

# keyestudio

---

## RC522 RFID Module for Arduino



### Introduction

MRC522 RFID module adopts Philips MFRC522 original reader circuit chip design, easy to use, low cost, suitable for equipment development, development of advanced applications such reader users, the need for RF card terminal design / production of the user. This module can be loaded directly into a variety of readers molds. Module uses voltage of 3.3V, through the SPI interface simple few lines can be directly connected to the user any CPU board communication module can guarantee stable and reliable work, reader distance;

### Electrical Parameters

Current :13-26mA / DC 3.3V

Idle Current :10-13mA / DC 3.3V

Sleep current: <80uA

Peak current: <30mA

Operating Frequency: 13.56MHz

Supported card types: mifare1 S50, mifare1 S70, mifare UltraLight, mifare Pro, mifare Desfire

×60mm `mce_style="BACKGROUND-COLOR: #fff">Product Physical Characteristics:`

Dimensions: 40mm × 60mm

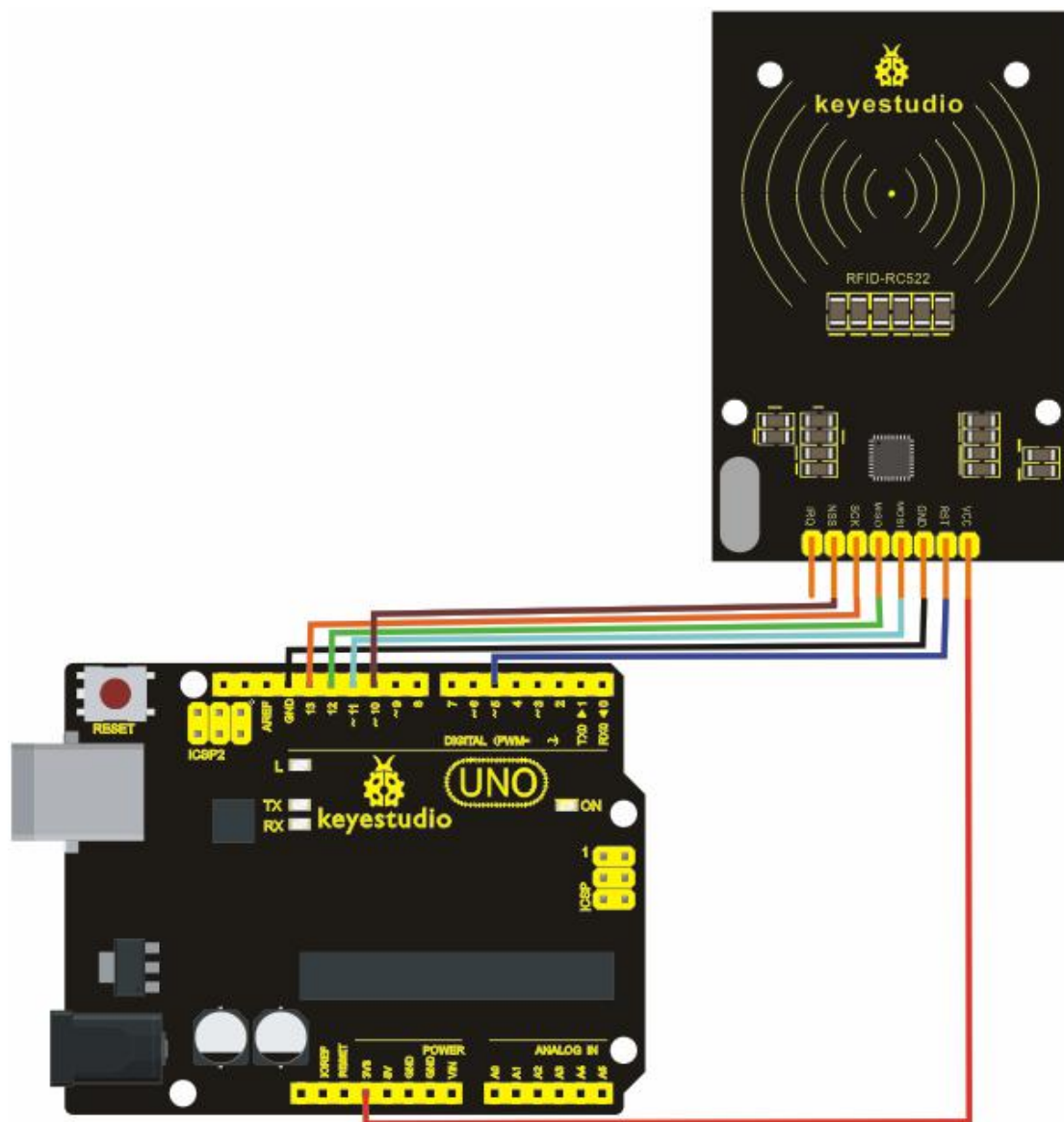
Environmental Operating temperature: -20-80 degrees Celsius

Environment Storage temperature: -40-85 degrees Celsius

Relative Humidity: 5% -95%

# keystudio

## Connection Diagram



## Sample Code

```
#include <SPI.h>
#define uchar unsigned char
#define uint unsigned int
#define MAX_LEN 16
const int chipSelectPin = 10;//if the controller is UNO,328,168
const int NRSTPD = 5;

//MF522command word
#define PCD_IDLE 0x00 //NO action; cancel current command
```

# keyestudio

```
#define PCD_AUTHENT      0x0E      //verify key
#define PCD_RECEIVE     0x08      //receive data

#define PCD_TRANSMIT    0x04      //send data
#define PCD_TRANSCEIVE  0x0C      //receive and send data
#define PCD_RESETPHASE  0x0F      //reset
#define PCD_CALCCRC     0x03      //CRC calculation

//Mifare_One Card command word
#define PICC_REQIDL     0x26      // line-tracking area is dormant
#define PICC_REQALL    0x52      //line-tracking area is
interfered
#define PICC_ANTICOLL   0x93      //Anti collision
#define PICC_SEIECTTAG  0x93      //choose cards
#define PICC_AUTHENT1A  0x60      //Verify A key
#define PICC_AUTHENT1B  0x61      //Verify B key
#define PICC_READ       0x30      // Reader Module
#define PICC_WRITE      0xA0      // letter block

#define PICC_DECREMENT  0xC0
#define PICC_INCREMENT  0xC1
#define PICC_RESTORE    0xC2      //Transfer data to buffer
#define PICC_TRANSFER   0xB0      //Save buffer data
#define PICC_HALT       0x50      //Dormancy

//MF522 Error code returned when communication
#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 DivIEnReg      0x03
#define CommIrqReg     0x04
#define DivIrqReg      0x05
#define ErrorReg       0x06
#define Status1Reg     0x07
#define Status2Reg     0x08
```

# keyestudio

---

```
#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
```

# keyestudio

```
//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

uchar serNum[5];
uchar  writeDate[16]={'T','e','n','g',' ','B','o',0,0,0,0,0,0,0,0,0,0};

uchar sectorKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             };

uchar sectorNewKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                               {0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF},
                               0xff,0x07,0x80,0x69, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                               {0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF,  0xFF},
                               0xff,0x07,0x80,0x69, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                               };

void setup() {
    Serial.begin(9600);                // RFID reader SOUT pin connected to Serial
    RX pin at 2400bps
    // start the SPI library:
    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 10 , Not Reset and
    Power-down
}
```

# keyestudio

---

```
digitalWrite(NRSTPD, HIGH);
```

```
MFRC522_Init();
}

void loop()
{
    uchar i,tmp;
    uchar status;
    uchar str[MAX_LEN];
    uchar RC_size;
    uchar blockAddr; //Select the address of the operation 0~63

    // searching card, return card type
    status = MFRC522_Request(PICC_REQIDL, str);
    if (status == MI_OK)
    {
    }

    status = MFRC522_Anticoll(str);
    memcpy.serNum, str, 5);
    if (status == MI_OK)
    {
        Serial.println("The card's number is  : ");
        Serial.print.serNum[0],BIN);
        Serial.print.serNum[1],BIN);
        Serial.print.serNum[2],BIN);
        Serial.print.serNum[3],BIN);
        Serial.print.serNum[4],BIN);
        Serial.println(" ");
    }

    // select card, return card capacity
    RC_size = MFRC522_SelectTag.serNum);
    if (RC_size != 0)
    {}

    // write data card
    blockAddr = 7; // data block 7
    status = MFRC522_Auth(PICC_AUTHENT1A, blockAddr, sectorKeyA[blockAddr/4],
```

# keyestudio

---

```
serNum); // authentication
    if (status == MI_OK)

    {
        // write data
        status = MFRC522_Write(blockAddr, sectorNewKeyA[blockAddr/4]);
        Serial.print("set the new card password, and can modify the data of
the Sector: ");

        Serial.print(blockAddr/4,DEC);

        // write data
        blockAddr = blockAddr - 3 ;
        status = MFRC522_Write(blockAddr, writeDate);
        if(status == MI_OK)
        {
            Serial.println("OK!");
        }
    }

    // read card
    blockAddr = 7;          // data block 7
    status = MFRC522_Auth(PICC_AUTHENT1A, blockAddr,

sectorNewKeyA[blockAddr/4], serNum); // authentication
    if (status == MI_OK)
    {
        // read data
        blockAddr = blockAddr - 3 ;
        status = MFRC522_Read(blockAddr, str);
        if (status == MI_OK)
        {
            Serial.println("Read from the card ,the data is : ");
            for (i=0; i<16; i++)
            {
                Serial.print(str[i]);
            }
            Serial.println(" ");
        }
    }
    Serial.println(" ");
    MFRC522_Halt();          // command card into sleeping mode
}
}
```

# keystudio

---

```
void Write_MFRC522(uchar addr, uchar val)
```

```
{
    digitalWrite(chipSelectPin, LOW);

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

    digitalWrite(chipSelectPin, HIGH);
}
```

```
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;
}
```

```
void SetBitMask(uchar reg, uchar mask)
```

```
{
    uchar tmp;
    tmp = Read_MFRC522(reg);
    Write_MFRC522(reg, tmp | mask); // set bit mask
}
```

```
void ClearBitMask(uchar reg, uchar mask)
```

```
{
    uchar tmp;
    tmp = Read_MFRC522(reg);
```



# keyestudio

---

```
    Write_MFRC522(reg, tmp & (~mask)); // clear bit mask
}

void AntennaOn(void)
{
    uchar temp;

    temp = Read_MFRC522(TxControlReg);
    if (!(temp & 0x03))
    {
        SetBitMask(TxControlReg, 0x03);
    }
}

void AntennaOff(void)
{
    ClearBitMask(TxControlReg, 0x03);
}

void MFRC522_Reset(void)
{
    Write_MFRC522(CommandReg, PCD_RESETPHASE);
}

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 original value 0x6363 ???

    AntennaOn(); // open antenna
}
uchar MFRC522_Request(uchar reqMode, uchar *TagType)
```

# keyestudio

---

```
{
    uchar status;

    uint backBits;          // bits of data 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;
}

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:    // card key authentication
        {
            irqEn = 0x12;
            waitIRq = 0x10;
            break;
        }
        case PCD_TRANSCEIVE: // send data in FIFO
        {
            irqEn = 0x77;
            waitIRq = 0x30;
            break;
        }
        default:
    }
```

# keyestudio

---

```
        break;
    }

    Write_MFRC522(CommIEnReg, irqEn|0x80); // permission for interrupt request
    ClearBitMask(CommIrqReg, 0x80); // clear all bits of the interrupt request
    SetBitMask(FIFOLevelReg, 0x80); //FlushBuffer=1, FIFO initialize

    Write_MFRC522(CommandReg, PCD_IDLE); //NO action; clear current command ???

    // write data into FIFO
    for (i=0; i<sendLen; i++)
    {
        Write_MFRC522(FIFODataReg, sendData[i]);
    }

    // execute command
    Write_MFRC522(CommandReg, command);
    if (command == PCD_TRANSCEIVE)
    {
        SetBitMask(BitFramingReg, 0x80); //StartSend=1,transmission of data starts
    }

    // wait for the completion of data transmission
    i = 2000; // adjust i according to clock frequency, max wait time for M1 card operation 25ms
    ???
    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)
```

# keystudio

---

```
{
    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 received in FIFO
    for (i=0; i<n; i++)
    {
        backData[i] = Read_MFRC522(FIFODataReg);
    }
}
else
{
    status = MI_ERR;
}

}

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

# keyestudio

---

```
//Write_MFRC522(CommandReg, PCD_IDLE);

return status;
}

uchar MFRC522_Anticoll(uchar *serNum)
{
    uchar status;
    uchar i;
    uchar serNumCheck=0;
    uint unLen;

    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 sequence number
        for (i=0; i<4; i++)
        {
            serNumCheck ^= serNum[i];
        }
        if (serNumCheck != serNum[i])
        {
            status = MI_ERR;
        }
    }

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

    return status;
}

void CalulateCRC(uchar *pIndata, uchar len, uchar *pOutData)
{
    uchar i, n;

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

# keystudio

---

```
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);

// wait for completion of CRC calculation
i = 0xFF;
do
{
    n = Read_MFRC522(DivIrqReg);
    i--;
}
while ((i!=0) && !(n&0x04));           //CRCIrq = 1

// read result from CRC calculation
pOutData[0] = Read_MFRC522(CRCResultRegL);
pOutData[1] = Read_MFRC522(CRCResultRegM);
}

uchar MFRC522_SelectTag(uchar *serNum)
{
    uchar i;
    uchar status;
    uchar size;
    uint recvBits;
    uchar buffer[9];

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

    buffer[0] = PICC_SEIECTTAG;
    buffer[1] = 0x70;
    for (i=0; i<5; i++)
    {
        buffer[i+2] = *(serNum+i);
    }
    CalculateCRC(buffer, 7, &buffer[7]);           //??
    status = MFRC522_ToCard(PCD_TRANSCEIVE, buffer, 9, buffer, &recvBits);
}
```

# keystudio

---

```
if ((status == MI_OK) && (recvBits == 0x18))

{
    size = buffer[0];
}
else
{
    size = 0;
}

return size;
}

uchar MFRC522_Auth(uchar authMode, uchar BlockAddr, uchar *Sectorkey, uchar *serNum)
{
    uchar status;
    uint recvBits;
    uchar i;
    uchar buff[12];

    // Verification instructions + block address + sector password + card sequence number
    buff[0] = authMode;
    buff[1] = BlockAddr;
    for (i=0; i<6; i++)

    {
        buff[i+2] = *(Sectorkey+i);
    }
    for (i=0; i<4; i++)
    {
        buff[i+8] = *(serNum+i);
    }
    status = MFRC522_ToCard(PCD_AUTHENT, buff, 12, buff, &recvBits);

    if ((status != MI_OK) || (!(Read_MFRC522(Status2Reg) & 0x08)))
    {
        status = MI_ERR;
    }

    return status;
}
```

# keyestudio

---

```
uchar MFRC522_Read(uchar blockAddr, uchar *recvData)
{
    uchar status;
    uint unLen;

    recvData[0] = PICC_READ;
    recvData[1] = blockAddr;
    CalculateCRC(recvData,2, &recvData[2]);
    status = MFRC522_ToCard(PCD_TRANSCEIVE, recvData, 4, recvData, &unLen);

    if ((status != MI_OK) || (unLen != 0x90))
    {
        status = MI_ERR;
    }

    return status;
}
```

```
uchar MFRC522_Write(uchar blockAddr, uchar *writeData)
{
    uchar status;
    uint recvBits;
    uchar i;
    uchar buff[18];

    buff[0] = PICC_WRITE;
    buff[1] = blockAddr;
    CalculateCRC(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 16Byte data into FIFO
        {
            buff[i] = *(writeData+i);
        }
    }
}
```



# keyestudio

---

```
    CalculateCRC(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;
}

void MFRC522_Halt(void)
{
    uchar status;
    uint unLen;
    uchar buff[4];

    buff[0] = PICC_HALT;
    buff[1] = 0;
    CalculateCRC(buff, 2, &buff[2]);

    status = MFRC522_ToCard(PCD_TRANSCEIVE, buff, 4, buff, &unLen);
}
```

## Test Phenomenon

In this experiment, when the IC card gets close, RFID module writes data into the IC card, then reads out the data and displays it on the monitor window. As below picture shown:

# keystudio

---

