC2 //Restore data into buffer

#define PICC\_TRANSFER 0xB0 //Save data into buffer

#define PICC\_HALT 0x50 //sleep mode

//THe mistake code that return when communicate with MF522

#define MI\_OK 0

#define MI\_NOTAGERR 1

#define MI\_ERR 2

//------------------MFRC522 register ---------------

//Page 0:Command and Status

#define Reserved00 0x00

#define CommandReg 0x01

#define CommIEnReg 0x02

#define DivlEnReg 0x03

#define CommIrqReg 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 Reserved01 0x0F

//Page 1:Command

#define Reserved10 0x10

#define ModeReg 0x11

#define TxModeReg 0x12

#define RxModeReg 0x13

#define TxControlReg 0x14

#define TxAutoReg 0x15

#define TxSelReg 0x16

#define RxSelReg 0x17

#define RxThresholdReg 0x18

#define DemodReg 0x19

#define Reserved11 0x1A

#define Reserved12 0x1B

#define MifareReg 0x1C

#define Reserved13 0x1D

#define Reserved14 0x1E

#define SerialSpeedReg 0x1F

//Page 2:CFG

#define Reserved20 0x20

#define CRCResultRegM 0x21

#define CRCResultRegL 0x22

#define Reserved21 0x23

#define ModWidthReg 0x24

#define Reserved22 0x25

#define RFCfgReg 0x26

#define GsNReg 0x27

#define CWGsPReg 0x28

#define ModGsPReg 0x29

#define TModeReg 0x2A

#define TPrescalerReg 0x2B

#define TReloadRegH 0x2C

#define TReloadRegL 0x2D

#define TCounterValueRegH 0x2E

#define TCounterValueRegL 0x2F

//Page 3:TestRegister

#define Reserved30 0x30

#define TestSel1Reg 0x31

#define TestSel2Reg 0x32

#define TestPinEnReg 0x33

#define TestPinValueReg 0x34

#define TestBusReg 0x35

#define AutoTestReg 0x36

#define VersionReg 0x37

#define AnalogTestReg 0x38

#define TestDAC1Reg 0x39

#define TestDAC2Reg 0x3A

#define TestADCReg 0x3B

#define Reserved31 0x3C

#define Reserved32 0x3D

#define Reserved33 0x3E

#define Reserved34 0x3F

//-----------------------------------------------

//4 bytes Serial number of card, the 5 bytes is verfiy bytes

uchar serNum[5];

void setup()

{

Serial.begin(57600);

SPI.begin();

pinMode(chipSelectPin,OUTPUT); // Set digital pin 10 as OUTPUT to connect it to the RFID /ENABLE pin

digitalWrite(chipSelectPin, LOW); // Activate the RFID reader

pinMode(NRSTPD,OUTPUT); // Set digital pin 5 , Not Reset and Power-down

MFRC522\_Init();

}

void loop()

{

uchar status;

uchar str[MAX\_LEN];

// Search card, return card types

status = MFRC522\_Request(PICC\_REQIDL, str);

if (status != MI\_OK)

{

return;

}

// Show card type

ShowCardType(str);

//Prevent conflict, return the 4 bytes Serial number of the card

status = MFRC522\_Anticoll(str);

// str[0..3]: serial number of the card

// str[4]: XOR checksum of the SN.

if (status == MI\_OK)

{

Serial.print("The card's number is: ");

memcpy(serNum, str, 5);

ShowCardID(serNum);

// Check people associated with card ID

uchar\* id = serNum;

if( id[0]==0x23 && id[1]==0x4B && id[2]==0x54 && id[3]==0xB6 ) {

Serial.println("Hello Mary!");

} else if(id[0]==0x87 && id[1]==0x05 && id[2]==0x6C && id[3]==0x55) {

Serial.println("Hello Greg!");

}else{

Serial.println("Hello unkown guy!");

}

}

MFRC522\_Halt(); //command the card into sleep mode

delay(200);

}

/\*

\* Function：ShowCardID

\* Description：Show Card ID

\* Input parameter：ID string

\* Return：Null

\*/

void ShowCardID(uchar \*id)

{

int IDlen=4;

for(int i=0; i<IDlen; i++){

Serial.print(0x0F & (id[i]>>4), HEX);

Serial.print(0x0F & id[i],HEX);

}

Serial.println("");

}

/\*

\* Function：ShowCardType

\* Description：Show Card type

\* Input parameter：Type string

\* Return：Null

\*/

void ShowCardType(uchar\* type)

{

Serial.print("Card type: ");

if(type[0]==0x04&&type[1]==0x00)

Serial.println("MFOne-S50");

else if(type[0]==0x02&&type[1]==0x00)

Serial.println("MFOne-S70");

else if(type[0]==0x44&&type[1]==0x00)

Serial.println("MF-UltraLight");

else if(type[0]==0x08&&type[1]==0x00)

Serial.println("MF-Pro");

else if(type[0]==0x44&&type[1]==0x03)

Serial.println("MF Desire");

else

Serial.println("Unknown");

}

/\*

\* Function：Write\_MFRC5200

\* Description：write a byte data into one register of MR RC522

\* Input parameter：addr--register address；val--the value that need to write in

\* Return：Null

\*/

void Write\_MFRC522(uchar addr, uchar val)

{

digitalWrite(chipSelectPin, LOW);

//address format：0XXXXXX0

SPI.transfer((addr<<1)&0x7E);

SPI.transfer(val);

digitalWrite(chipSelectPin, HIGH);

}

/\*

\* Function：Read\_MFRC522

\* Description：read a byte data into one register of MR RC522

\* Input parameter：addr--register address

\* Return：return the read value

\*/

uchar Read\_MFRC522(uchar addr)

{

uchar val;

digitalWrite(chipSelectPin, LOW);

//address format：1XXXXXX0

SPI.transfer(((addr<<1)&0x7E) | 0x80);

val =SPI.transfer(0x00);

digitalWrite(chipSelectPin, HIGH);

return val;

}

/\*

\* Function：SetBitMask

\* Description：set RC522 register bit

\* Input parameter：reg--register address;mask--value

\* Return：null

\*/

void SetBitMask(uchar reg, uchar mask)

{

uchar tmp;

tmp = Read\_MFRC522(reg);

Write\_MFRC522(reg, tmp | mask); // set bit mask

}

/\*

\* Function：ClearBitMask

\* Description：clear RC522 register bit

\* Input parameter：reg--register address;mask--value

\* Return：null

\*/

void ClearBitMask(uchar reg, uchar mask)

{

uchar tmp;

tmp = Read\_MFRC522(reg);

Write\_MFRC522(reg, tmp & (~mask)); // clear bit mask

}

/\*

\* Function：AntennaOn

\* Description：Turn on antenna, every time turn on or shut down antenna need at least 1ms delay

\* Input parameter：null

\* Return：null

\*/

void AntennaOn(void)

{

uchar temp;

temp = Read\_MFRC522(TxControlReg);

if (!(temp & 0x03))

{

SetBitMask(TxControlReg, 0x03);

}

}

/\*

\* Function：AntennaOff

\* Description：Turn off antenna, every time turn on or shut down antenna need at least 1ms delay

\* Input parameter：null

\* Return：null

\*/

void AntennaOff(void)

{

ClearBitMask(TxControlReg, 0x03);

}

/\*

\* Function：ResetMFRC522

\* Description： reset RC522

\* Input parameter：null

\* Return：null

\*/

void MFRC522\_Reset(void)

{

Write\_MFRC522(CommandReg, PCD\_RESETPHASE);

}

/\*

\* Function：InitMFRC522

\* Description：initilize RC522

\* Input parameter：null

\* Return：null

\*/

void MFRC522\_Init(void)

{

digitalWrite(NRSTPD,HIGH);

MFRC522\_Reset();

//Timer: TPrescaler\*TreloadVal/6.78MHz = 24ms

Write\_MFRC522(TModeReg, 0x8D); //Tauto=1; f(Timer) = 6.78MHz/TPreScaler

Write\_MFRC522(TPrescalerReg, 0x3E); //TModeReg[3..0] + TPrescalerReg

Write\_MFRC522(TReloadRegL, 30);

Write\_MFRC522(TReloadRegH, 0);

Write\_MFRC522(TxAutoReg, 0x40); //100%ASK

Write\_MFRC522(ModeReg, 0x3D); //CRC initilizate value 0x6363 ???

//ClearBitMask(Status2Reg, 0x08); //MFCrypto1On=0

//Write\_MFRC522(RxSelReg, 0x86); //RxWait = RxSelReg[5..0]

//Write\_MFRC522(RFCfgReg, 0x7F); //RxGain = 48dB

AntennaOn(); //turn on antenna

}

/\*

\* Function：MFRC522\_Request

\* Description：Searching card, read card type

\* Input parameter：reqMode--search methods，

\* TagType--return card types

\* 0x4400 = Mifare\_UltraLight

\* 0x0400 = Mifare\_One(S50)

\* 0x0200 = Mifare\_One(S70)

\* 0x0800 = Mifare\_Pro(X)

\* 0x4403 = Mifare\_DESFire

\* return：return MI\_OK if successed

\*/

uchar MFRC522\_Request(uchar reqMode, uchar \*TagType)

{

uchar status;

uint backBits; //the data bits that received

Write\_MFRC522(BitFramingReg, 0x07); //TxLastBists = BitFramingReg[2..0] ???

TagType[0] = reqMode;

status = MFRC522\_ToCard(PCD\_TRANSCEIVE, TagType, 1, TagType, &backBits);

if ((status != MI\_OK) || (backBits != 0x10))

{

status = MI\_ERR;

}

return status;

}

/\*

\* Function：MFRC522\_ToCard

\* Description：communicate between RC522 and ISO14443

\* Input parameter：command--MF522 command bits

\* sendData--send data to card via rc522

\* sendLen--send data length

\* backData--the return data from card

\* backLen--the length of return data

\* return：return MI\_OK if successed

\*/

uchar MFRC522\_ToCard(uchar command, uchar \*sendData, uchar sendLen, uchar \*backData, uint \*backLen)

{

uchar status = MI\_ERR;

uchar irqEn = 0x00;

uchar waitIRq = 0x00;

uchar lastBits;

uchar n;

uint i;

switch (command)

{

case PCD\_AUTHENT: //verify card password

{

irqEn = 0x12;

waitIRq = 0x10;

break;

}

case PCD\_TRANSCEIVE: //send data in the FIFO

{

irqEn = 0x77;

waitIRq = 0x30;

break;

}

default:

break;

}

Write\_MFRC522(CommIEnReg, irqEn|0x80); //Allow interruption

ClearBitMask(CommIrqReg, 0x80); //Clear all the interrupt bits

SetBitMask(FIFOLevelReg, 0x80); //FlushBuffer=1, FIFO initilizate

Write\_MFRC522(CommandReg, PCD\_IDLE); //NO action;cancel current command ???

//write data into FIFO

for (i=0; i<sendLen; i++)

{

Write\_MFRC522(FIFODataReg, sendData[i]);

}

//procceed it

Write\_MFRC522(CommandReg, command);

if (command == PCD\_TRANSCEIVE)

{

SetBitMask(BitFramingReg, 0x80); //StartSend=1,transmission of data starts

}

//waite receive data is finished

i = 2000; //i should adjust according the clock, the maxium the waiting time should be 25 ms???

do

{

//CommIrqReg[7..0]

//Set1 TxIRq RxIRq IdleIRq HiAlerIRq LoAlertIRq ErrIRq TimerIRq

n = Read\_MFRC522(CommIrqReg);

i--;

}

while ((i!=0) && !(n&0x01) && !(n&waitIRq));

ClearBitMask(BitFramingReg, 0x80); //StartSend=0

if (i != 0)

{

if(!(Read\_MFRC522(ErrorReg) & 0x1B)) //BufferOvfl Collerr CRCErr ProtecolErr

{

status = MI\_OK;

if (n & irqEn & 0x01)

{

status = MI\_NOTAGERR; //??

}

if (command == PCD\_TRANSCEIVE)

{

n = Read\_MFRC522(FIFOLevelReg);

lastBits = Read\_MFRC522(ControlReg) & 0x07;

if (lastBits)

{

\*backLen = (n-1)\*8 + lastBits;

}

else

{

\*backLen = n\*8;

}

if (n == 0)

{

n = 1;

}

if (n > MAX\_LEN)

{

n = MAX\_LEN;

}

//read the data from FIFO

for (i=0; i<n; i++)

{

backData[i] = Read\_MFRC522(FIFODataReg);

}

}

}

else

{

status = MI\_ERR;

}

}

//SetBitMask(ControlReg,0x80); //timer stops

//Write\_MFRC522(CommandReg, PCD\_IDLE);

return status;

}

/\*

\* Function：MFRC522\_Anticoll

\* Description：Prevent conflict, read the card serial number

\* Input parameter：serNum--return the 4 bytes card serial number, the 5th byte is recheck byte

\* return：return MI\_OK if successed

\*/

uchar MFRC522\_Anticoll(uchar \*serNum)

{

uchar status;

uchar i;

uchar serNumCheck=0;

uint unLen;

//ClearBitMask(Status2Reg, 0x08); //strSensclear

//ClearBitMask(CollReg,0x80); //ValuesAfterColl

Write\_MFRC522(BitFramingReg, 0x00); //TxLastBists = BitFramingReg[2..0]

serNum[0] = PICC\_ANTICOLL;

serNum[1] = 0x20;

status = MFRC522\_ToCard(PCD\_TRANSCEIVE, serNum, 2, serNum, &unLen);

if (status == MI\_OK)

{

//Verify card serial number

for (i=0; i<4; i++)

{

serNumCheck ^= serNum[i];

}

if (serNumCheck != serNum[i])

{

status = MI\_ERR;

}

}

//SetBitMask(CollReg, 0x80); //ValuesAfterColl=1

return status;

}

/\*

\* Function：CalulateCRC

\* Description：Use MF522 to caculate CRC

\* Input parameter：pIndata--the CRC data need to be read，len--data length，pOutData-- the caculated result of CRC

\* return：Null

\*/

void CalulateCRC(uchar \*pIndata, uchar len, uchar \*pOutData)

{

uchar i, n;

ClearBitMask(DivIrqReg, 0x04); //CRCIrq = 0

SetBitMask(FIFOLevelReg, 0x80); //Clear FIFO pointer

//Write\_MFRC522(CommandReg, PCD\_IDLE);

//Write data into FIFO

for (i=0; i<len; i++)

{

Write\_MFRC522(FIFODataReg, \*(pIndata+i));

}

Write\_MFRC522(CommandReg, PCD\_CALCCRC);

//waite CRC caculation to finish

i = 0xFF;

do

{

n = Read\_MFRC522(DivIrqReg);

i--;

}

while ((i!=0) && !(n&0x04)); //CRCIrq = 1

//read CRC caculation result

pOutData[0] = Read\_MFRC522(CRCResultRegL);

pOutData[1] = Read\_MFRC522(CRCResultRegM);

}

/\*

\* Function：MFRC522\_Write

\* Description：write block data

\* Input parameters：blockAddr--block address;writeData--Write 16 bytes data into block

\* return：return MI\_OK if successed

\*/

uchar MFRC522\_Write(uchar blockAddr, uchar \*writeData)

{

uchar status;

uint recvBits;

uchar i;

uchar buff[18];

buff[0] = PICC\_WRITE;

buff[1] = blockAddr;

CalulateCRC(buff, 2, &buff[2]);

status = MFRC522\_ToCard(PCD\_TRANSCEIVE, buff, 4, buff, &recvBits);

if ((status != MI\_OK) || (recvBits != 4) || ((buff[0] & 0x0F) != 0x0A))

{

status = MI\_ERR;

}

if (status == MI\_OK)

{

for (i=0; i<16; i++) //Write 16 bytes data into FIFO

{

buff[i] = \*(writeData+i);

}

CalulateCRC(buff, 16, &buff[16]);

status = MFRC522\_ToCard(PCD\_TRANSCEIVE, buff, 18, buff, &recvBits);

if ((status != MI\_OK) || (recvBits != 4) || ((buff[0] & 0x0F) != 0x0A))

{

status = MI\_ERR;

}

}

return status;

}

/\*

\* Function：MFRC522\_Halt

\* Description：Command the cards into sleep mode

\* Input parameters：null

\* return：null

\*/

void MFRC522\_Halt(void)

{

uchar status;

uint unLen;

uchar buff[4];

buff[0] = PICC\_HALT;

buff[1] = 0;

CalulateCRC(buff, 2, &buff[2]);

status = MFRC522\_ToCard(PCD\_TRANSCEIVE, buff, 4, buff,&unLen);

}