

**课 程 实 验 报 告**

**课程名称： 射频识别技术原理及应用**

**专业班级： 物联网工程1501**

**学 号： U201514881**

**姓 名： 赵浩东**

**指导教师： 甘早斌**

**报告日期： 2018.5.10**

**计算机科学与技术学院**

**目 录**

[1 实验一 低频读写器实验 3](#_Toc514870932)

[1.1 实验目的 3](#_Toc514870933)

[1.2 实验内容及结果 3](#_Toc514870934)

[1.3 实验体会与总结 4](#_Toc514870935)

[1.4 核心源码说明 4](#_Toc514870936)

[2 实验二 高频读写器实验ISO14443A 6](#_Toc514870937)

[2.1实验目的 6](#_Toc514870938)

[2.2实验内容及结果 6](#_Toc514870939)

[2.3实验体会与总结 10](#_Toc514870940)

[2.4核心源码说明 10](#_Toc514870941)

[3 实验三 高频读写器实验ISO15693 14](#_Toc514870942)

[3.1实验目的 14](#_Toc514870943)

[3.2实验内容及结果 14](#_Toc514870944)

[3.3实验体会与总结 20](#_Toc514870945)

[3.4核心源码说明 20](#_Toc514870946)

[4 实验四 超高频读写器实验 24](#_Toc514870947)

[4.1实验目的 24](#_Toc514870948)

[4.2实验内容及结果 24](#_Toc514870949)

[4.3实验体会与总结 30](#_Toc514870950)

[5 实验五 RFID综合应用实验 31](#_Toc514870951)

[5.1需求分析 31](#_Toc514870952)

[5.2系统详细设计 32](#_Toc514870953)

[5.3系统实现与系统测试 39](#_Toc514870954)

[5.4总结 45](#_Toc514870955)

[5.5系统源代码 46](#_Toc514870956)

# 1 实验一 低频读写器实验

## 1.1 实验目的

通过本次实验了解博创科技 RFID 读写器的结构组成，熟悉各个模块的功能，掌握试验箱的连接和操作方法。掌握串口命令参数的意义和设置方式。

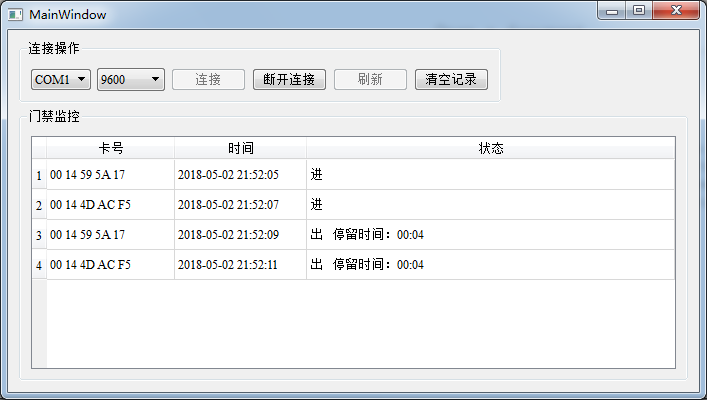
了解低频读写器的基本原理，学会如何使用实训软件对低频读写器进行读卡操作(验证性实验)。

学习和掌握在低频读写器的编程操作，对标签进行读操作，了解低频读写器的工作机理，并完成一个示例程序。

## 1.2 实验内容及结果

1. 完成低频读写器的标签读取试验；
2. 打开试验箱，连接电源，将串口连接至电脑。
3. 使用模块切换按钮直至切换到M125K模块。
4. 打开程序M125KDemo，选择串口号，点击连接按钮连接。
5. 连接成功后，将带有标签的卡片靠近M125K模块。
6. 观察输出信息。

2、熟悉低频LF-125K开发实例，完成低频读写器编程实验，熟悉和了解低频读写器API函数；分析应用demo存在的问题，进而完善应用demo的功能。完善的功能包括：记录保存进出的历史记录、停留时间。

原始demo存在的问题如下：同一个卡号会执行覆盖写，只会显示最后一次的操作而无法获取之前的操作信息。同时还应添加显示每次进入的停留时间。最终效果图如下所示：

## 1.3 实验体会与总结

本次实验较为简单，了解了125KHz模块以及相应的电子标签，学会了Qt关于串口的API，对于Qt有了更深的掌握，在初始程序的基础上完善了其功能，包括清楚记录，计算停留时间等。

## 1.4 核心源码说明

/\*\*

\* @brief MainWindow::readData

\* 读取串口数据

\*/

void MainWindow::readData()

{

if(serialPort->bytesAvailable() < 5)

return;

QByteArray data = serialPort->readAll();

if(m125dll->LF125K\_FrameAnalysis((uint8 \*)(data.data())) == 0)

{

QString tagId = CharStringtoHexString(tr(" "),data.data(),data.length());

QString time = CurrentDateTime();

int index = model->findRecord(tagId);

if(index >= 0 )

{

QString text = model->record(index).value(2).toString();

if(text == tr("进")){

QString lasttime=model->record(index).value(1).toString();

QDateTime start = QDateTime::fromString(lasttime, "yyyy-MM-dd hh:mm:ss");

QDateTime end = QDateTime::fromString(time, "yyyy-MM-dd hh:mm:ss");

uint stime = start.toTime\_t();

uint etime = end.toTime\_t();

QDateTime sttime = QDateTime::fromTime\_t(etime-stime);

QString staytime=sttime.toString("mm:ss");

model->addRecord(tagId,time,tr("出 停留时间：")+staytime);

}

else{

model->addRecord(tagId,time,tr("进"));

}

}

else {

model->addRecord(tagId,time,tr("进"));

}

}

}

通过findRecord返回表中对应ID的最后一条记录的行号，若无该ID则插入记录，状态为“进”。若存在该ID则判断状态，如果状态为“进”，则读取该记录的时间，将当下时间减去记录时间，即为停留时间，并与状态为“出”一起添加到记录中；如果状态为“出”，则添加记录为“进”的状态。

需要注意的是，从表中读取到的时间为字符串类型，需要转换成时间类型再转换成整型计算差值，再变回字符串类型打印在状态栏。

# 2 实验二 高频读写器实验ISO14443A

## 2.1实验目的

通过本次实验了解高频读写器的基本原理，学会如何使用高频读写器，掌握串口命令参数的意义和设置方式。

阅读和了解ISO14443A协议的主要内容，进一步加深对S50卡的存储结构和 ISO14443A协议的理解，掌握ISO14443A协议的常用命令的含义和用法。

通过高频读写器的实验，掌握对 S50 卡各个扇区数据的读写方法，并熟悉高 频读写器(ISO14443A)API函数。

## 2.2实验内容及结果

1、通过试验箱，学会对ISO14443A协议下标签进行寻卡、唤醒、休眠操作，掌握寻卡、唤醒、休眠的命令代码、命令功能和命令数据包构成，并分析唤醒、休眠命令对标签寻卡所产生的影响。

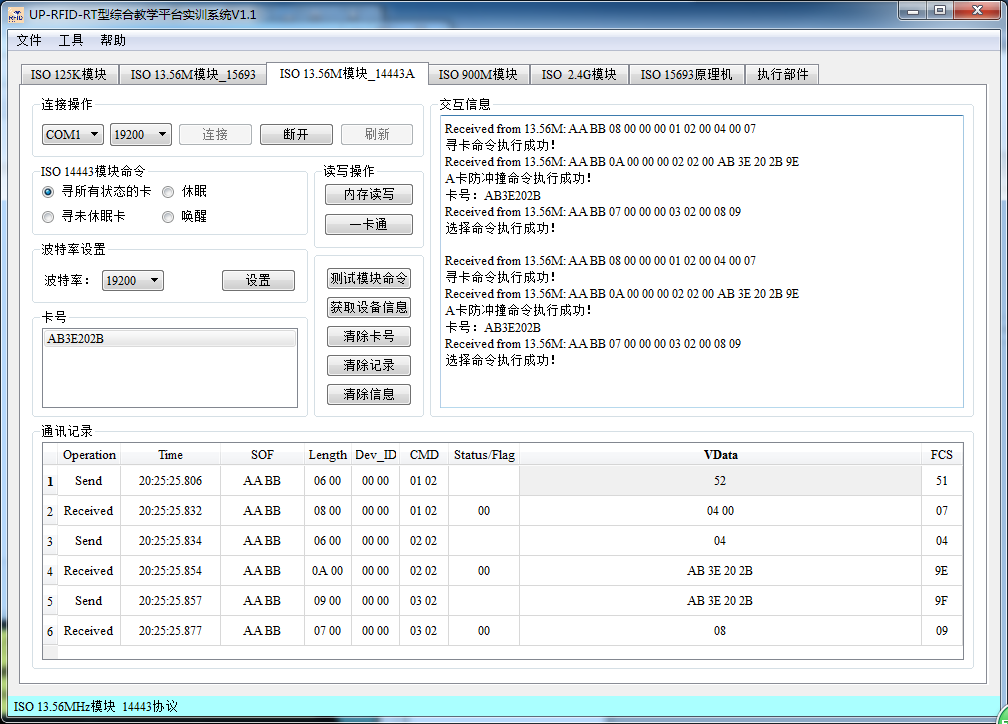
2、通过试验箱，学会对ISO14443A协议下标签指定内存的数据读写。掌握块3存取控制位对本扇区其他块的读写操作控制逻辑。掌握密码A和密码B的加载方法以及对不同块的读写控制逻辑。

3、学会通过试验箱对ISO14443A协议下标签进行加值减值操作;

4、掌握高频读写器API函数的调用方法，并能够通过编程实现对ISO14443A协议下标签数据的读写操作以及加值减值操作。

**操作过程**

1. 打开试验箱，连接电源，将串口连接至电脑。
2. 使用模块切换按钮直至切换到13.56MHz模块。
3. 打开程序RFIDDemo，切换到14443A协议界面，选择串口号，点击连接按钮连接，波特率为19200。
4. 连接成功后，将带有标签的卡片置于模块上。



模块命令选择“寻所有状态卡”，点击“测试模块命令”，观察到卡号信息框显示放置的卡片的卡号。

观察通讯记录窗口，命令帧数据结构如下表所示：

|  |  |  |
| --- | --- | --- |
| **字段** | **值** | **说明** |
| SOF | 0xAABB | 主机（如PC）与13.56MHz模块通信的命令帧起始字 |
| Lenth | 0xXXXX | 一帧命令的长度，从DEV\_ID（包含）开始到FCS（包含）结束为止的总长度 |
| Dev\_ID | 0x0000 | 设备编号，用于扩展使用 |
| CMD | 0xXXXX | 命令字，如寻卡命令字为0x0201 |
| Status/Flag | 0xXX | 状态字节或/和标志字节，主要在响应帧中出现 |
| VData | … | 可变字节的负载，发送时可以携带参数，接收时为响应数据 |
| FCS | 0xXX | 校验和，从LENGTH（不包含）开始到FCS（不包含）结束的所有字节的异或值 |

ISO 14443A协议指令是13.56MHz模块和14443A标签之间的通信协议指令，串口协议指令是13.56MHz模块与串口终端（如PC端、ARM端）之间的通信协议指令。

ISO 14443A协议指令如下表所示：

|  |  |  |
| --- | --- | --- |
| **指令标识** | **指令代码（Hex）** | **说明** |
| PICC\_REQIDL | 26 | 寻天线区内未进入休眠状态 |
| PICC\_REQALL | 52 | 寻天线区内全部卡 |
| PICC\_ANTICOLL1 | 93 | 防冲撞（一级串联） |
| PICC\_ANTICOLL2 | 95 | 防冲撞（二级串联） |
| PICC\_AUTHENT1A | 60 | 验证A密钥 |
| PICC\_AUTHENT1B | 61 | 验证B密钥 |
| PICC\_READ | 30 | 读块 |
| PICC\_WRITE | A0 | 写块 |
| PICC\_DECREMENT | C0 | 扣款 |
| PICC\_INCREMENT | C1 | 充值 |
| PICC\_RESTORE | C2 | 调块数据到缓冲区 |
| PICC\_TRANSFER | B0 | 保存缓冲区中数据 |
| PICC\_HALT | 50 | 休眠（暂停） |
| PICC\_RESET | E0 | 复位 |

串口协议指令如下表所示：

|  |  |  |
| --- | --- | --- |
| **指令标识** | **指令代码（Hex）** | **功能** |
| REQUEST\_A | 0201 | 寻卡A卡 |
| ANTICOLL | 0202 | A卡防冲突 |
| SELECT | 0203 | 锁定A卡 |
| SLEEP | 0204 | 休眠A卡 |
| AUTHENTICATION | 0207 | A卡密钥验证 |
| M1READ | 0208 | 读M1卡命令 |
| M1WRITE | 0209 | 写M1卡 |
| M1INITVAL | 020A | 初始化M1卡 |
| M1READVAL | 020B | 读取M1卡 |
| M1DECREAMENT | 020C | M1卡扣款 |
| M1INCREAMENT | 020D | M1卡充值 |
| ANT\_CONTROL | 010C | 天线控制 |
| GET\_HARDMODEL | 0104 | 获取版本号 |
| SET\_BAUDRATE | 0101 | 设置波特率 |

根据这三张表，可以分析出数据帧的含义。如第一条命令中VData字段为0x52，表示寻卡方式为寻所有卡，与实验操作一致。

休眠操作数据帧如下所示：



当休眠后，该卡将无法被寻到。

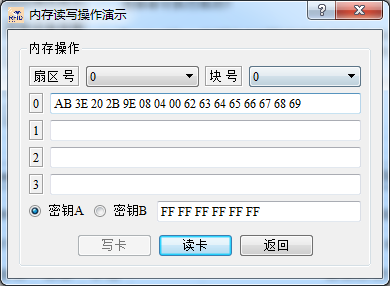
唤醒操作数据帧如下所示：



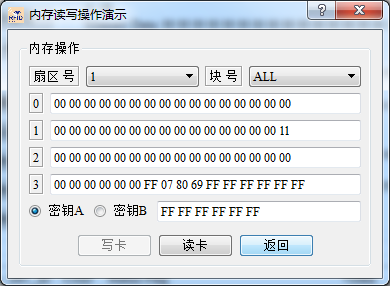
唤醒后可重新寻该卡。

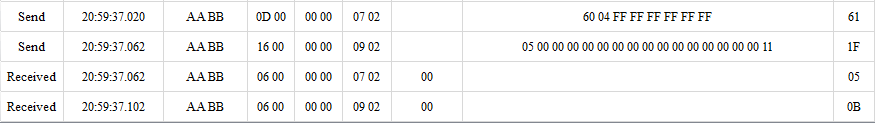
点击“内存读写”按钮，进入内存读写界面。

厂商信息存储在0区0块中：

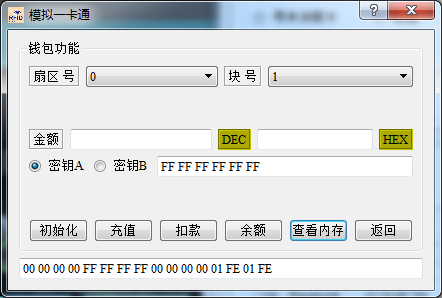


选择一块区域进行写操作，结果如下所示：

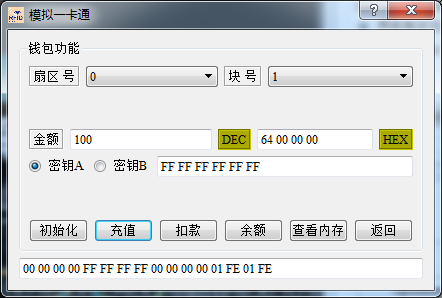




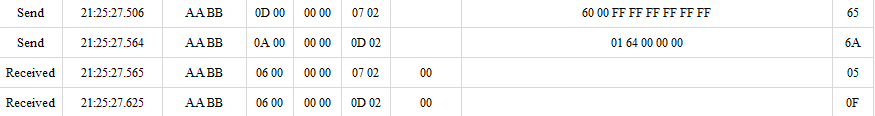
点击返回，回到主界面，点击“一卡通”按钮，进入一卡通应用模式。其界面如下所示：



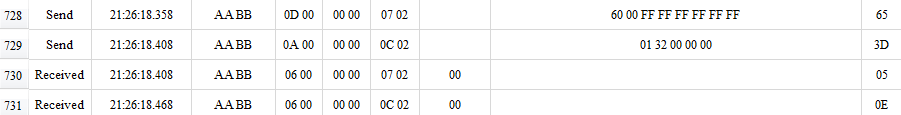
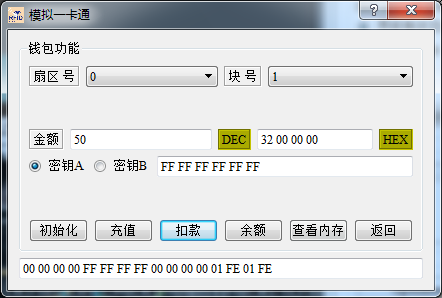
在金额框中输入金额，点击充值即可向卡里添加余额：



数据帧如下所示：



扣款操作同理：



## 2.3实验体会与总结

本次实验较为复杂，学习了HF 13.56MHz（ISO14443）模块的串口通信协议，掌握了Mifare-S50的存储结构和对其的操作方法。了解了14443的基本概念、国际标准、协议内容以及指令操作和接口标准，掌握了14443的相关命令操作。

通过本次实验我对于校园卡的工作机制有了一定的了解。更是体会到了RFID对生活带来的便利。

## 2.4核心源码说明

/\*\*

\* @brief MainWindow::Connect

\* 连接菜单点击事件

\*/

void MainWindow::**Connect**()

{

settingsDialog->setWindowTitle("读卡器连接配置");

settingsDialog->*exec*();

}

/\*\*

\* @brief MainWindow::updateConnect

\* 更新串口参数

\*/

void MainWindow::**updateConnect**()

{

serialPort->setPortName(settingsDialog->settings().name);

serialPort->setBaudRate(settingsDialog->settings().baudRate);

serialPort->setDataBits(settingsDialog->settings().dataBits);

serialPort->setParity(settingsDialog->settings().parity);

serialPort->setStopBits(settingsDialog->settings().stopBits);

serialPort->setFlowControl(settingsDialog->settings().flowControl);

if(serialPort->isOpen())

{

serialPort->*close*();

}

if(serialPort->*open*(QIODevice::ReadWrite))

{

ui->action\_connect->setEnabled(false);

ui->action\_Disconnect->setEnabled(true);

serialPortThread->setSerialPort(serialPort);

serialPortThread->setRetryTimes(RETRY\_20);

serialPortThread->start();

uint16 frameLen;

uint8 data[1];

uint8 \*p;

data[0] = RC632\_WORK\_MODEL\_1443A;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_CONFIG\_ISOTYPE,data,1);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

ui->statusBar->showMessage(tr("Connected to %1 : %2, %3, %4, %5, %6 success").arg(

settingsDialog->settings().name).arg(

settingsDialog->settings().baudRate).arg(

settingsDialog->settings().dataBits).arg(

settingsDialog->settings().parity).arg(

settingsDialog->settings().stopBits).arg(

settingsDialog->settings().flowControl));

}else{

QMessageBox::critical(this, tr("Error"), serialPort->errorString());

ui->statusBar->showMessage(tr("Open error"));

}

}

/\*\*

\* @brief MainWindow::Disconnect

\* 与读卡器断开连接，关闭串口，停止读写线程

\*/

void MainWindow::**Disconnect**()

{

if(serialPort->isOpen())

{

serialPort->*close*();

ui->action\_connect->setEnabled(true);

ui->action\_Disconnect->setEnabled(false);

ui->statusBar->showMessage(tr("Close Success"));

}

}

/\*\*

\* @brief MainWindow::ExitApplication

\* 退出按钮点击事件

\*/

void MainWindow::**ExitApplication**()

{

this->close();

}

/\*\*

\* @brief MainWindow::onOperationError

\* @param msg 发送的命令

\* 当串口写入数据无响应时调用该方法

\*/

void MainWindow::**onOperationError**(QString msg)

{

QMessageBox::warning(this,tr("温馨提示"),msg,QMessageBox::Yes);

}

//显示发送消息

void MainWindow::**onSendMessage**(char \*data, int frameLen)

{

Q\_UNUSED(data);

Q\_UNUSED(frameLen);

//这里没用到,这里就是提供一个接口,方便您在学习后使用

}

/\*\*

\* @brief MainWindow::on\_serialMsgreceived

\* @param bytes 接收到的数据字节数组

\* 串口接收消息后的处理槽函数

\*/

void MainWindow::**on\_serialMsgreceived**(QByteArray bytes){

M1356\_RspFrame\_t frame = m1356dll->M1356\_RspFrameConstructor(bytes);

if(frame.status.left(2) == "00")

{

if(frame.cmd.remove(" ") == "0102")//寻卡

{

uint16 frameLen;

quint8 buffer[1];

uint8 \*p;

memset(buffer, 0, 1);

buffer[0] = 0x04;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_ANTICOLL, buffer, 1);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

}

else if(frame.cmd.remove(" ") == "0202")//寻卡结果

{

uint16 frameLen;

quint8 buffer[4];

uint8 \*p;

memset(buffer, 0, 4);

QSTRING\_TO\_HEX(frame.vdata.remove(" "),buffer,4); // 卡号

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_SELECT,buffer,4);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

tagId = frame.vdata.remove(" ");

}

else if(frame.cmd.remove(" ") == "0302")

{

emit sendCardId(tagId);

}

}

else

{

if(frame.cmd.remove(" ") == "0102")//寻卡

{

QMessageBox::warning(this,tr("温馨提示"),tr("寻卡失败，请调整卡与读卡器的距离后再试！"),QMessageBox::Yes);

}

else if(frame.cmd.remove(" ") == "0202")//寻卡结果

{

QMessageBox::warning(this,tr("温馨提示"),tr("A卡防冲撞失败，请调整卡与读卡器的距离后再试！"),QMessageBox::Yes);

}

}

}

# 3 实验三 高频读写器实验ISO15693

## 3.1实验目的

通过本次实验了解高频读写器的基本原理，学会如何使用高频读写器，掌握 系统命令参数的意义和设置方式。

进一步加深对 ISO15693 协议下标签的存储结构以及 ISO15693 协议的理解。 通过读写器试验箱，掌握对 ISO15693 协议下标签读写操作以及 ISO15693 协议 标签存储结构的功能，并熟悉高频读写器 API 函数。

## 3.2实验内容及结果

1、完成 ISO15693 协议下的单标签和多标签手工寻卡和自动寻卡；

2、根据标签内存地址，完成ISO15693协议下标签指定地址的数据读写实验；

3、根据标签内存地址，完成 ISO15693 协议下标签指定地址范围的内存数 据读取实验；

4、ISO15693 协议的命令，完成标签静默状态设置、重置到准备状态、标签 选择命令实验；

5、完成 ISO15693 协议下标签 DSFID、AFI 的读写和块安全位的读取实；

6、熟悉和了解高频 HF1356M 15693 开发实例，掌握高频读写器 API 函数，并通过编程实现 ISO15693 协议下标签的读写功能。

**操作过程：**

1. 准备好硬件，RFID 13.56MHz模块（ISO15693协议）和15693标签。
2. 连接好RFID平台串口3，另一端连接电脑串口，打开教学平台电源开关，切换至RFID 13.56MHz模块。
3. 打开RFID-DEMO模块测试软件，切换到15693页面。
4. 连接串口，波特率为19200。
5. 开始功能测试。

手动寻卡如下图所示：



自动寻卡：



ISO15693协议指令如下表：

|  |  |  |
| --- | --- | --- |
| **指令标识** | **指令代码（Hex）** | **类型** |
| 寻卡 | 01 | 强制的 |
| 静止 | 02 | 强制的 |
| RFU | 03~1F | 强制的 |
| 读单一块 | 20 | 可选的 |
| 写单一块 | 21 | 可选的 |
| 锁定块 | 22 | 可选的 |
| 读多重块 | 23 | 可选的 |
| 写多重块 | 24 | 可选的 |
| 选择 | 25 | 可选的 |
| 重启准备 | 26 | 可选的 |
| 写AFI | 27 | 可选的 |
| 锁定AFI | 28 | 可选的 |
| 写DSFID | 29 | 可选的 |
| 锁定DSFID | 2A | 可选的 |
| 获取系统信息 | 2B | 可选的 |
| 获得多重块安全状态 | 2C | 可选的 |
| RFU | 2D~9F | 可选的 |
| IC生产厂家确定 | A0~DF | 自定义的 |
| IC生产厂家确定 | E0~FF | 专用的 |

串口协议如下表：

|  |  |  |
| --- | --- | --- |
| **指令标识** | **指令代码（Hex）** | **功能** |
| INVENTORY16 | 1000 | 寻卡（可寻多张） |
| INVENTORY | 1001 | 寻卡（不建议直接使用） |
| STAY\_QUIET | 1002 | 设置静默态 |
| SELECT | 1003 | 选择 |
| RESET\_TO\_READY | 1004 | 重置到准备态 |
| READ\_SM | 1005 | 读单个或者多个数据块 |
| WRITE\_SM | 1006 | 写单个数据块 |
| LOCK\_BLOCK | 1007 | 锁定数据块 |
| WRITE\_AFI | 1008 | 写应用族标志 |
| LOCK\_AFI | 1009 | 锁定应用族标志 |
| WRITE\_DSFID | 100A | 写数据存储格式标志 |
| LOCK\_DSFID | 100B | 锁定数据存储格式标志 |
| GET\_SYSINFO | 100C | 获取系统信息 |
| GET\_MULTIBLOCK\_SECURITY | 100D | 获取多个块的安全状态 |
| GET\_HARDMODEL | 0104 | 获取版本号 |
| SET\_BAUDRATE | 0101 | 设置波特率 |

根据上表可解读数据帧。

读写单个数据块：



多个数据块读取：



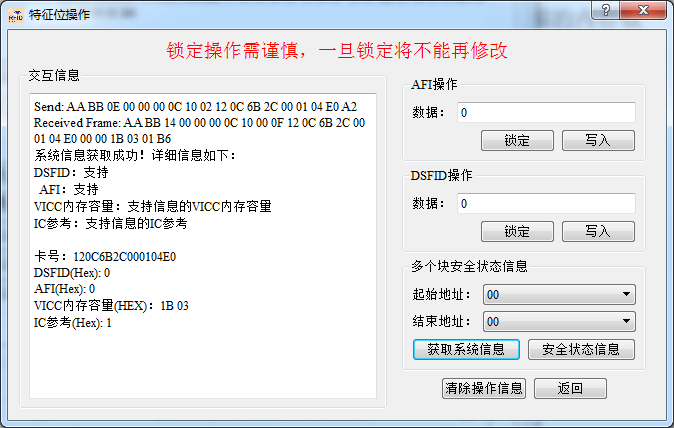
保持静默状态：静默状态下卡片不会被发现。



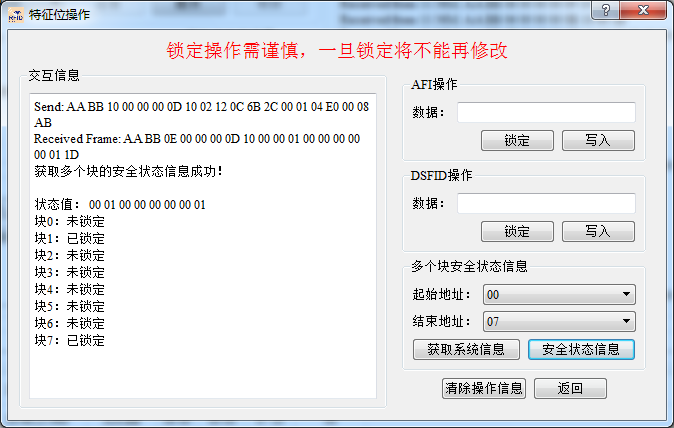
重置到准备状态：



获取系统信息：



获取多个块安全状态：



## 3.3实验体会与总结

本次实验较为和上一次区别不大，学习了HF 13.56MHz（ISO15693）模块的串口通信协议，掌握了15693标签的存储结构和对其的操作方法。了解了15693的基本概念、国际标准、协议内容以及指令操作和接口标准，掌握了15693的相关命令操作。

## 3.4核心源码说明

BooksManage::**BooksManage**(QWidget \*parent) : QWidget(parent)

{

QString LabelName[] = {"卡号：", "书名：", "作者：", "出版社：", "总数（本）", "剩余（本）"};//标签文本

QString ButtonName[] = {"添加", "删除", "修改", "搜索"};//按钮文本

QVBoxLayout \*MainLayout = new QVBoxLayout();//主布局

QHBoxLayout \*ButtonLayout = new QHBoxLayout();//按钮布局

QHBoxLayout \*EditLayout = new QHBoxLayout();//文本框布局

QHBoxLayout \*TableLayout = new QHBoxLayout();//表格布局

QGroupBox \*BookTable = new QGroupBox();//表格区域

QGroupBox \*BookInfo = new QGroupBox();//信息

sql = new Sqlite();

for(int i = 0; i < Edit\_Count\_BOOKS; i++) //初始化文本框和标签

{

Edit[i] = new QLineEdit();

Label[i] = new QLabel(LabelName[i]);

EditLayout->addWidget(Label[i]);//将文本框和标签添加到布局中

EditLayout->addWidget(Edit[i]);

}

QString pattern("[A-Fa-f9-0]\*");

QRegExp regExp(pattern);

Edit[ID\_Books]->setValidator(new QRegExpValidator(regExp, this));

pattern="[9-0]{3}";

regExp.setPattern(pattern);

Edit[Count\_Books]->setValidator(new QRegExpValidator(regExp, this));

Edit[Residue\_Books]->setValidator(new QRegExpValidator(regExp, this));

BookInfo->setLayout(EditLayout);//设置信息组合框的布局

for(int i = 0; i < Button\_Count\_BOOKS; i++)//初始化按钮

{

Button[i] = new QPushButton();

Button[i]->setText(ButtonName[i]);

ButtonLayout->addWidget(Button[i]);//按钮添加到布局中

}

ButtonLayout->addStretch(0);

ButtonLayout->setSpacing(20);

Table = new QTableWidget();

Table->setColumnCount(Table\_Column\_BOOKS);

Table->setSelectionBehavior ( QAbstractItemView::SelectRows);//选中整行

Table->setEditTriggers ( QAbstractItemView::NoEditTriggers );//不可编辑

Table->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);//列宽度自适应

TableLayout->addWidget(Table);

BookTable->setLayout(TableLayout);

BookTable->setTitle("图书列表");

MainLayout->addWidget(BookInfo);

MainLayout->addLayout(ButtonLayout);

MainLayout->addWidget(BookTable);

MainLayout->setSpacing(10);

this->setLayout(MainLayout);

SetSlot();

}

void BooksManage::**SetSlot**()//设置槽函数

{

connect(Button[Add\_Books],SIGNAL(clicked()),this,SLOT(add\_books()));//添加按钮连接槽函数add\_books()

connect(Button[Delete\_Books],SIGNAL(clicked()),this,SLOT(delete\_books()));//删除按钮连接槽函数delete\_books()

connect(Button[Updata\_Books],SIGNAL(clicked()),this,SLOT(updata\_books()));//修改按钮连接槽函数updata\_books()

connect(Button[Select\_Books],SIGNAL(clicked()),this,SLOT(select\_books()));//查找按钮连接槽函数select\_books()

connect(Table,SIGNAL(cellClicked(int,int)),this,SLOT(get\_table\_line(int, int)));//表格单击事件连接槽函数get\_table\_line(int, int)

}

void BooksManage::**add\_books**()//添加按钮槽函数

{

int residue;//图书的剩余数量

/\*文本框为空时显示错误提示\*/

QString LabelName[] = {"卡号：", "书名：", "作者：", "出版社：", "总数（本）"};

for(int i = 0; i < Edit\_Count\_BOOKS-1; i++)

{

if(Edit[i]->text().isEmpty())

{

QMessageBox::warning(NULL, "warning", LabelName[i]+"不能为空！", QMessageBox::Yes, QMessageBox::Yes);

return;

}

}

if (sql->SelectUser(Edit[ID\_Books]->text()).next())

{

QMessageBox::warning(NULL, "warning", "卡号已经注册为用户！", QMessageBox::Yes, QMessageBox::Yes);

return;

}

if (Edit[Residue\_Books]->text().toInt() > Edit[Count\_Books]->text().toInt())

{

QMessageBox::warning(NULL, "warning", "剩余数量不可以超出总数！", QMessageBox::Yes, QMessageBox::Yes);

return;

}

/\*不填写剩余数量默认为总数量\*/

if (Edit[Residue\_Books]->text().isEmpty())

{

residue = Edit[Count\_Books]->text().toInt();

}

else

{

residue = Edit[Residue\_Books]->text().toInt();

}

//向数据库中添加书籍

bool ret = sql->InsertBooks(Edit[ID\_Books]->text(),Edit[Name\_Books]->text(),Edit[Author\_Books]->text(),Edit[PublishingHouse\_Books]->text(),Edit[Count\_Books]->text().toInt(),residue);

if(!ret)

{

QMessageBox::warning(NULL, "warning", "添加失败，卡号已存在！", QMessageBox::Yes, QMessageBox::Yes);

return;

}

QMessageBox::warning(NULL, "warning", "添加成功！", QMessageBox::Yes, QMessageBox::Yes);

ClearEdit(); //清空文本框

ShowTable(sql->SelectBooks());//更新表格

}

# 4 实验四 超高频读写器实验

## 4.1实验目的

通过本次实验了解超高频读写器的基本原理，学会如何使用超高频读写器，掌握超高频读写器和标签参数的含义和设置方法。

进一步加深对 Gen2 协议下标签的存储结构以及 Gen2 协议的理解。通过读写器试验箱，掌握对 Gen2 协议下标签读写操作，并熟悉超高频读写器 API 函数的 调用。

## 4.2实验内容及结果

1、学会通过试验箱对 Gen2 协议下标签指定存储区的数据读写;

2、理解和掌握 Gen2 协议下标签存储器结构的特点及含义;

3、通过多次设置读写器功率，不断移动电子标签与读写器之间的距离，分 析理解读写器功率和频率对电子标签读写的影响;

4、理解和掌握访问密码的用途和使用方法;

5、掌握超高频读写器 API 函数的调用方法，并能够通过编程实现对 Gen2 协议下标签数据的读写控制。

**操作过程：**

1. 准备好硬件，RFID 900MHz模块和Gen2标签。
2. 连接好RFID平台的串口3，另一端连接电脑，切换到900MHz模块。
3. 打开RFID-DEMO模块测试软件。
4. 波特率设为57600，连接。

**操作过程：**

寻卡，单步识别如下图所示：

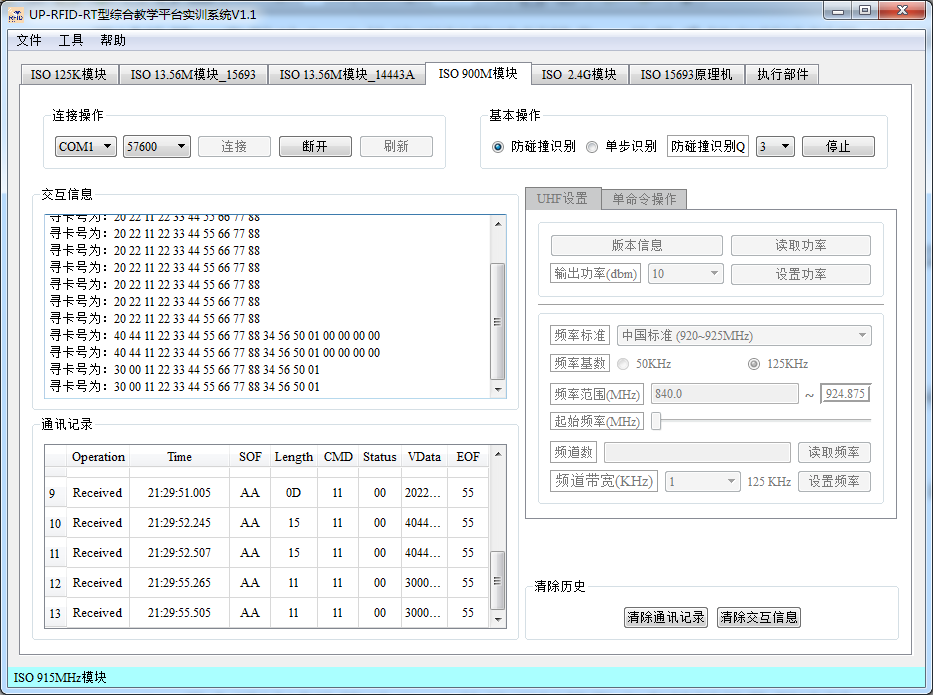


串口协议如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **指令标识** | **指令代码（Hex）** | **功能** | **响应等待时间（ms）** |
| UHFCMD\_GET\_STATUS | 00 | 询问状态 | 200 |
| UHFCMD\_GET\_POWER | 01 | 读取功率 | 200 |
| UHFCMD\_SET\_POWER | 02 | 设置功率 | 200 |
| UHFCMD\_GET\_FRE | 05 | 读取频率 | 200 |
| UHFCMD\_SET\_FRE | 06 | 设置频率 | 200 |
| UHFCMD\_GET\_VERSION | 07 | 读取版本信息 | 200 |
| UHFCMD\_INVENTORY | 10 | 识别标签（单标签识别） | 200 |
| UHFCMD\_INVENTORY\_ANTI | 11 | 识别标签（防碰撞识别） | 200 |
| UHFCMD\_STOP\_GET | 12 | 停止操作 | 200 |
| UHFCMD\_READ\_DATA | 13 | 读取标签数据 | 200 |
| UHFCMD\_WRITE\_DATA | 14 | 写入标签数据 | 200 |
| UHFCMD\_ERASE\_DATA | 15 | 擦除标签数据 | 200 |
| UHFCMD\_LOCK\_MEM | 16 | 锁定标签 | 200 |
| UHFCMD\_KILL\_TAG | 17 | 销毁标签 | 200 |
| UHFCMD\_INVENTORY\_SINGLE | 18 | 识别标签（单步识别） | 200 |
| UHFCMD\_WIEGAND\_INVENTORY | 19 | 韦根识别 | 200 |
| UHFCMD\_SINGLE\_READ\_DATA | 20 | 读取标签数据（不指定UII） | 200 |
| UHFCMD\_SINGLE\_WRITE\_DATA | 21 | 写入标签数据（不指定UII） | 200 |

根据表格可以解读命令帧。

防碰撞识别：



查看并设置频率功率：





读取数据：



写入数据：



擦除数据：



## 4.3实验体会与总结

本次实验较为简单，学习了UHF 900MHz模块的串口通信协议，掌握了900MHz标签的存储结构和对其的操作方法。了解了基本概念、国际标准、协议内容以及指令操作和接口标准，掌握了900MHz的相关命令操作。

# 5 实验五 RFID综合应用实验

## 5.1需求分析

### 1、基于 RFID 技术校园卡综合应用系统设计与实现

采用北京博创RFID实验箱模拟地铁收费系统的读卡设备、支持14443A协议的S50卡（5张）模拟校园卡（相当于一个电子钱包）。

用户首次申请领用该卡时，保存个人手机号，并对卡进行初始化和充值。将 卡与个人手机关联、姓名关联起来（采用实名制，便于挂失）。

统一将某扇区0块作为电子钱包，将某扇区1块、2块保存个人信息，将某 扇区作为存储空间专门保存该卡最近的5次交易记录。

一旦该卡丢失，马上通知管理员挂失，系统自动将该卡设置为未激活状态，

锁定消费。当然，每次消费时，一定要判断该卡是否处于激活状态。

假设用户利用该卡在不同用户食堂、不同超市购买不同商品时，刷卡一次， 按照预先设定的收费标准实行自动扣款，在卡内记录最近五条消费明细，消费明 细同时写入系统数据库表中。

假定读写器设备与上位机始终保持联系，上位机与数据库服务器始终保持联 系。

### 2、系统功能

(1) 用户发卡管理；

(2) 校园卡充值管理；

(3) 用户毕业时的销卡管理，清除卡内消费数据以及个人手机号，退还余款；

(4) 用户消费时自动刷卡扣费，并在卡内和数据库中同时保存消费记录信息，假定记录信息不超过5条;消费记录保存在的扇区依据学号不同而不同(学号尾 数为奇数的学生，消费记录保存区域从 2 扇区开始;学号尾数为偶数的学生，消 费记录保存区域从 3 扇区开始)；

(5) 校园卡消费明细查询。显示消费明细时，必须同时显示用户手机号、电 子标签ID号；

(6) 卡内余额查询。

### 3、系统表结构

(1) 用户基本信息表（卡号、姓名、手机号、...、是否激活）；

(2) 用户消费信息表（卡号、姓名、商品编号、购买时间、商品数量、扣款 金额、...）；

(3) 商品资费信息表（商品编号、单价、单位、...）；

(4) 用户充值信息表（卡号、充值金额、充值时间、充值地点、...）；

(5) 商户信息表（商户编号、商户名称、位置、...，商户名称:东一食堂、喻 园教工超市、...）。

### 4、开发工具选项

采用C/C++或其他编程语言，Qt为平台，Sqlite作为数据库管理平台。

## 5.2系统详细设计

### 1、数据库表设计：

(1) 用户信息表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 编号 | Varchar primary key | 用户ID |
| 姓名 | Varchar | 用户的姓名 |
| 身份类型 | Varchar | 学生、老师 |
| 备注信息 | Varchar | 额外的信息字段 |

(2) 注册表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 卡号 | Varchar primary key | 注册时发放的卡号 |
| 用户编号 | Varchar | 用户表中的用户编号 |
| 时间 | Varchar | 注册时的时间 |
| 备注信息 | Varchar | 额外的信息字段 |

(3) 注销表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 卡号 | Varchar primary key | 注销用户的卡号 |
| 时间 | Varchar | 注销的时间 |
| 备注信息 | Varchar | 额外的信息字段 |

(4) 管理员表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 用户名 | Varchar primary key | 管理员的用户名 |
| 密码 | Varchar | 登录密码 |
| 备注信息 | Varchar | 额外的信息字段 |

(5) 消费记录表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 卡号 | Varchar primary key | 当前消费的卡ID |
| 时间 | Varchar | 消费时间 |
| 地点 | Varchar | 消费地点 |
| 读卡器编号 | Varchar | 刷卡机的ID |
| 金额 | Varchar | 消费的金额 |
| 备注信息 | Varchar | 额外的信息字段 |

(6) 充值记录表：

|  |  |  |
| --- | --- | --- |
| 字段 | 字段类型 | 描述 |
| 卡号 | Varchar primary key | 当前充值的卡ID |
| 时间 | Varchar | 充值的时刻 |
| 原有金额 | Varchar | 卡内剩余金额 |
| 充值金额 | Varchar | 本次充值金额 |
| 当前金额 | Varchar | 充值后的金额 |
| 备注信息 | Varchar | 额外的信息字段 |

数据库表的代码实现如用户表为例：

PersonTableModel::**PersonTableModel**(QObject \*parent) : QSqlTableModel(parent)

{

tableName = TABLE\_NAME\_PERSON;

header<<QObject::trUtf8("编号")<<QObject::trUtf8("姓名")<<

QObject::trUtf8("身份类型")<< QObject::trUtf8("信息备注");

}

/\*\*

\* @brief PersonTableModel::createTable

\* @param tableName 数据块表名称

\* 用于创建Person表

\*/

void PersonTableModel::**createTable**()

{

QSqlQuery query;

QString str;

str = tr("create table ") + tableName + tr(" ( ");

str += header.at(0) + tr(" varchar PRIMARY KEY not null, ");

str += header.at(1) + tr(" varchar, ");

str += header.at(2) + tr(" varchar, ");

str += header.at(3) + tr(" varchar) ");

qDebug()<<"Sql: " << str.toUtf8().data();

bool ret = query.exec(str);

if(ret == true){

qDebug()<<tableName<<QObject::tr(" table create success");

}

else{

qDebug()<<tableName<<QObject::tr(" table create failed");

}

}

/\*\*

\* @brief PersonTableModel::restore

\* 绑定表名

\*/

void PersonTableModel::**bindTable**(void)

{

*setTable*(tableName);

*select*();

}

/\*\*

\* @brief PersonTableModel::findRecord

\* @param personId 人员编号

\* @return QSqlRecord型记录集

\* 根据人员编号查找记录

\*/

int PersonTableModel::**findRecordById**(const QString personId)

{

int count = *rowCount*();

for(int row=0; row < count; row++){

if(*data*(*index*(row, 0)).toString() == personId)

return row;

}

return -1;

}

/\*\*

\* @brief PersonTableModel::findRecord

\* @param personId 人员编号

\* @return QSqlRecord型记录集

\* 根据人员编号查找记录

\*/

QSqlRecord PersonTableModel::**findRecordByName**(const QString userName)

{

*setFilter*(QObject::tr("姓名= '%1'").arg(userName)); //根据姓名进行筛选

*select*();

return record();

}

/\*\*

\* @brief PersonTableModel::insertRecords

\* @param personId 人员编号

\* @param personName 人员名称

\* @param personType 身份类别

\* @param remark 备注信息

\* @return 成功返回true，失败返回false

\* 向表格中插入记录

\*/

bool PersonTableModel::**insertRecords**(QString personId,QString personName, QString personType, QString remark)

{

QSqlQuery query;

QString str;

str = tr("insert into ") + tableName + tr(" values( \"%1\" , \"%2\" , \"%3\" , \"%4\" ) ")

.arg(personId).arg(personName).arg(personType) .arg(remark);

qDebug()<<"Sql: " << str.toUtf8().data();

return query.exec(str);

}

/\*\*

\* @brief PersonTableModel::deleteRecords

\* @param row 待删除的行

\* @return 如果删除成功返回true，否则false

\* 删除一行记录

\*/

bool PersonTableModel::**deleteRecords**(int row)

{

return removeRow(row);

}

其他的表与用户表类似，因此不详述。

### 2、数据库表管理器

多个表通过一个数据库管理对象DBManager来管理。代码如下：

DBManager::**DBManager**(QObject \*parent) : QObject(parent)

{

if(this->createDB(DATABASE\_NAME))

{

if(!this->tableExist(TABLE\_NAME\_PERSON))

{

PersonTableModel p;

p.createTable();

}

if(!this->tableExist(TABLE\_NAME\_RECORD))

{

RecordTableModel r;

r.createTable();

}

if(!this->tableExist(TABLE\_NAME\_REGISTER))

{

RegisterTableModel r;

r.createTable();

}

if(!this->tableExist(TABLE\_NAME\_WRITEOFF))

{

WriteOffTableModel w;

w.createTable();

}

if(!this->tableExist(TABLE\_NAME\_ADMIN))

{

AdminTableModel a;

a.createTable();

}

if(!this->tableExist(TABLE\_NAME\_RECHARGE))

{

RechargeTableModel r;

r.createTable();

}

}

}

/\*\*

\* @brief RecordTableModel::createDB

\* @param dbName 数据库名称

\* 用于创建数据库

\*/

bool DBManager::**createDB**(const QString &dbName)

{

db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName(dbName);

if(db.open()){

qDebug()<<dbName<<" create success";

return true;

}

else{

qDebug()<<dbName<<" create failed!";

return false;

}

}

/\*\*

\* @brief DBManager::tableExist

\* @param tableName 表名

\* @return 如果存在返回true，否则返回false

\* 用于判断表是否存在

\*/

bool DBManager::**tableExist**(const QString &tableName)

{

int count = 0 ;

QString sqlText = QObject::tr("select count(\*) from sqlite\_master where type='table' and name='%1'").arg(tableName);

QSqlQuery query;

query.exec(sqlText);

if(query.next())

count = query.value(0).toInt();

if(count > 0)

return true;

else

return false;

}

/\*\*

\* @brief DBManager::getTableNames

\* @return 数据库表集合

\* 获取数据库所有表的表名

\*/

QStringList DBManager::**getTableNames**()

{

QStringList tables;

QString sqlText = QObject::tr("select name from sqlite\_master where type='table' order by name;");

QSqlQuery query;

query.exec(sqlText);

while(query.next())

{

tables << query.value(0).toString();

}

return tables;

}

/\*\*

\* @brief DBManager::dbClose

\* 关闭数据库

\*/

void DBManager::**dbClose**()

{

if(db.isOpen())

db.close();

}

static DBManager \*dbManager;

/\*\*

\* @brief DB\_Init

\* 创建数据库管理对象实体，初始化数据库

\*/

void **DB\_Init**()

{

dbManager = new DBManager();

}

/\*\*

\* @brief DB\_Close

\* 关闭数据库

\*/

void **DB\_Close**()

{

dbManager->dbClose();

}

/\*\*

\* @brief getTableNames

\* @return QStringList型的表名集合

\* 获取数据库所有表的表名

\*/

QStringList **getTableNames**()

{

return dbManager->getTableNames();

}

DB\_Init在main.cpp里调用，负责数据库的初始化，实际上就是创建了一个DBManager的实例。

在创建数据库实例时，检测各个表是否被创建，如果没有则创建之。调用tableExist方法，采用sql语句直接查找sqlite\_master里是否有指定的表名的表。如果tableExist返回false，则创建该表。DB\_Init完成后，程序便可使用数据库。

### 3、菜单点击事件

在Qt中，每一个菜单选项都是QAction，在主窗口中用connect语句将菜单和对应的时间槽函数连接起来，代码如下：

connect(settingsDialog,SIGNAL(applySettings()),this,SLOT(updateConnect()));

connect(ui->action\_about,SIGNAL(triggered(bool)),this,SLOT(About()));

connect(ui->action\_changepwd,SIGNAL(triggered(bool)),this,SLOT(ChangePasswd()));

connect(ui->action\_export,SIGNAL(triggered(bool)),this,SLOT(ExportTable()));

connect(ui->action\_login,SIGNAL(triggered(bool)),this,SLOT(Login()));

connect(ui->action\_connect,SIGNAL(triggered(bool)),this,SLOT(Connect()));

connect(ui->action\_Disconnect,SIGNAL(triggered(bool)),this,SLOT(Disconnect()));

connect(ui->action\_persontable,SIGNAL(triggered(bool)),this,SLOT(ViewTables()));

connect(ui->action\_recordtable,SIGNAL(triggered(bool)),this,SLOT(ViewTables()));

connect(ui->action\_register,SIGNAL(triggered(bool)),this,SLOT(Register()));

connect(ui->action\_registtable,SIGNAL(triggered(bool)),this,SLOT(ViewTables()));

connect(ui->action\_writeoff,SIGNAL(triggered(bool)),this,SLOT(WriteOff()));

connect(ui->action\_writeofftable,SIGNAL(triggered(bool)),this,SLOT(ViewTables()));

connect(ui->action\_consume,SIGNAL(triggered(bool)),this,SLOT(ViewConsume()));

connect(ui->action\_mainPage,SIGNAL(triggered(bool)),this,SLOT(viewMainPage()));

connect(ui->actionquit,SIGNAL(triggered(bool)),this,SLOT(ExitApplication()));

connect(ui->action\_recharge,SIGNAL(triggered(bool)),this,SLOT(Recharge()));

connect(ui->action\_recharges,SIGNAL(triggered(bool)),this,SLOT(ViewTables())); connect(serialPortThread,SIGNAL(sendMsg(char\*,int)),this,SLOT(onSendMessage(char\*,int))); connect(serialPortThread,SIGNAL(wirteMsgError(QString)),this,SLOT(onOperationError(QString)));

connect(serialPortThread,SIGNAL(receivedMsg(QByteArray)),this,SLOT(on\_serialMsgreceived(QByteArray)));

当鼠标点击菜单选项的时候会发出一个triggered信号，这里自定义槽函数去实现这一事件。以action\_login为例，实现代码如下：

void MainWindow::**Login**()

{

LoginPage \*loginpage = new LoginPage(this,&adminName);

loginpage->setWindowTitle("管理员登录");

loginpage->*exec*();

if(this->adminName.count() >= 4)

{

ui->statusBar->showMessage(tr("登录成功,用户名：%1").arg(adminName));

this->IsLogin = true;

}

else

{

ui->statusBar->showMessage(tr("登录失败"));

this->IsLogin = false;

}

delete loginpage;

}

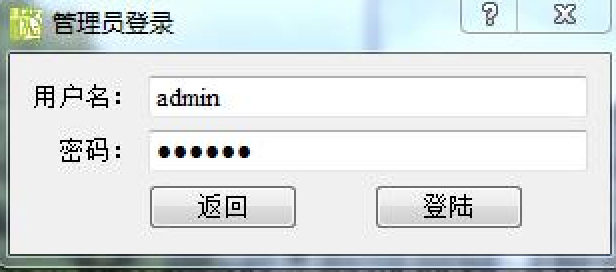
实现所有菜单的槽函数，分别有：登录、连接、断开、退出、用户注册、用户注销、用户充值、修改密码、查询记录、模拟消费、返回主页。一一实现上述功能。

## 5.3系统实现与系统测试

打开工程，运行程序，主界面如下图所示：



登录管理员账号，用户名为admin，密码为123456，如下：

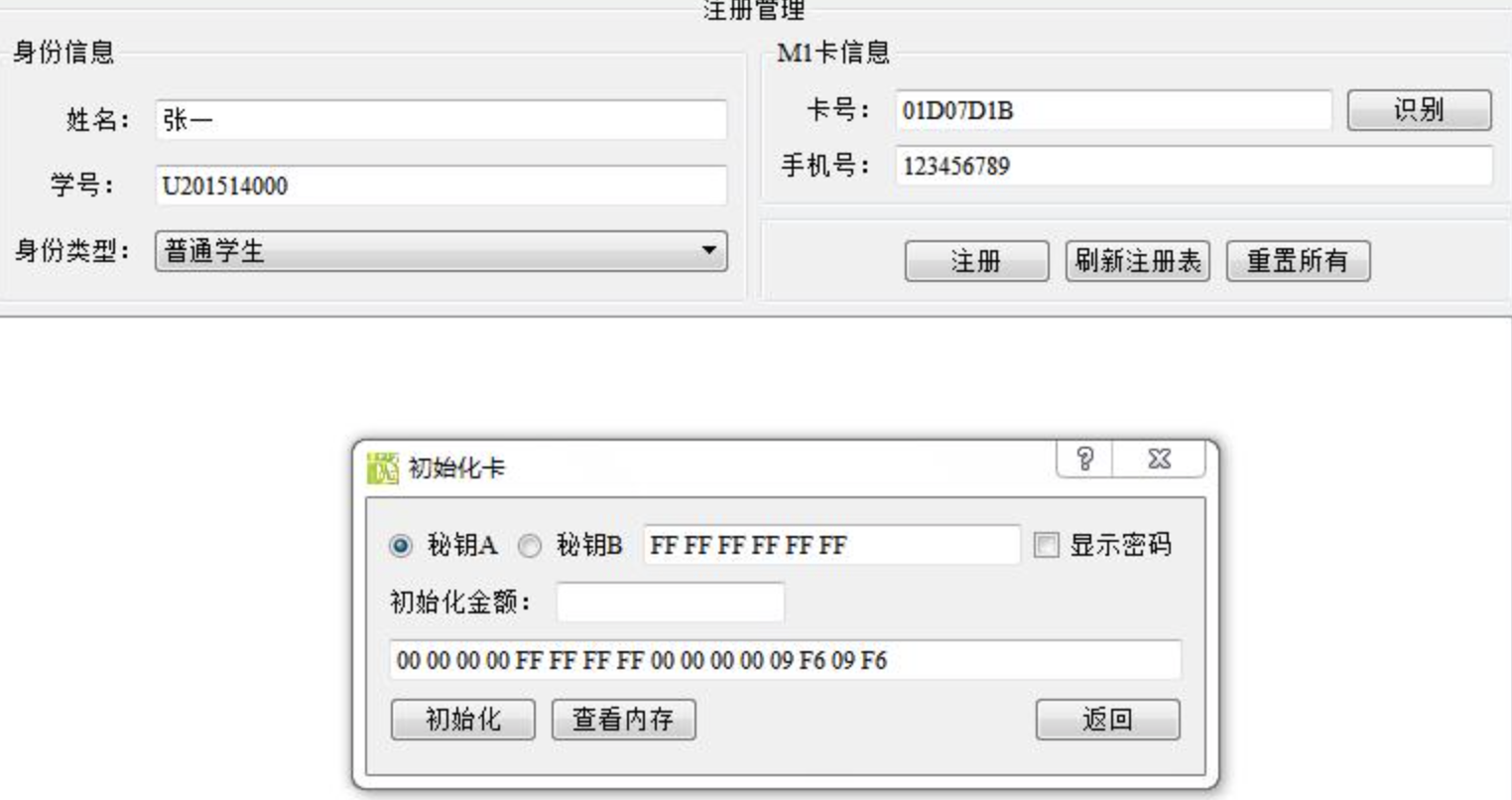


将设备连接上电脑后，点击“连接”按钮，配置如下图所示：

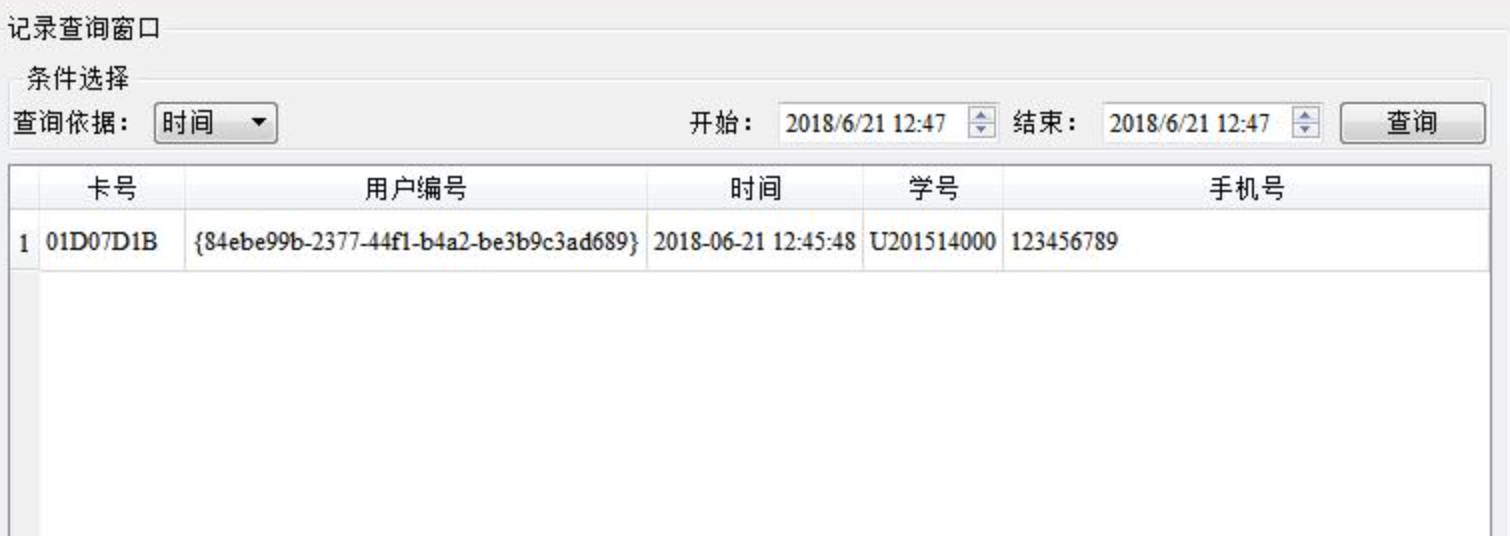


点击“应用”即连接完毕。

用户注册如下图所示，将卡片置于读写器上，点击识别按钮，即可显示卡号，输入身份信息，点击注册，即可完成注册。



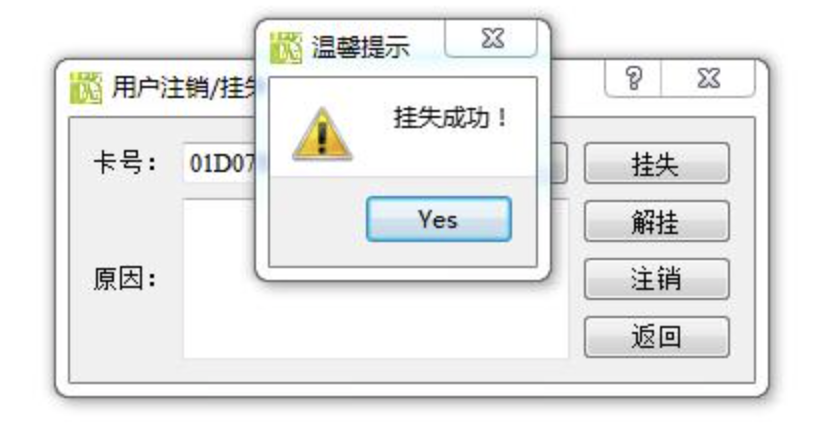
查看注册表，即可查询到刚才的注册记录：



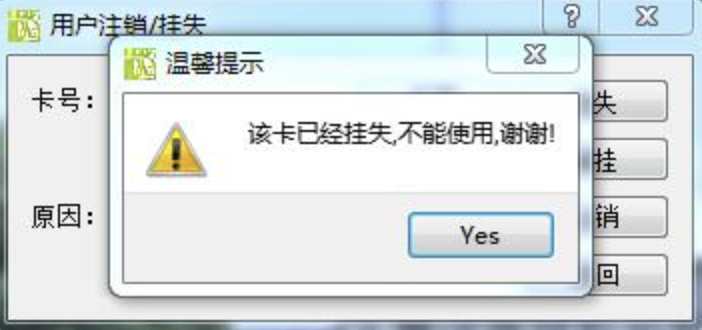
进入消费模拟界面，使用刚才注册的卡消费，初始金额为99元。

点击识别按钮出现卡号，在食堂消费兰州拉面，金额3元。消费成功后界面上方会显示一行信息，余额为96元。

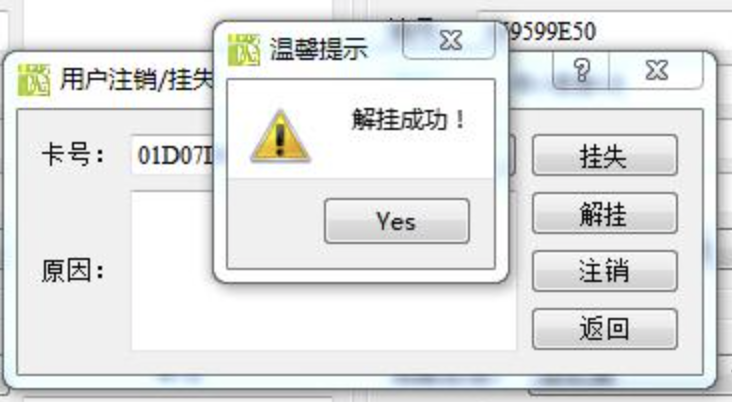
测试挂失功能，卡片挂失后，该卡无法继续使用直到解挂为止。点击用户注销/挂失菜单选项，点击识别按钮出现卡号，点击挂失，提示挂失成功。



在模拟消费中使用挂失的卡进行消费操作，将会提示该卡已挂失无法使用：



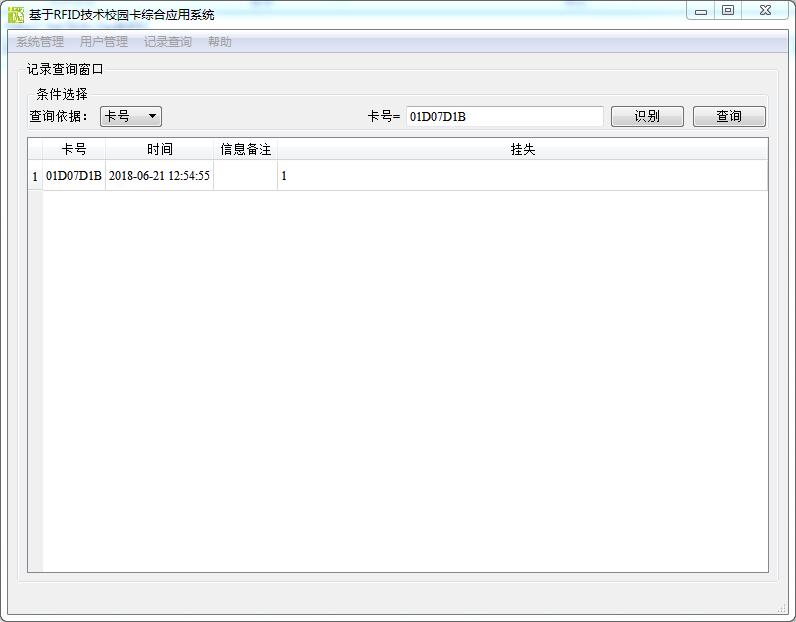
回到挂失的页面进行解挂操作。识别卡片后点击解挂提示解挂成功：



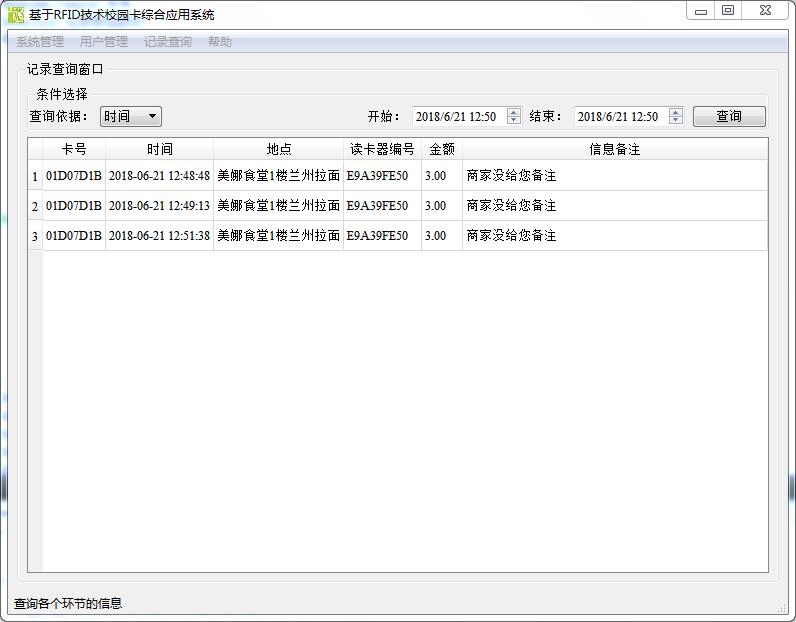
解挂成功后卡片可以继续消费使用：



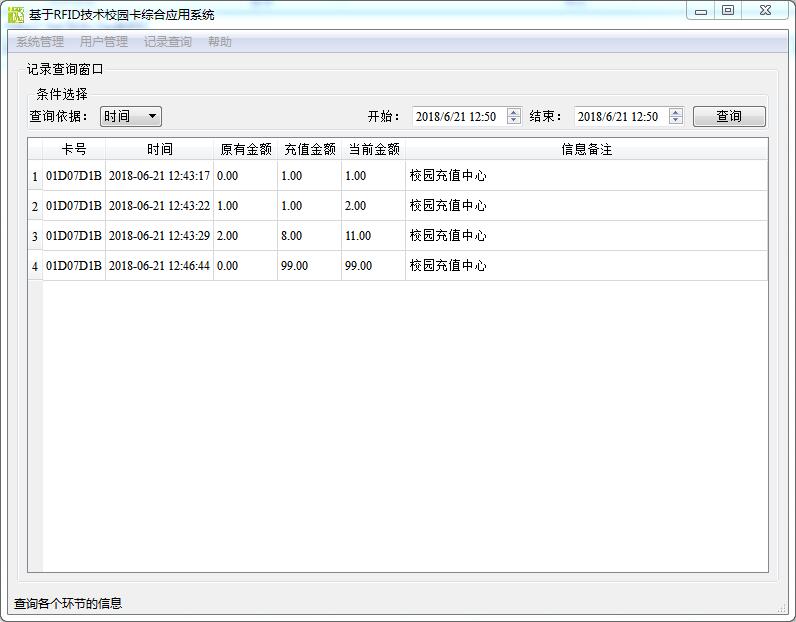
挂失的原理是将卡上的一个存储位设定为挂失位，当该位为1时表示处于挂失状态，为0时为解挂状态。挂失记录如下所示：



消费记录：



充值记录：



对卡片内存的写操作如下代码：

//写入授权

void ConsumePage::**writeauthentication**()

{

uint16 frameLen;

quint8 buffer[8];

uint8 \*p;

//wait

QTime dieTime = QTime::currentTime().addMSecs(1000);

while( QTime::currentTime() < dieTime )

QCoreApplication::processEvents(QEventLoop::AllEvents, 100);

buffer[0] = 0x60;

RegisterTableModel \*registerTableModel = new RegisterTableModel(this);

registerTableModel->bindTable();

QString t = ui->lineEdit\_tagId->text();

int num = registerTableModel->findstunum(t);

if(num == 1) //学号为奇数从2号扇区开始

buffer[1] = 0x0A; // 绝对块号

else //学号为偶数从3号扇区开始

buffer[1] = 0x0C;

for(int i = 2 ; i < 8 ; i ++)

buffer[i] = 0xFF;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_AUTHENTICATION,buffer,8);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

}

//写入数据

void ConsumePage::**write**()

{

uint16 frameLen;

quint8 buffer[17];

uint8 \*p;

//取出学号

RegisterTableModel \*registerTableModel = new RegisterTableModel(this);

registerTableModel->bindTable();

QString t = ui->lineEdit\_tagId->text();

int num = registerTableModel->findstunum(t);

//进行写操作

if(num == 1) //学号为奇数从2号扇区开始

buffer[0] = 0x0A; // 绝对块号

else //学号为偶数从3号扇区开始

buffer[0] = 0x0D;

for(int i = 1 ; i < 16 ; i ++)

buffer[i] = 0x00;

buffer[16] = 0x60;

p = m1356dll->RC632\_SendCmdReq(RC632\_CMD\_M1WRITE,buffer,17);

frameLen = BUILD\_UINT16(p[0], p[1]);

serialPortThread->writeData((char \*)(p + 2 ),frameLen);

}

设置一个包含17个qunit8类型元素的数组buffer[17]，其中第0位为块号，其他16位为数据，通过方法RC632\_SendCmdReq想该数组发送给卡片。即可完成写操作。

## 5.4总结

本次实验为RFID综合设计实验，需要动用之前所学到的很多知识点和方法方法，实现一个可以真正使用的校园卡一卡通系统。除了需要掌握HF 13.56MHz模块的串口通信协议，还需要对Qt有很熟练的掌握：Qt多线程、Qt信号槽机制、sqlite的使用。好在有了样例工程的参考，实验的重点落在的理解代码上来。

在本次实验过程中，我对样例工程的理解花费了较多时间。由于之前对于Qt不是很熟练，如何用Qt设计一套UI界面对我来说算是一件费时的事情，再加上Qt特有的信号槽机制也需要去学习掌握。从而导致虽然有样例工程，自己也花费了大量时间去完成它。

最后通过这次实验，我对于RFID系统有了进一步的掌握，对于编写Qt工程变得更加熟练。总的来说这次实验也是让我受益匪浅。

## 5.5系统源代码

代码见附带工程