Tutoriels pour Arduino

Tutoriels Tiptoopboards

Vers le contenu

Recherche… Rechercher

Recherche avancée

- Index du forum ‹ Tutoriels ‹ Capteurs et composants
- Modifier la taille de la police
- Imprimer le sujet

- FAQ
- M’enregistrer
- Connexion

Module d'identification de puce RFID avec Arduino

Répondre

Dans ce sujet… Rechercher

3 messages • Page **1** sur **1**

Module d'identification de puce RFID avec Arduino

par **tiptop** » Mer 21 Aoû 2013 18:41

Module d'identification de puce RFID avec Arduino

Composants utilisés

- 1 module de lecture RFID (vendu en kit avec des clés RFID)
- 1 carte Arduino Uno
- 5 badges RFID (clés), vendus par lot
- 1 carte RFID (clé RFID au format carte de crédit)
- 1 plaque de branchements
- quelques fils de câblage Dupont

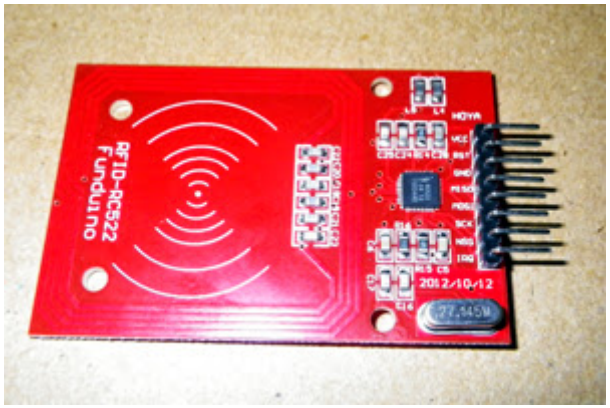
Principe de fonctionnement

La radio identification permet de récupérer des données à distance en utilisant des radio-étiquettes (*RFID tag*, ou transpondeur), collées ou fixées sur des objets (badges, port clé, étiquette électronique...).

Les étiquettes très fines comportent une antenne associée à une puce électronique qui leur permet de recevoir et de répondre aux requêtes radio émises depuis l'émetteur-récepteur.

Elles contiennent un identifiant individuel (standard Epc-96 sur 96 bits). Leur fonctionnement est passif (sans source d'énergie, donc c'est économique et inusable).

Le Lecteur RFID



Un module RFID (Philips MFRC522) est monté sur une carte déjà câblée avec une antenne prêt à l'emploi.
Il permet d'identifier sans contact des puces RFID présentées à proximité placées dans une carte ou un badge.

Ceci permet une lecture de badge sans contact.

Utilisations variées:

- Badges d'accès
- identifier une personne ou un objet sans contact
- Cartes de visite numérique
- Payement direct
- Identifier une carte de transport
- Péage urbain rapide

- Identification de livre de bibliothèque...

Des puces RFID miniatures sont également utilisées pour le tatouage de chiens et chats, de bétail, d'arbres en ville...

Câblage du lecteur Rfid

L'alimentation est de 3.3V (pas 5V), le lecteur utilise une interface SPI, et fonctionne sur une fréquence de 13.56 MHz (donc il lit tous les badges répondant à cette fréquence).

Le **bus SPI** se pilote par 4 signaux logiques

Brochage sur Carte Arduino **UNO** :

- SCLK Horloge (généré par le maître) --> pin 13 ou ICSP-3
- MOSI Master Output, Slave Input (généré par le maître) --> pin 11 ou ICSP-4
- MISO Master Input, Slave Output (généré par l'esclave) --> pin 12 ou ICSP-1
- SS Slave Select, actif à l'état bas, (généré par le maître) --> pin 10

Le brochage SPI dépend des cartes (Uno, Mega, Nano) voir sur http://arduino.cc/en/Reference/SPI

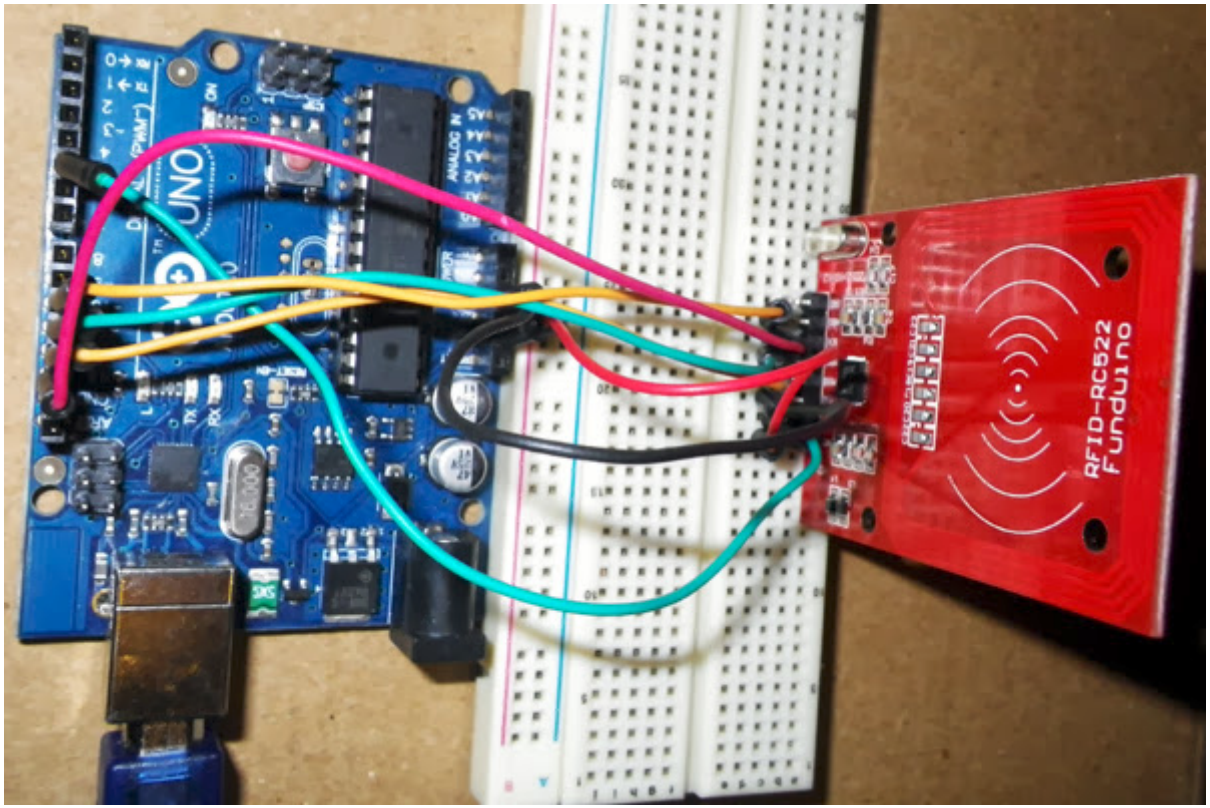
Arduino Board	MOSI	MISO	SCK	SS (slave)	SS (master)
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	-
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	-
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-
Due	ICSP-4	ICSP-1	ICSP-3	-	4, 10, 52

Note that MISO, MOSI, and SCK are available in a consistent physical location on the ICSP header; this is useful, for example, in designing a shield that works on every board.

Branchements à réaliser

- Vcc --> +3.3V alimentation
- Rst --> pin 5 reset
- Gnd -->-gnd masse
- Miso --> pin 12
- Mosi -->pin 11

- Sck --> pin 13
- Nss --> pin 10
- Irq --> non connecté



Programmation

Il faut installer la librairie SPI.

Indiquer les pins utilisées dans le programme :
chipSelectPin = 10;

const int NRSTPD = 5;

Exemple de programme :

Code: Tout sélectionner

```
// ==== Programme pour utiliser le lecteur RFID ====
// Le module Philips MFRC522 fonctionne à 13.56 MHz
//
// Commentaires et modifications tiptopboards.com 19 08 2013
//
// Source : RFID.pde - Dr.Leong - WWW.B2CQSHOP.COM - 2011.09.19

 // Le capteur communique avec SPI, inclure cette librairie
#include <SPI.h>

#define   uchar   unsigned char
#define   uint   unsigned int

//==== Broches à câbler entre lecteur RFID / Carte Arduino UNO
// VCC sur +3.3V (alimentation)
// RST sur pin 5 (reset)
// GND sur GND (masse)
// MISO sur pin 12 (uno)
// MOSI sur pin 11 (uno)
// SCK  sur pin 13 (uno)
// NSS sur pin 10 (uno)
// IRQ  non utilisé ici
//=====================================================
#define MAX_LEN 16
//Définir les broches
const int chipSelectPin = 10;
```

```
const int NRSTPD = 5;

//MF522
#define PCD_IDLE              0x00               //NO action;
#define PCD_AUTHENT           0x0E
#define PCD_RECEIVE           0x08
#define PCD_TRANSMIT          0x04
#define PCD_TRANSCEIVE        0x0C
#define PCD_RESETPHASE        0x0F
#define PCD_CALCCRC           0x03

//Mifare_Ono
#define PICC_REQIDL           0x26
#define PICC_REQALL           0x52
#define PICC_ANTICOLL         0x93
#define PICC_SElECTTAG        0x93
#define PICC_AUTHENT1A        0x60
#define PICC_AUTHENT1B        0x61
#define PICC_READ             0x30
#define PICC_WRITE            0xA0
#define PICC_DECREMENT        0xC0
#define PICC_INCREMENT        0xC1
#define PICC_RESTORE          0xC2
#define PICC_TRANSFER         0xB0
#define PICC_HALT             0x50


//MF522
#define MI_OK                 0
#define MI_NOTAGERR           1
#define MI_ERR                2
```

```
//------------------MFRC522--------------
//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
//-----------------------------------------------

uchar serNum[5];

uchar  writeData[16]={0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100};
uchar  moneyConsume = 18 ;
uchar  moneyAdd = 10 ;
//6Byte
 uchar sectorKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                             //{0x19, 0x84, 0x07, 0x15, 0x76, 0x14},
                             {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                            };
 uchar sectorNewKeyA[16][16] = {{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF},
                                {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff,0x07,0x80,0x69, 0x19,0x84,0x07,0x15,0x76,0x14},
                                 //you can set another ket , such as  " 0x19, 0x84, 0x07, 0x15, 0x76, 0x14 "
                                 //{0x19, 0x84, 0x07, 0x15, 0x76, 0x14, 0xff,0x07,0x80,0x69, 0x19,0x84,0x07,0x15,0x76,0x14},
                                 // but when loop, please set the  sectorKeyA, the same key, so that RFID module can read the card
                                {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xff,0x07,0x80,0x69, 0x19,0x33,0x07,0x15,0x34,0x14},
                               };

// ================Setup ====================
```

```
void setup() {
   Serial.begin(9600);         // RFID reader SOUT pin connected to Serial RX pin at 2400bps
 // lance la librairie SPI :
  SPI.begin();

  pinMode(chipSelectPin,OUTPUT);        // digital pin 10 en OUTPUT pour connecter au RFID /ENABLE pin
    digitalWrite(chipSelectPin, LOW);   // Activer le lecteur RFID
  pinMode(NRSTPD,OUTPUT);               // digital pin 5 , Not Reset and Power-down
    digitalWrite(NRSTPD, HIGH);

  MFRC522_Init();  //Initialise le lecteur RFID
}

// ====== Lecture des cartes RFID placées devant le lecteur ===========
void loop()
{
     uchar i,tmp;
   uchar status;
        uchar str[MAX_LEN];
        uchar RC_size;
        uchar blockAddr;
        String mynum = "";


      //status
      status = MFRC522_Request(PICC_REQIDL, str);
      if (status == MI_OK)
      {
                        Serial.println("Carte detectee");
         Serial.print(str[0],BIN);
                        Serial.print(" , ");
```

```
   Serial.print(str[1],BIN);
                Serial.println(" ");
}

//
status = MFRC522_Anticoll(str);
memcpy(serNum, str, 5);
if (status == MI_OK)
{

                Serial.println("Le numero de carte est  : ");
   Serial.print(serNum[0]);
                Serial.print(" , ");
   Serial.print(serNum[1],BIN);
                Serial.print(" , ");
   Serial.print(serNum[2],BIN);
                Serial.print(" , ");
   Serial.print(serNum[3],BIN);
                Serial.print(" , ");
   Serial.print(serNum[4],BIN);
                Serial.println(" ");

                // Il faut comparer toutes les paires, dans cet exemple on ne regarde que la première
                //Regarder si on a reconnu une carte déterminée
                if(serNum[0] == 125) {
                  Serial.println("Bonjour Titopboards.com");
                } else if(serNum[0] == 253) {
                  Serial.println("Bonjour Christian");
                }
                delay(1000);
}
```

```
                //Serial.println(" ");
              MFRC522_Halt();

}

//======= Write_MFRC5200  addr-- ; val--  ======================
void Write_MFRC522(uchar addr, uchar val)
{
   digitalWrite(chipSelectPin, LOW);

   //:0XXXXXX0
   SPI.transfer((addr<<1)&0x7E);
   SPI.transfer(val);

   digitalWrite(chipSelectPin, HIGH);
}


//======== Read _MFRC522 : addr---  ============================
uchar Read_MFRC522(uchar addr)
{
   uchar val;

   digitalWrite(chipSelectPin, LOW);

   //:1XXXXXX0
   SPI.transfer(((addr<<1)&0x7E) | 0x80);
   val =SPI.transfer(0x00);

   digitalWrite(chipSelectPin, HIGH);
```

```
   return val;
}

//=========== SetBitMask  RC522:reg-- ;mask--  ====================
void SetBitMask(uchar reg, uchar mask)
{
    uchar tmp;
    tmp = Read_MFRC522(reg);
    Write_MFRC522(reg, tmp | mask);  // set bit mask
}


// === ClearBitMask RC522 :reg--;mask--  ==============
void ClearBitMask(uchar reg, uchar mask)
{
    uchar tmp;
    tmp = Read_MFRC522(reg);
    Write_MFRC522(reg, tmp & (~mask));  // clear bit mask
}


// ========= AntennaOn ======================================
void AntennaOn(void)
{
   uchar temp;

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

```
}


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


//======== ResetMFRC522 ================================
void MFRC522_Reset(void)
{
    Write_MFRC522(CommandReg, PCD_RESETPHASE);
}


//====== InitMFRC522 ================================
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初始值0x6363   ???

   //ClearBitMask(Status2Reg, 0x08);      //MFCrypto1On=0
   //Write_MFRC522(RxSelReg, 0x86);      //RxWait = RxSelReg[5..0]
   //Write_MFRC522(RFCfgReg, 0x7F);         //RxGain = 48dB

   AntennaOn();
}


/* ==== MFRC522_Request :reqMode, ====================
 *          TagType--
 *             0x4400 = Mifare_UltraLight
 *            0x0400 = Mifare_One(S50)
 *            0x0200 = Mifare_One(S70)
 *            0x0800 = Mifare_Pro(X)
 *            0x4403 = Mifare_DESFire
 *  MI_OK
 */
uchar MFRC522_Request(uchar reqMode, uchar *TagType)
{
   uchar status;
   uint backBits;

   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;
}


/* MFRC522_ToCard RC522 ISO14443 : command--MF522
 *          sendData--RC522,
 *          sendLen--
 *          backData--,
 *          backLen--
 * MI_OK
 */
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:
      {
         irqEn = 0x12;
         waitIRq = 0x10;
         break;
      }
```

```
    case PCD_TRANSCEIVE:   //FIFO
    {
       irqEn = 0x77;
       waitIRq = 0x30;
       break;
    }
    default:
       break;
 }

 Write_MFRC522(CommIEnReg, irqEn|0x80);
 ClearBitMask(CommIrqReg, 0x80);
 SetBitMask(FIFOLevelReg, 0x80);         //FlushBuffer=1, FIFO

Write_MFRC522(CommandReg, PCD_IDLE);   //NO action;

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

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

//
i = 2000;   //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)
        {
        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;
               }

               //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;
   }
```

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


    //ClearBitMask(Status2Reg, 0x08);      //TempSensclear
    //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)
   {
      //
      for (i=0; i<4; i++)
      {
          serNumCheck ^= serNum[i];
      }
      if (serNumCheck != serNum[i])
      {
         status = MI_ERR;
      }
    }
```

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

    return status;
}


// CalulateCRC  MF522 CRC :pIndata--,len--,pOutData--
void CalulateCRC(uchar *pIndata, uchar len, uchar *pOutData)
{
    uchar i, n;

    ClearBitMask(DivIrqReg, 0x04);         //CRCIrq = 0
    SetBitMask(FIFOLevelReg, 0x80);         //FIFO
    //Write_MFRC522(CommandReg, PCD_IDLE);

   //FIFO
    for (i=0; i<len; i++)
    {
      Write_MFRC522(FIFODataReg, *(pIndata+i));
   }
    Write_MFRC522(CommandReg, PCD_CALCCRC);

   //CRC
    i = 0xFF;
    do
    {
        n = Read_MFRC522(DivIrqReg);
        i--;
    }
    while ((i!=0) && !(n&0x04));         //CRCIrq = 1
```

```
   //CRC
    pOutData[0] = Read_MFRC522(CRCResultRegL);
    pOutData[1] = Read_MFRC522(CRCResultRegM);
}


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

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

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

    if ((status == MI_OK) && (recvBits == 0x18))
    {
      size = buffer[0];
   }
```

```
    else
    {
      size = 0;
   }

    return size;
}


/* MFRC522_Auth
 *
 *:authMode--
             0x60 = 验证A密钥
             0x61 = 验证B密钥
             BlockAddr--块地址
             Sectorkey--扇区密码
             serNum--卡片序列号,4字节
 * MI_OK
 */
uchar MFRC522_Auth(uchar authMode, uchar BlockAddr, uchar *Sectorkey, uchar *serNum)
{
    uchar status;
    uint recvBits;
    uchar i;
   uchar buff[12];

   //
    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;
}


// ======= MFRC522_Read :blockAddr--;recvData--
uchar MFRC522_Read(uchar blockAddr, uchar *recvData)
{
    uchar status;
    uint unLen;

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

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

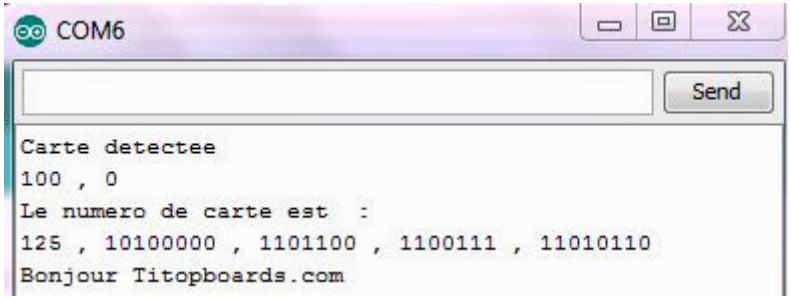
```
    }

    return status;
}


// ====MFRC522_Write :blockAddr--;writeData-- 16
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++)      //FIFO 16Byte
        {
           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;
}


// === MFRC522_Halt ==============
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);
}
```

Exemple de détection de carte et de badges

carte 125 , 10100000 , 1101100 , 1100111 , 11010110

clé 1 125 , 1000101 , 10111001 , 11010011 , 1010010
clé 2 125 , 10100011 , 10001001 , 11001001 , 10011110
clé 3 125 , 111110 , 11110100 , 11010011 , 1100100
clé 4 237 , 1001 , 1000 , 11010100 , 111000
clé 5 253 , 1000 , 10010100 , 11010011 , 10110010
autre 91 , 10011111 , 11100100 , 1110101 , 1010101



Chaque numéro lu est unique (milliards de combinaisons).
Il suffit de tester si un code lu est présent dans une base de données ou une liste pour réaliser ensuite une action particulière (ouverture de porte par relais, etc...)

Références

Librairie SPI
http://arduino.cc/en/Reference/SPI

Code RFID pour Arduino (commenté en chinois)
http://www.vetco.net/catalog/product_in ... s_id=13579

Autres lectures possibles(non testé)

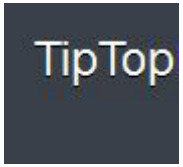
RFID et Eprom
http://skyduino.wordpress.com/2011/12/1 ... terne-i2c/

Plyground
http://playground.arduino.cc/Learning/PRFID

Instructables
http://www.instructables.com/id/Arduino ... eedstudio/

Datasheet MFRC522
http://www.vetco.net/datasheets/VUPN632 ... asheet.pdf

Pour carte Nano
http://www.onemansanthology.com/blog/rf ... an-module/
tiptopboards - admin



tiptop
Administrateur du site

Messages: 85
Inscription: Mar 13 Aoû 2013 20:38
Localisation: 38
- Site Internet

Haut

Re: Module d'identification de puce RFID avec Arduino

par **soda34** » Dim 15 Fév 2015 20:12

Bonsoir,

je suis tout nouveau dans le RFID, je recherche les code source pour lecture et ecriture dans carte ou balge

Merci à tous pour votre aide
ARDUINO

soda34

Messages: 1
Inscription: Dim 15 Fév 2015 16:37

Haut

Re: Module d'identification de puce RFID avec Arduino

par **vidalv** » Mer 13 Avr 2016 20:09

Bonsoir, bon petit tuto sur les modules RFID bravo !
du coup j'ai moi même testé ce kit sur un Arduino Mega avec les librairies glanées sur : https://github.com/miguelbalboa/rfid
bon même si j'ai pas tout compris j'ai put relire les deux Tag livré avec ce qui me donne,

Pour le porte clef :

Card UID: E6 03 FB 30
PICC type: MIFARE 1KB
Success with key: FF FF FF FF FF FF
Block 0: E6 03 FB 30 2E 08 04 00 62 63 64 65 66 67 68 69

et sur celle au format carte de crédit :

Card UID: 67 8E B1 65
PICC type: MIFARE 1KB
Success with key: FF FF FF FF FF FF
Block 0: 67 8E B1 65 3D 88 04 00 85 00 B4 2E F0 BB 6A A8

Questions:
– Que peut-on (Re)programmer la dedans afin de personnaliser le Tag (si possible) ?
– Qu'est-ce qui reste "unique" l'UID ?
– Combien de bloc possible à part le 0 ?

Merci à vous.

vidalv

Messages: 1
Inscription: Mer 13 Avr 2016 16:38

Haut

Afficher les messages postés depuis: Tous Trier par Date Croissant Aller

Répondre
3 messages • Page **1** sur **1**

Retourner vers Capteurs et composants

Aller à: Capteurs et composants Aller

Qui est en ligne

Utilisateurs parcourant ce forum: Aucun utilisateur enregistré et 10 invités

- Index du forum
- L'équipe du forum • Supprimer les cookies du forum • Heures au format UTC [Heure d'été]