

**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1601**

**学 号： U2016148989**

**姓 名： 潘翔**

**指导教师： 甘早斌**

**报告日期： 2019.4**

**计算机科学与技术学院**

**目 录**

[1 实验一 低频读写器实验 1](#_Toc318454614)

[1.1 实验目的 1](#_Toc974016314)

[1.2 实验内容与要求 1](#_Toc1303145877)

[1.3 实验过程与结果 2](#_Toc1823531468)

[1.4 核心源码说明 6](#_Toc1467681833)

[1.5 思考题 14](#_Toc1411896071)

[1.6 实验体会与总结 15](#_Toc1694338255)

[2 实验二 高频读写器实验(ISO14443A) 16](#_Toc1635021187)

[2.1 实验目的 16](#_Toc712386389)

[2.2 实验内容与要求 16](#_Toc191114524)

[2.3 实验过程与结果 17](#_Toc1531253575)

[2.4 核心源码说明 21](#_Toc326715619)

[2.5 思考题 28](#_Toc395864605)

[2.6 实验体会与总结 29](#_Toc1955582294)

[3 实验三 高频读写器实验(ISO15693) 31](#_Toc1197243029)

[3.1 实验目的 31](#_Toc27246611)

[3.2 实验内容与要求 31](#_Toc2053985034)

[3.3 实验过程与结果 32](#_Toc309463636)

[3.4 核心源码说明 44](#_Toc282157845)

[3.5 思考题 56](#_Toc87796018)

[3.6 实验体会与总结 56](#_Toc575791604)

[4 实验四 超高频读写器实验 58](#_Toc584039154)

[4.1 实验目的 58](#_Toc2055522958)

[4.2 实验内容与要求 58](#_Toc19260554)

[4.3 实验过程与结果 59](#_Toc1147355961)

[4.4 核心源码说明 67](#_Toc1816183587)

[4.5 思考题 71](#_Toc833824581)

[4.6 实验体会与总结 72](#_Toc1079374709)

[参考文献 73](#_Toc458391103)

# 实验一 低频读写器实验

## 实验目的

1. 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）。
2. 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。

## 实验内容与要求

### 实验内容

1. 完成低频读写器的标签读取试验；
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。

### 实验要求

1. 学会通过试验箱对低频标签的读取，通过反复循环读取十张低频电子标签，记录 在读取过程中遇到的问题或发现的现象，并分析遇到的这些问题或现象的原因；
2. 掌握低频读写器 API 函数的调用方法，编写程序打开串口，建立连接，读取标签信息；
3. 将标签移进移出低频模块天线，能够记录标签读取的次数。

## 实验过程与结果

### 环境配置

考虑在自身PC进行相关实验，故进行环境配置。使用x86交叉编译器进行编译。

1. 平台环境

OS: Manjaro 18.0.4 Illyria

Kernel: x86\_64 Linux 5.0.7-1-MANJARO

CPU: Intel Core i7-6700HQ @ 8x 3.5GHz

GPU: GeForce GTX 965M

RAM: 7865MiB

1. 软件环境

Qt Creator 3.5.1 (opensource)

Based on Qt 5.5.1 (GCC 4.9.1 20140922 (Red Hat 4.9.1-10), 32 bit)

### 实验过程

1. 发现问题

对系统进行测试，发现只能使用单卡进行相应操作，考虑纪录更新函数进行更改，同时希望添加时间记录。

1. 更改代码



图1.1 代码修改图

|  |
| --- |
| /\*\*   \* @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());//获取标签ID          QString **time** = CurrentDateTime();//获取时间  **int** index = model->findRecord(tagId);//查询此标签记录  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else** {              model->addRecord(tagId,**time**,tr("进"));          }      }  } |

1. 测试



图1.2 系统进出测试

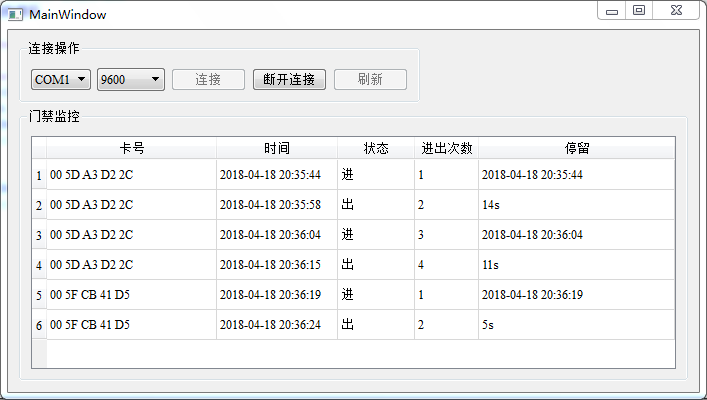


图1.3 系统停留时间测试

## 核心源码说明

### Qt事件相应机制

事件是由窗口系统或 qt 本身对各种事务的反应而产生的。 当用户按下、 释放一个键或鼠标按钮， 一个键盘或鼠标事件被产生； 当窗口第一次显示， 一个绘图事件产生， 从而告知最新的可见窗口需要重绘自身。

大多数事件是由于响应用户的动作而产生的， 但还有一些， 比如定时器等， 是由系统独立产生的。

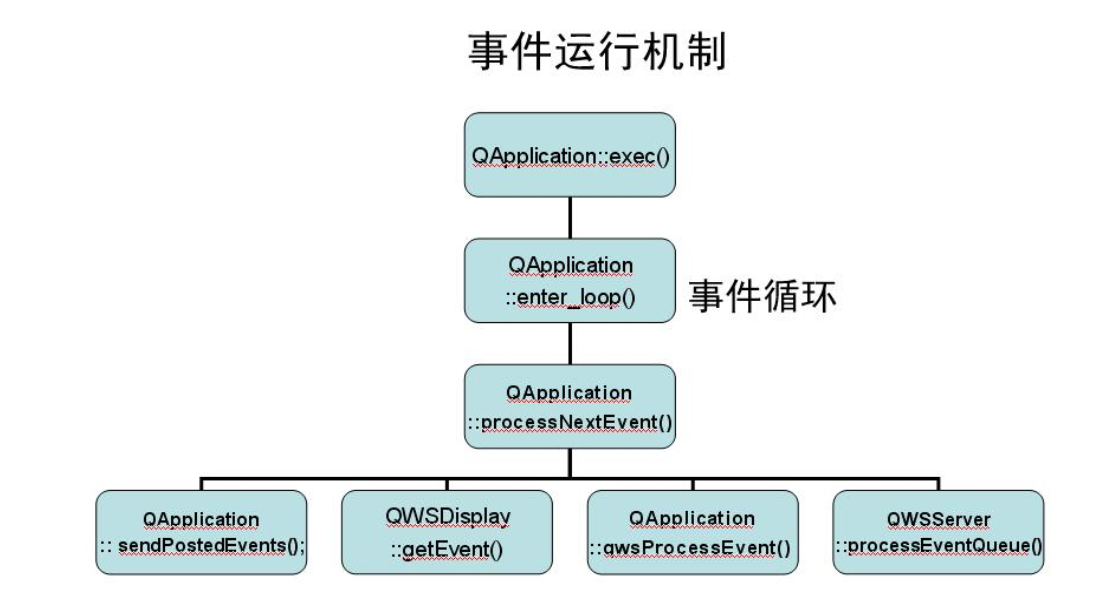


图1.4 Qt事件相应机制

### 数据库操作源码

|  |
| --- |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "recordtablemodel.h"  #include <QMessageBox>  #include <QDebug>  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   \*作者: jianghj@up-tech.com   \*日期: 2016-09-30   \*描述: 125K演示程序主要代码,此处模拟的人员通道,进出需要刷卡,   \*      125K在实际应用中主要也是这个功能,比如小区的门禁卡.   \*      注意:人为主动刷卡,2.4G是被动刷卡  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  MainWindow::MainWindow(QWidget \*parent) :      QMainWindow(parent),      ui(**new** Ui::MainWindow)  {      ui->setupUi(**this**);  **this**->fillPortsParameters(ui->baudRateBox);//波特率填充  **this**->serialPort = **new** QSerialPort(**this**);      db = **new** Database(**this**);//连接数据库      model = **new** RecordTableModel(**this**);      ui->tableView->setModel(model);      ui->tableView->resizeColumnsToContents();      ui->tableView->horizontalHeader()->setStretchLastSection(**true**);      intValidator = **new** QIntValidator(0, 4000000,**this**);      ui->btn\_connect->setEnabled(**true**);      ui->btn\_refresh->setEnabled(**true**);      ui->btn\_disconnect->setEnabled(**false**);  **this**->on\_btn\_refresh\_clicked();      m125dll = **new** M125Dll();      //关联相关槽函数      connect(ui->baudRateBox, SIGNAL(currentIndexChanged(**int**)),**this**, SLOT(checkCustomBaudRatePolicy(**int**)));      connect(serialPort, SIGNAL(error(QSerialPort::SerialPortError)), **this**, SLOT(handleError(QSerialPort::SerialPortError)));//收到串口错误信息      connect(serialPort, SIGNAL(readyRead()), **this**, SLOT(readData()));//收到串口信息  }    MainWindow::~MainWindow()  {      model->submitAll();  **delete** model;  **delete** db;  **delete** m125dll;  **delete** intValidator;  **delete** serialPort;  **delete** ui;  }  //Baudrate parameter init  **void** MainWindow::fillPortsParameters(QComboBox \*box)  {      box->clear();      box->addItem(QStringLiteral("9600"), QSerialPort::Baud9600);      box->addItem(QStringLiteral("19200"), QSerialPort::Baud19200);      box->addItem(QStringLiteral("38400"), QSerialPort::Baud38400);      box->addItem(QStringLiteral("57600"), QSerialPort::Baud57600);      box->addItem(QStringLiteral("115200"), QSerialPort::Baud115200);      box->addItem(tr("Custom"));  }  /\*\*   \* @brief MainWindow::on\_btn\_connect\_clicked   \* 连接串口   \*/  **void** MainWindow::on\_btn\_connect\_clicked()  {      QString name = ui->serialNameBox->currentText();      QString baud = ui->baudRateBox->currentText().trimmed();  **if**(baud.isEmpty())      {          QMessageBox::critical(**this**, tr("Error"), "波特率输入错误！");  **return** ;      }      serialPort->setPortName(name);      serialPort->setBaudRate(baud.toInt(),QSerialPort::AllDirections);  **if** (serialPort->open(QIODevice::ReadWrite)) {          ui->btn\_connect->setEnabled(**false**);          ui->btn\_disconnect->setEnabled(**true**);          ui->btn\_refresh->setEnabled(**false**);      } **else** {          ui->btn\_connect->setEnabled(**true**);          ui->btn\_refresh->setEnabled(**true**);          ui->btn\_disconnect->setEnabled(**false**);          QMessageBox::warning(**this**,tr("提示"),tr("初始化%1失败！请检查串口是否已经被占用？").arg(name),QMessageBox::Yes);      }  }  /\*\*   \* @brief MainWindow::on\_btn\_disconnect\_clicked   \* 断开连接   \*/  **void** MainWindow::on\_btn\_disconnect\_clicked()  {  **if**(!serialPort->isOpen())  **return** ;      serialPort->close();      ui->btn\_connect->setEnabled(**true**);      ui->btn\_refresh->setEnabled(**true**);      ui->btn\_disconnect->setEnabled(**false**);  }  /\*\*   \* @brief MainWindow::on\_btn\_refresh\_clicked   \* 刷新按钮点击事件   \*/  **void** MainWindow::on\_btn\_refresh\_clicked()  {      QStringList list = getSerialName();      ui->serialNameBox->clear();      ui->serialNameBox->addItems(list);  }  /\*\*   \* @brief MainWindow::checkCustomBaudRatePolicy   \* @param idx combox被选中的索引值   \* 设置自定义波特率   \*/  **void** MainWindow::checkCustomBaudRatePolicy(**int** idx)  {      QComboBox \*box = **dynamic\_cast**<QComboBox\*>(QObject::sender());  **bool** isCustomBaudRate = !box->itemData(idx).isValid();      box->setEditable(isCustomBaudRate);  **if** (isCustomBaudRate) {          box->clearEditText();          box->setValidator(intValidator);      }  }    /\*\*   \* @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());//获取标签ID          QString **time** = CurrentDateTime();//获取时间  **int** index = model->findRecord(tagId);//查询此标签记录  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else** {              model->addRecord(tagId,**time**,tr("进"));          }      }  }  /\*\*   \* @brief MainWindow::handleError   \* @param error SerialPortError枚举类,详细请看SerialPortError的定义   \* 处理错误信息   \*/  **void** MainWindow::handleError(QSerialPort::SerialPortError error)  {  **if** (error == QSerialPort::ResourceError) {          QMessageBox::critical(**this**, tr("Critical Error"), serialPort->errorString());  **this**->on\_btn\_disconnect\_clicked();      }  } |

### 125K接口

进行验证读取的数据是否符合 125K 卡号格式， 在 API 的头文件中有介绍分析 125K 帧接口的方法， 头文件中代码如下：

//125K模块上报卡号的起始字节

#define LF125K\_SOF    0x00

//成功定义为0

#define SUCCESS      0

//失败定义为-1

#define FAILURE     -1

**typedef** unsigned **char**   uint8;

//125K模块上报卡号的结构

**typedef** **struct**

{

    uint8 sof;

    uint8 data[3];

    uint8 fcs;    // xor verify

}LF125K\_RspFrame\_t;

**class** M125DLLSHARED\_EXPORT M125Dll

{

**public**:

    M125Dll();

    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

    \*函数名        ：LF125K\_FrameAnalysis

    \*描述     ：解析一帧数据是否有误

    \*输入参数   ：

    \*   @frame  const uint8 \* 125K的数据帧

    \*输出参数   ：无

    \*返回值        ：正确返回0，失败返回-1

    \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**int** LF125K\_FrameAnalysis(**const** uint8 \*frame);

};

## 思考题

1. 通过试验箱，反复循环读取十张低频电子标签。在读取过程中可能会遇到哪些问 题或发生哪些现象，并分析遇到的这些问题或现象的原因；

发生只有一张卡进行记录，为单卡读取

1. 在利用低频读写器模拟门禁系统中，如何获取读写器发送过来的卡号？请写出相 应的函数体（含注释），并说明函数的调用方法。

|  |
| --- |
| /\*\*   \* @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)                  // call function in m125k DLL      {          QString tagId = CharStringtoHexString(tr(" "),data.data(),data.length());   // get tag ID          QString **time** = CurrentDateTime();                                           // get cuttent time  **int** index = model->findRecord(tagId);                                       // findRecord by tagID  **if**(index >= 0 )          {              QString text = model->record(index).value(2).toString();                // set time  **if**(text == tr("进"))                  model->updateRecord(index,tagId,**time**,tr("出"));  **else**                  model->updateRecord(index,tagId,**time**,tr("进"));          }  **else**          {              model->addRecord(tagId,**time**,tr("进"));          }      }  } |

## 实验体会与总结

实验过程中，对试验箱进行了相应的熟悉，在系统的相应配置过程中出现了一些问题，试验箱对应的整体实验分为两部分，可以通过本地PC进行宿主机进行reader的相应操控，也可以使用试验箱板载的arm进行相应的操控。

如果使用ARM进行操控，需要使用交叉编译技术，实验环境为32bit的Qt Version。 Qt本身为跨平台，但是出于DLL限制，无法使用64bit Qt进行编译，但是Linux上32 Bit Qt已经停止维护，故从官网上的开源仓库获取较老的版本。

整个代码较为清晰，各个工具类分的比较详细，但是操作函数完全位于mainwindow类，较为混乱且拥挤，对于DLL封装切不提供多平台版本和源码，对于跨平台不友好。

整个试验箱更类似于一个教学平台，其中不同reader的相应切换没有提供相应的接口，且只能依靠物理方式切换，有些不方便。

实验过程中，查询了相关文档，对于系统串口通信方式和系统调用过程有了更深的理解。

# 实验二 高频读写器实验(ISO14443A)

## 实验目的

1. 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。
2. 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。
3. 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。

## 实验内容与要求

### 实验内容

1. 完成 ISO14443A 协议下标签寻卡、唤醒、休眠实验；
2. 完成 ISO14443A 协议下标签内存读写实验；
3. 完成 ISO14443A 协议下标签一卡通实验；
4. 熟悉和了解高频 HF1356M 14443A 开发实例，掌握高频读写器（14443A）API 函数，并通过编程实现对 S50 卡的指定内存进行数据读写、充值、减值。

### 实验要求

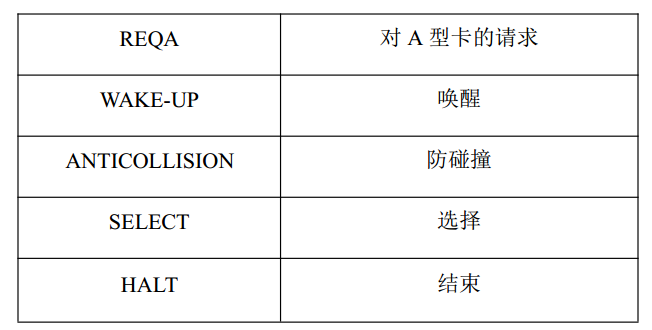
1. 通过试验箱，学会对 ISO14443A 协议下标签进行寻卡、唤醒、休眠操作， 掌握寻卡、唤醒、休眠的命令代码、命令功能和命令数据包构成，并分析唤醒、 休眠命令对标签寻卡所产生的影响。
2. 通过试验箱，学会对 ISO14443A 协议下标签指定内存的数据读写。掌握块 3 存取控制位对本扇区其他块的读写操作控制逻辑。掌握密码 A 和密码 B 的 加载方法以及对不同块的读写控制逻辑。
3. 学会通过试验箱对 ISO14443A 协议下标签进行加值减值操作；
4. 掌握高频读写器API函数的调用方法，并能够通过编程实现对ISO14443A 协议下标签数据的读写操作以及加值减值操作。

## 实验过程与结果

### ISO14443A协议理解

1. PICC状态集
   1. 调电状态： 由于没有足够的载波能量， PICC 没有工作， 也不能发送反射波。
   2. 闲置状态： 在这个状态时， PICC 已经上电， 能够解调信号， 并能够识别有效的 REQA 和 WAKE-UP命令。
   3. 准备状态： 本状态下， 实现位帧的防碰撞算法或其它可行的防碰撞算法。
   4. 激活状态： PCD 通过防碰撞已经选出了单一的卡。
2. 命令集

表2.1 PCD 用于管理与 PICC 之间通信的命令



1. 协议激活

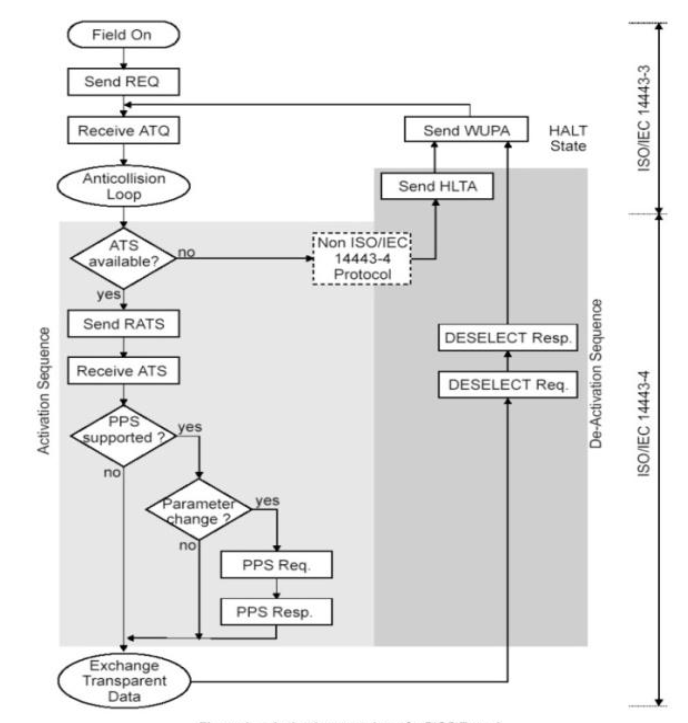


图2.1 A卡协议激活图

### ISO14443A寻卡、唤醒、休眠实验



图2.2 寻卡、唤醒、休眠实验图

### ISO14443A标签内存读写实验；

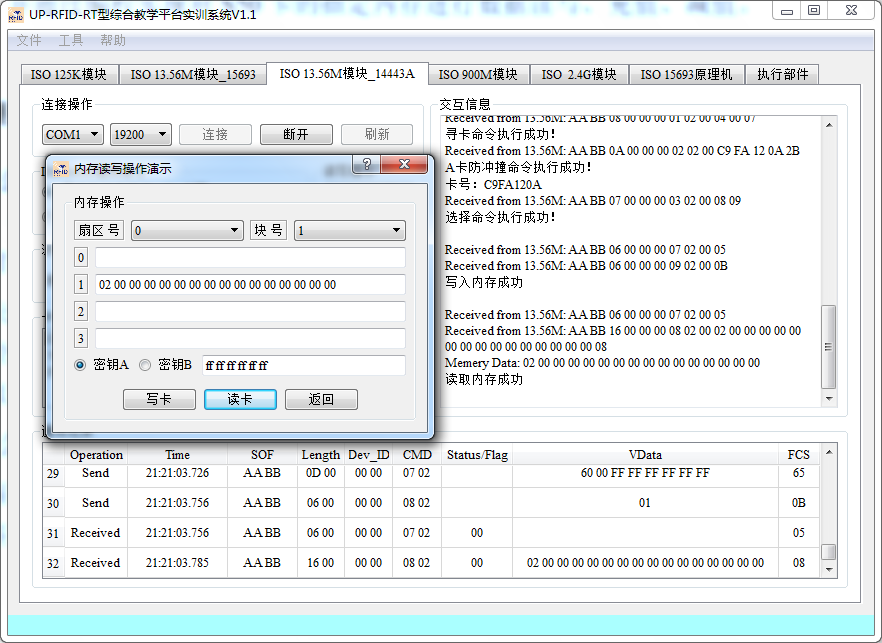


图2.3 读卡实验图

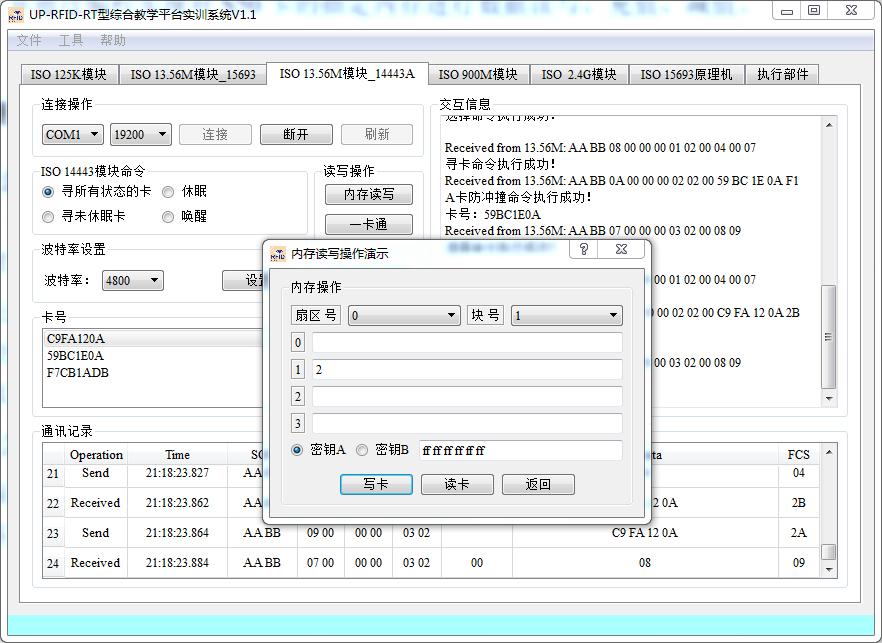


图2.4 写卡实验图

3、完成 ISO14443A 协议下标签一卡通实验；



图2.5 一卡通充值图

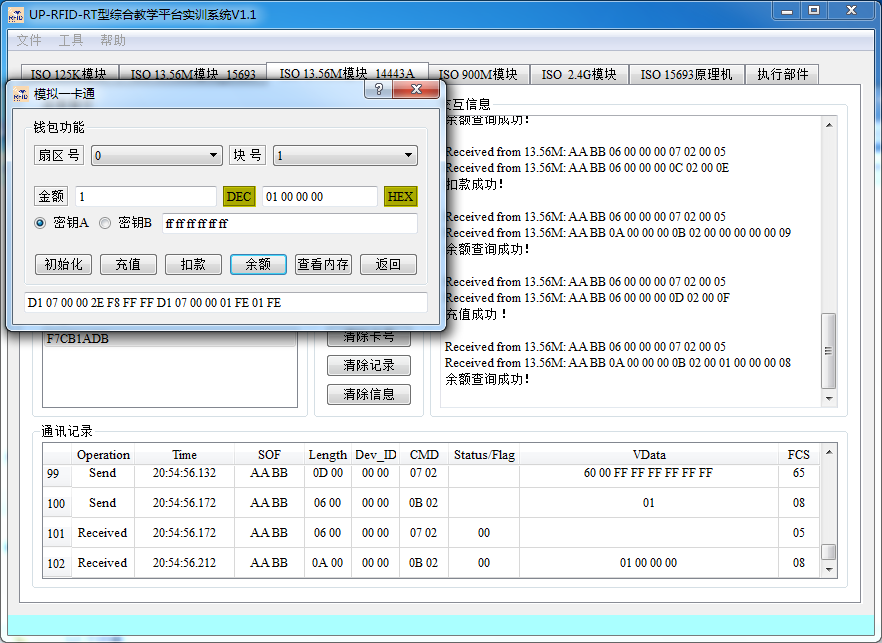


图2.6 一卡通查询余额图

## 核心源码说明

### 系统架构

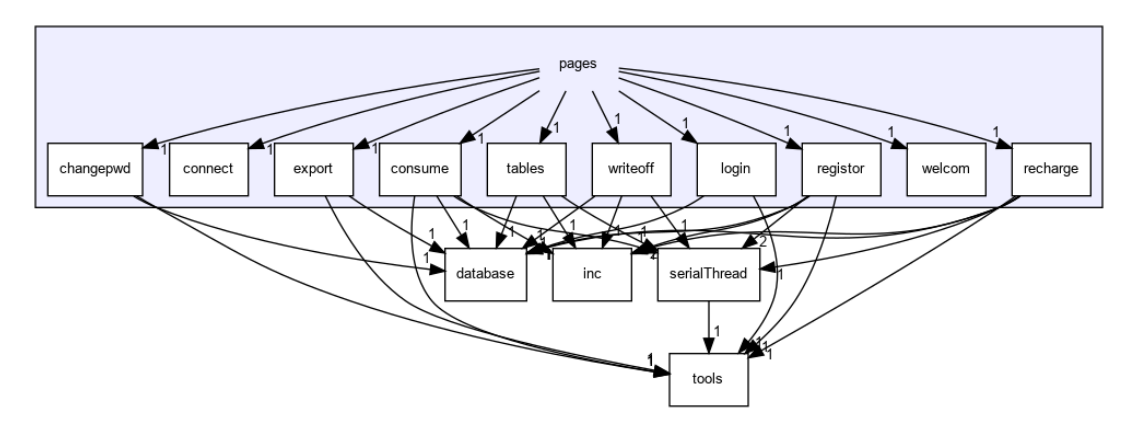


图2.7 工具类调用关系图

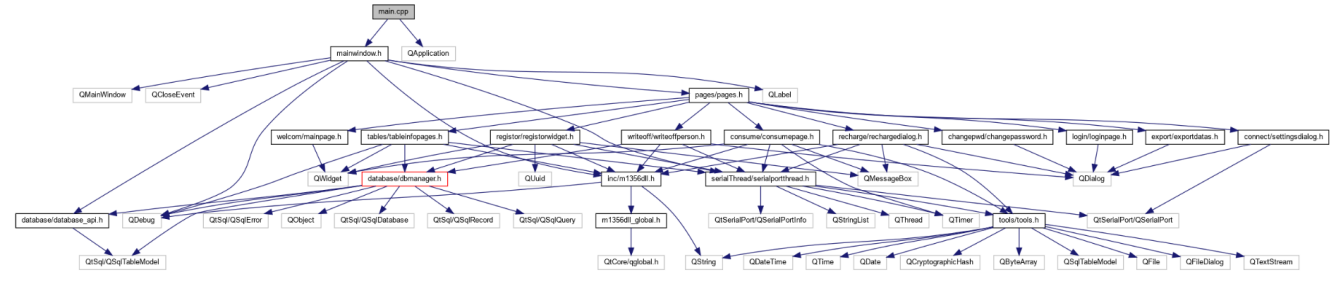


图2.8 系统架构图

### DBManager类型声明

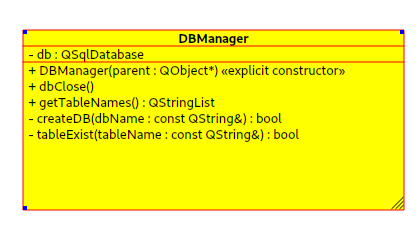


图2.9 DBManager接口声明图

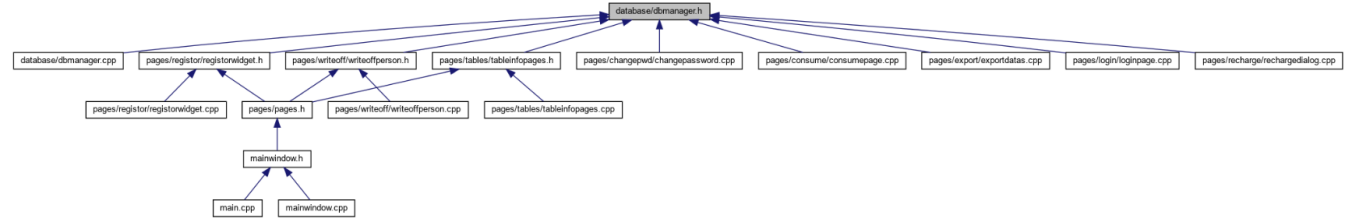


图2.10 DBManager接口调用图

**class** DBManager : **public** QObject

{

    Q\_OBJECT

**public**:

**explicit** DBManager(QObject \*parent = 0);

**void** dbClose(); //关闭数据库

    QStringList getTableNames();//获取所有表的表名

**private**:

    QSqlDatabase db;//sqlite database

    //创建数据库

**bool** createDB(**const** QString &dbName);

    //检测该表是否存在

**bool** tableExist(**const** QString &tableName);

signals:

**public** slots:

};

### DBManager方法实现

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

{

**if**(**this**->createDB(DATABASE\_NAME))               //create DB by DB name

    {

**if**(!**this**->tableExist(TABLE\_NAME\_PERSON))    //op PersonTableModel

        {

            PersonTableModel p;

            p.createTable();

        }

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

        {

            RecordTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_REGISTER))  //op RegisterTableModel

        {

            RegisterTableModel r;

            r.createTable();

        }

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

        {

            WriteOffTableModel w;

            w.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_ADMIN))     //op AdminTableModel

        {

            AdminTableModel a;

            a.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECHARGE))  //op RechargeTableModel

        {

            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) // createDB by name

{

    db = QSqlDatabase::addDatabase("QSQLITE");  //sqlite DB

    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)    //judge DB exist

{

**int** count = 0 ;

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

    QSqlQuery query;        // judge by select ans

    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; // get table by name

    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();  // wrap for get table

}

### Model实现方式

对于不同的对象采用不同的Model进行设计实现，其实现方式如下

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

{

tableName = TABLE\_NAME\_REGISTER;

header<<"卡号"<<"用户编号"<<"时间"<<"信息备注";

}

### 串口通信

程序作为数据库和串口相应的交互，需要进行串口通信单独类的剥离

SerialPortThread::SerialPortThread(QObject \*parent) : QThread(parent)

{

    serialIsOpen = **false**;

    Stop = **false**;

    timer = **new** QTimer(**this**);

    connect(timer,SIGNAL(timeout()),**this**,SLOT(onError()));

    connect(**this**,SIGNAL(hasResponse()),**this**,SLOT(stopTimer()));

}

利用QThred创建线程，作为独立的串口通信线程，利用定时器机制进行轮训

## 思考题

1. S50 卡共有 16 个扇区，请问第 4 扇区的绝对块地址号是多少？请详细说明计算的方法和依据。

S50把1K字节的容量分为16个扇区(Sector0-Sector15)，每个扇区包括4个数据块(Block0-Block3，我们也将16个扇区的64个块按绝对地址编号为0~63)，每个数据块包含16个字节(Byte0-Byte15)，64\*16=1024。所以，第4扇区绝对地址号是：3\*4=12，为12。

1. S50 卡第 1 扇区第 0 块是否可读写？为什么？

S50从0开始起标，故第1扇区可以读写，因为只有第0扇区，第0块号为卡序列号，不可写(只读)。

1. “S50 卡共有 16 个扇区，每个扇区由 4 块组成，第 4 块为控制块，其余 三块为数据块，都可用于存储数据”。这句话正确吗？如果不正确，请改正。

不正确，控制块也可以用于存储数据。应该改正为S50卡除了0号扇区以外的每个扇区其他三块都为数据块，都可用于存储数据，0号扇区的第一块是用来存储卡的序列号。

1. S50 卡的数据块用于存储数据时，可以有哪几种用途？

所有扇区都由3个块组成，每个块由16字节用于存储数据(扇区0只有两个数据块，一个只读的厂商数据块)。数据块可以设置为：读写块，例如用于非接触门禁管理，有效命令： read, write;数值块，例如用于电子钱包，允许执行电子钱包功能(有效的命令是:读、写增量、减量、恢复、转移)。数值块有一个固定的数据格式允许错误检测和校正和备份管理。

1. 如何将一张空白的 S50 卡初始化成电子钱包？

可将卡放入读卡器，点击寻卡功能，找到该卡，执行读卡、写卡、初始化卡等功能，并进行相应的功能设置，就能将卡初始化为电子钱包，并将金额初始化为0。

## 实验体会与总结

实验过程中，对于ISO14443A协议有了更深的理解，由于实验提供了对应的一卡通管理系统，故可以直接将卡进行一卡通化，实际上，本质上仍然为定义了自己的数据存储方式，进行了相应的数据存储形式定义，与普通的信息管理系统无异，仅仅是存储形式在内存，硬盘还是RFID卡的区别。

此实验告诉我们，仅仅需要从数据结构的角度理解RFID卡，对于主键(ID)等进行保护控制，其余数据结构可以自己定义，同时满足相应的读写接口。

存取权限控制上，有硬件控制方式(reader/tag)设定权限，软件设置(界面仅仅提供允许用户更改的相应权限)。实际上，硬件仅仅能对TagID等进行控制，对于需要灵活更改的部分，需要考虑加密进行软件控制。

# 实验三 高频读写器实验(ISO15693)

## 实验目的

1. 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。
2. 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。

## 实验内容与要求

### 实验内容

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验；
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。

### 实验要求

1. 学会通过试验箱对 ISO15693 协议下标签指定内存地址的数据进行读写操作；
2. 理解和掌握应用族标识符（AFI）、数据存储格式标识符（DSFID）以及锁的基本概念和含义；
3. 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 ISO15693 协议下标签数据的读写控制。

## 实验过程与结果

### 读取状态设置



图3.1 修改波特率为38400图

图3.2 修改波连接特率为38400图



图3.3 断电后38400连接失败图



图3.4 恢复19200连接成功图

### ISO15693单标签/多标签手工寻卡和自动寻卡



图3.5 单标签寻卡图



图3.6 多标签寻卡图

### ISO15693指定地址的数据读写实验

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；

### ISO15693指定地址范围的内存数据读取实验

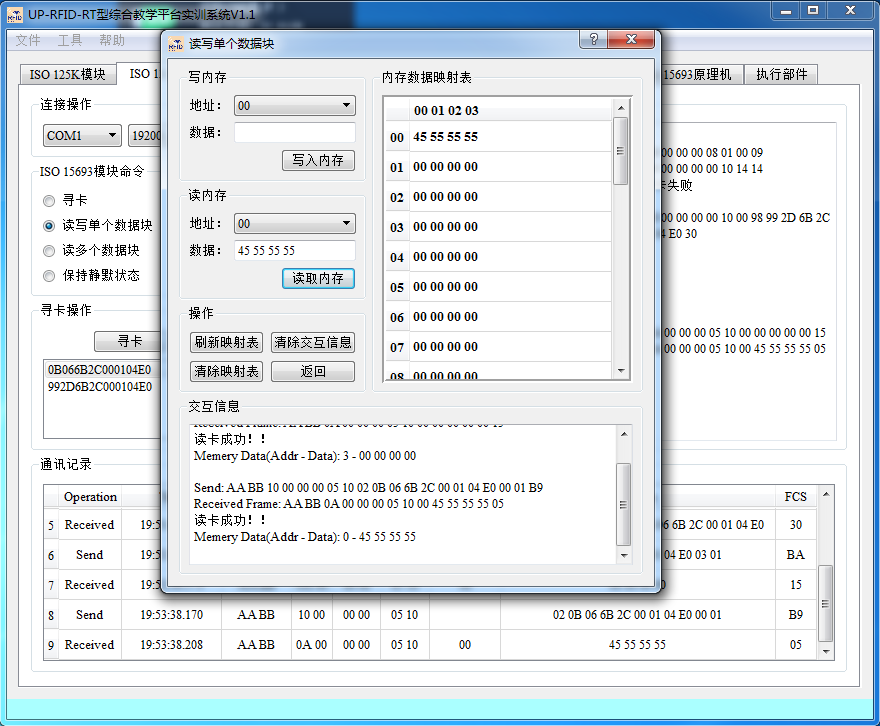


图3.7 指定地址范围读卡(00)实验图

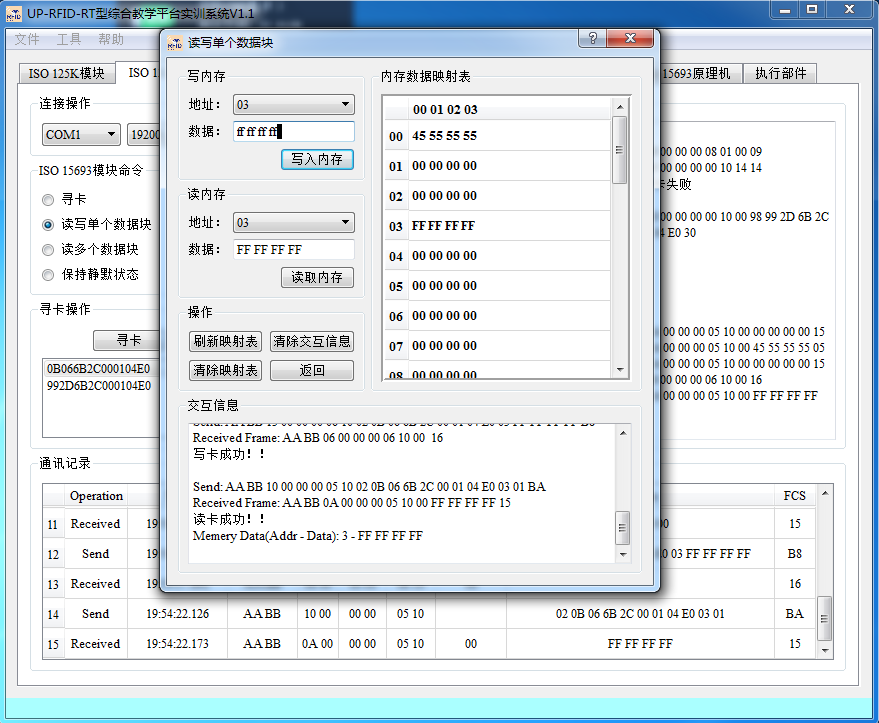


图3.8 读写(03)实验图

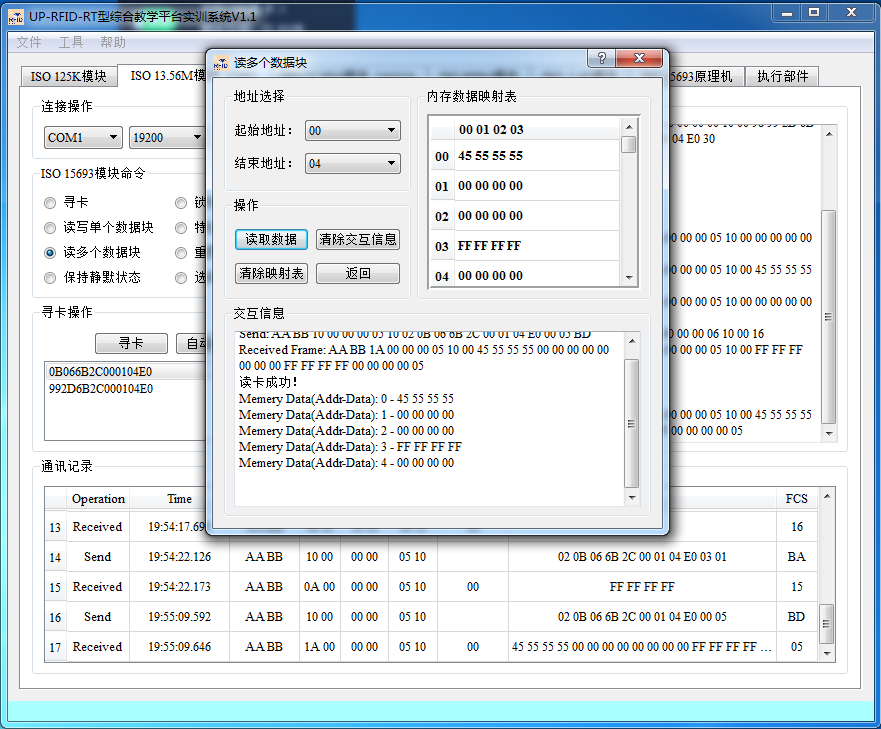


图3.9 读多个数据块实验图

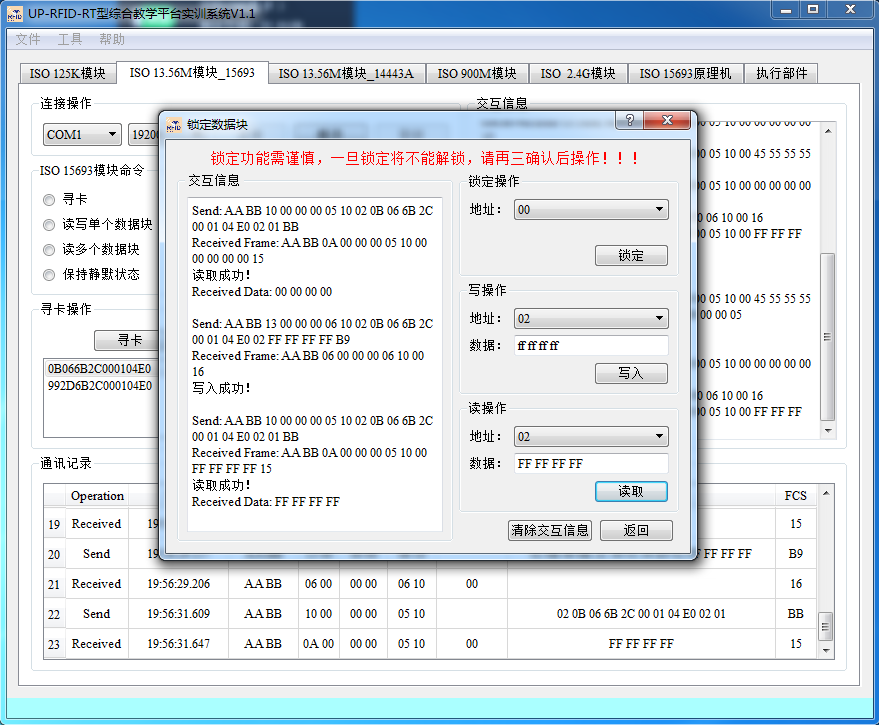


图3.10 读写(02)实验图

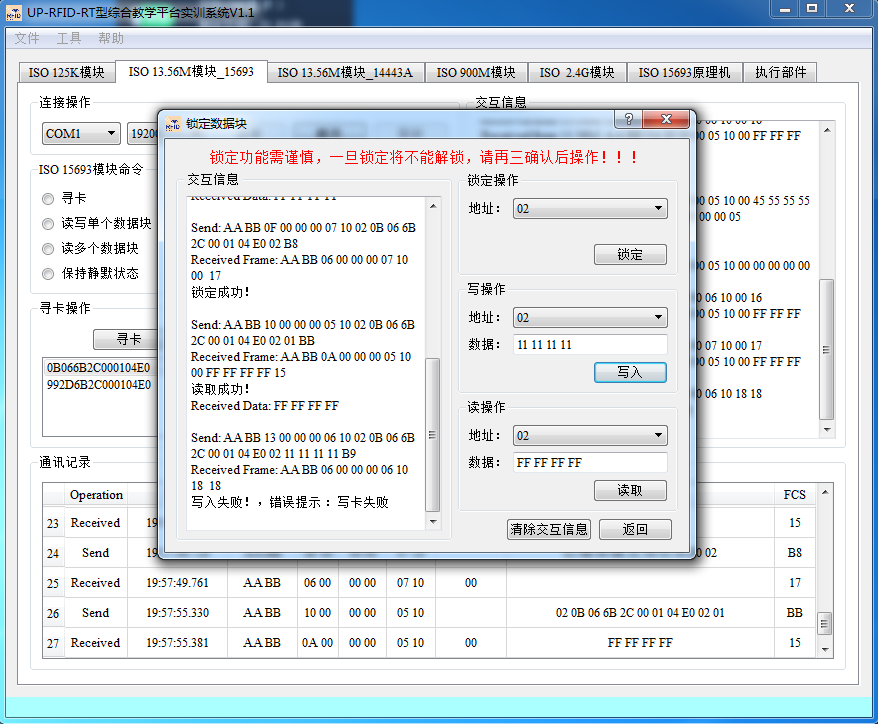


图3.11 锁定02后可读不可写实验图

### ISO15693协议的命令

1. 完成标签静默状态设置



图3.12 静默实验

1. 重置到准备状态



图3.13 恢复实验

### ISO15693 协议下标签 DSFID、AFI 的读写和块安全位的读取实验

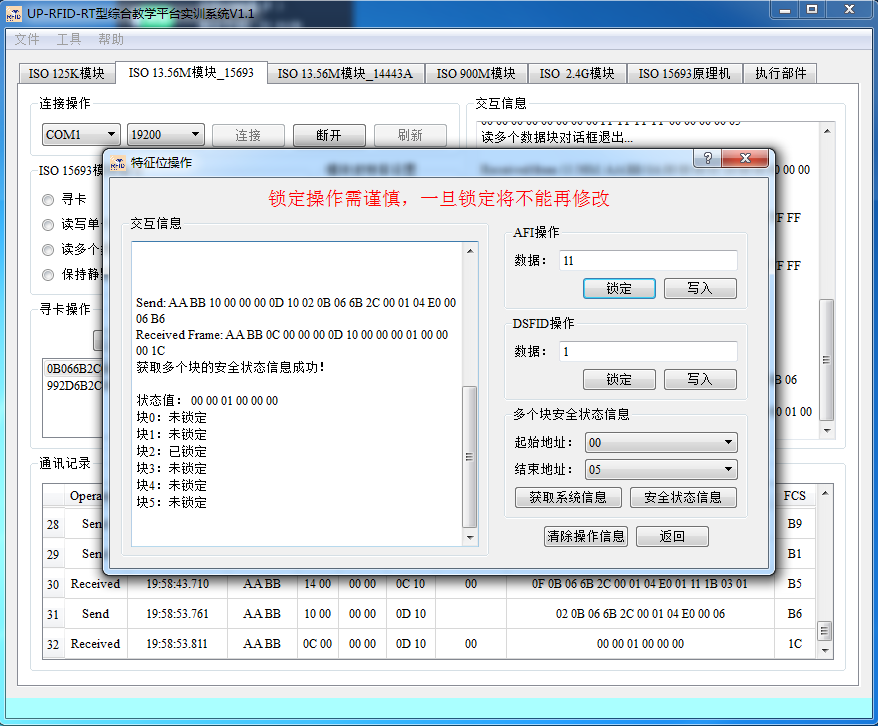


图3.14 安全状态图

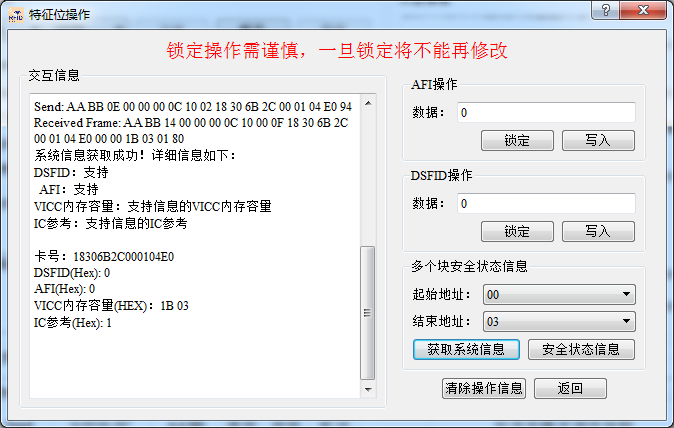


图3.15 DSFID、AFI 的读写和块安全位的读取实验

### ISO15693 协议下标签的读写功能

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



图3.16 开发实例测试图

## 核心源码说明

### 系统架构图

系统在BooksManage中进行系统相应槽函数的刷新

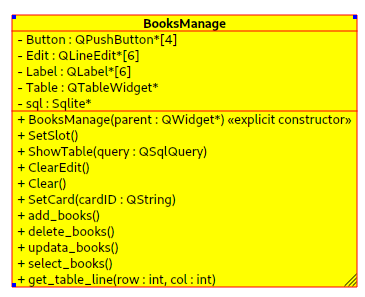


图3.17 BooksManage接口图

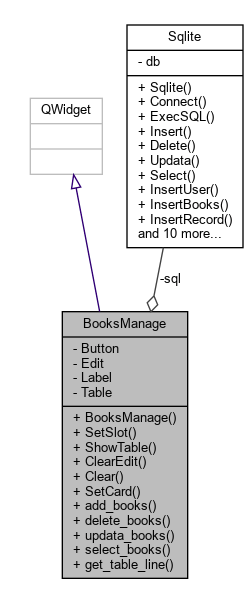


图3.18 BooksManage与DB交互图

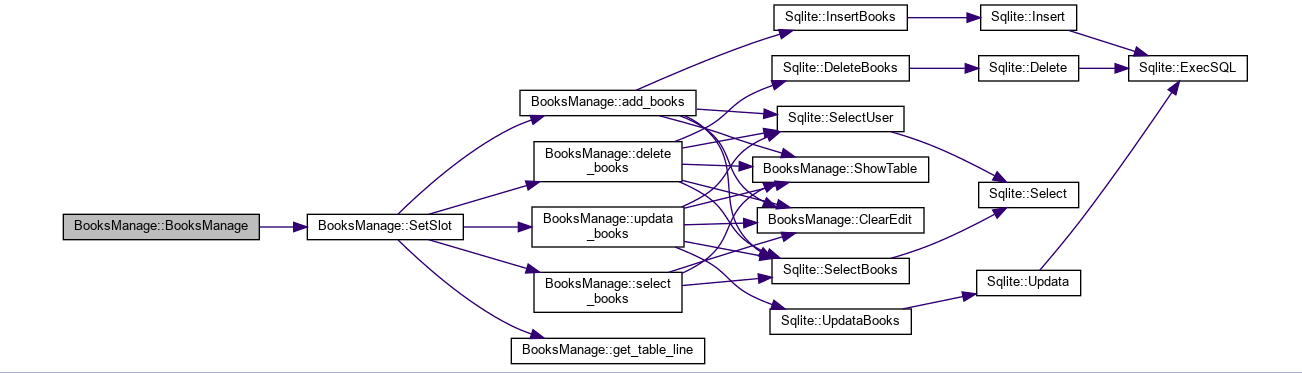


图3.19 BooksManage调用图

### BooksManage接口定义

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

{

**if**(**this**->createDB(DATABASE\_NAME))               //create DB by DB name

    {

**if**(!**this**->tableExist(TABLE\_NAME\_PERSON))    //op PersonTableModel

        {

            PersonTableModel p;

            p.createTable();

        }

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

        {

            RecordTableModel r;

            r.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_REGISTER))  //op RegisterTableModel

        {

            RegisterTableModel r;

            r.createTable();

        }

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

        {

            WriteOffTableModel w;

            w.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_ADMIN))     //op AdminTableModel

        {

            AdminTableModel a;

            a.createTable();

        }

**if**(!**this**->tableExist(TABLE\_NAME\_RECHARGE))  //op RechargeTableModel

        {

            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) // createDB by name

{

    db = QSqlDatabase::addDatabase("QSQLITE");  //sqlite DB

    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)    //judge DB exist

{

**int** count = 0 ;

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

    QSqlQuery query;        // judge by select ans

    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; // get table by name

    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();  // wrap for get table

}

### 字符串转换函数

Qt采用UTF8编码，需要将QString转化成为HEXString

///普通字符串转为16进制字符串

QString CharStringtoHexString(QString space, **const** **char** \* src, **int** start, **int** end)

{

    QString hex = "";

**if**(space == NULL)

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.toUpper();

    }

**else**

    {

**for**(**int** i = start ; i < end ; i ++)

        {

            hex += space + QString("%1").arg(src[i]&0xFF,2,16,QLatin1Char('0'));

        }

**return** hex.right(hex.length() - space.length()).toUpper();

    }

}

### 发送请求帧

需要对帧长进行计算，并对相应的控制位进行设置，作为cmd帧请求

//发送请求帧

uint8\* M1356Dll::RC632\_SendCmdReq(uint16 cmd, **const** uint8 \*data, uint16 len)

{

    uint8 \*p;

    uint16 frameLen, uartdatalen;

    frameLen = len+9;   // 9 = sop(2)+len(2)+nc(2)+cmd(2)+fcs(1)

    uartdatalen = frameLen;

**for**(uint16 i=0; i<len; i++)

**if**(data[i] == 0xAA)  uartdatalen++;

    p = **new** uint8[uartdatalen+2];  // p[0] is used return frameLen

    p[0] = LO\_UINT16(uartdatalen);

    p[1] = HI\_UINT16(uartdatalen);

    p[2] = LO\_UINT16(RC632\_FRAME\_SOP);

    p[3] = HI\_UINT16(RC632\_FRAME\_SOP);

    p[4] = LO\_UINT16(frameLen-4);  // 4 = sop(2)+len(2)

    p[5] = HI\_UINT16(frameLen-4);  // 4 = sop(2)+len(2)

    p[6] = 0;  // NC

    p[7] = 0;  // NC

    p[8] = LO\_UINT16(cmd);

    p[9] = HI\_UINT16(cmd);

    //memcpy(p+10, data, len);

**int** k=0;

**for**(**int** j=0; j<len; j++, k++){

        p[10+k] = data[j];

**if**(data[j]==0xAA)  p[10+ ++k] = 0x00;

    }

    p[10+k] = RC632\_UartCalcFCS(p+5, uartdatalen-4);

**return** p;

}

### 解析返回帧

其中，对于不同返回帧的校验进行了处理，qDebug()用于返回值错误的输出，但此处并没有进行相应的错误处理

//解析某一帧数据

uint8 M1356Dll::RC632\_AnalysisFrame(uint8 \*frame, uint16 cmd)

{

    uint16 sof = BUILD\_UINT16(frame[RC632\_RPC\_SOP], frame[RC632\_RPC\_SOP+1]);

    uint16 len = BUILD\_UINT16(frame[RC632\_RPC\_LEN], frame[RC632\_RPC\_LEN+1]);

    uint16 rxCmd = BUILD\_UINT16(frame[RC632\_RPC\_CMD], frame[RC632\_RPC\_CMD+1]);

**if**(sof == RC632\_FRAME\_SOP)

    {

**int** tempLen = len,i=0,j=0;

**while**(tempLen > i){  //

            frame[RC632\_RPC\_NC+i] = frame[RC632\_RPC\_NC+j];

**if**(frame[RC632\_RPC\_NC+i] == 0xAA)  j++;

            i++; j++;

        }

**if**(rxCmd == cmd)

        {

            uint8 fcs = RC632\_UartCalcFCS(&frame[RC632\_RPC\_LEN+1], len);

**if**(fcs == frame[RC632\_RPC\_LEN+len+1])

**return** frame[RC632\_RPC\_DAT];

**else**{

                qDebug()<<"[ERR-RC632 Frame FCS ERR]";

                qDebug("%02X!=%02X", fcs, frame[RC632\_RPC\_LEN+len+1]);

            }

        }

**else**

        {

            qDebug()<<"[ERR-RC632 Frame CMD ERR]";

        }

    }

**else**

    {

        qDebug()<<"[ERR-RC632] Frame SOP ERR";

    }

**return** RC632\_FAULT255;  // frame fcs error

}

### 错误解析

/\*\*

 \* @brief GET\_ERROR\_STRING

 \* @param error 错误码

 \* @return  错误信息

 \* 获取错误码

 \*/

QString M1356Dll::GET\_ERROR\_STRING(QString error)

{

    QString result="未知错误";

**bool** ok = **true**;

**if**(error.length() == 2)

    {

**int** id = error.toInt(&ok,16);

**switch** (id) {

**case** RC632\_FAULT10:

            result = "通用错误";

**break**;

**case** RC632\_FAULT11:

            result = "不支持该命令";

**break**;

**case** RC632\_FAULT12:

            result = "无卡";

**break**;

**case** RC632\_FAULT13:

            result = "射频基站损坏";

**break**;

**case** RC632\_FAULT14:

            result = "射频基站损坏";

**break**;

**case** RC632\_FAULT20:

            result = "寻卡失败";

**break**;

**case** RC632\_FAULT21:

            result = "卡复位失败";

**break**;

**case** RC632\_FAULT22:

            result = "密码验证失败";

**break**;

**case** RC632\_FAULT23:

            result = "读卡失败";

**break**;

**case** RC632\_FAULT24:

            result = "写卡失败";

**break**;

**case** RC632\_FAULT100:

            result = "未响应";

**break**;

**default**:

            result = "未知错误";

**break**;

        }

    }

**return** result;

}

## 思考题

1. 什么是 AFI？AFI 如何编码？在通过编程对 AFI 进行读写、锁定时，其对应 ISO15693 的协议命令代码、上位机对读写器的命令代码和数据包分别是怎 么样的？

AFI是应用标示的简称，在ISO15693的电子标签中就有AFI写和锁。用户可以自己写一个关键字作为标签的类别。AFI被编码在一个字节里，由两个半字节组成，AFI的高半位字节，用于编码一个特定的或所有应用族，AFI的低半位字节用于编码一个特定的或所有应用子族。子族不同于0的，有其自己的所有权。

1. 什么是DSFID？在通过编程对DSFID进行读写、锁定时，其对应ISO15693 的协议命令代码、上位机对读写器的命令代码和数据包分别是怎么样的？

特殊功能 DSFID（数据存储格式标识符）可用来表示数据在存储器中的存储结构，具 体内容请自己查阅相关文档。数据存储格式标识符（DSFID）数据存储格式标识符指出了数据在内存中是怎样构成的。DSFID 被相应的命令编程和锁定。DSFID 被编码在一个字节里。DSFID 允许即时知 道数据的逻辑组织。假如标签不支持 DSFID 的编程，标签将以值“0”作为应答。

1. ISO15693 协议的电子标签 ID 有何特点？

ISO/IEC 15693 协议标准的高频 RFID 无源 IC 卡，专为供应链与运筹管理应用所设计，具有高度防冲突与长距离运作等优点，适合于高速、长距离应用。

工作频率范围为13.56MHz±7KHz，工作磁场中工作场最小值0.15A/m,最大值5A/m，支持多种调制方式，支持防冲突和传输协议。

## 实验体会与总结

通过这次实验，通过试验箱对 ISO15693 协议下标签指定内存地址的数据进行读写操作，加深了对应用族标识符（AFI）、数据存储格式标识符（DSFID）以及锁理解，了解高频读写器 API 函数的调用方法。

实验代码部分，通过DLL进行GPIO串口通信，采用DB工具类进行数据库操作，同时将种不同身份的类别进行隔离进行权限管理。

实验代码对于错误处理上，并没有进行处理，而是直接将错误输出，系统的鲁棒性不够，将错误处理交给了用户，可以采取重新发送指令再次读取，或者提供相应的操作方法。

代码大量使用qDebug()函数，在实际过程中，系统最终无法呈现某些错误信息，可以考虑将qDebug()输出为log进行记录。

# 实验四 超高频读写器实验

## 实验目的

1. 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。
2. 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。

## 实验内容与要求

### 实验内容

1. 超高频读写器的基本认知，完成超高频读写器频率和功率读取和设置实验；
2. 完成 Gen2 协议下单标签和多标签识别实验；
3. 执行 Gen2 协议下单命令操作实验，并分别对 EPC 标签各个存储区进行 读写擦除操作试验；
4. 熟悉和了解超高频 UHF-900M 开发实例，掌握超高频读写器 API 函数， 并通过编程实现 Gen2 协议下标签的读写功能。

### 实验要求

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。

## 实验过程与结果

### 读写状态设置



图4.1 读取版本号和功率图



图4.2 设置功率图

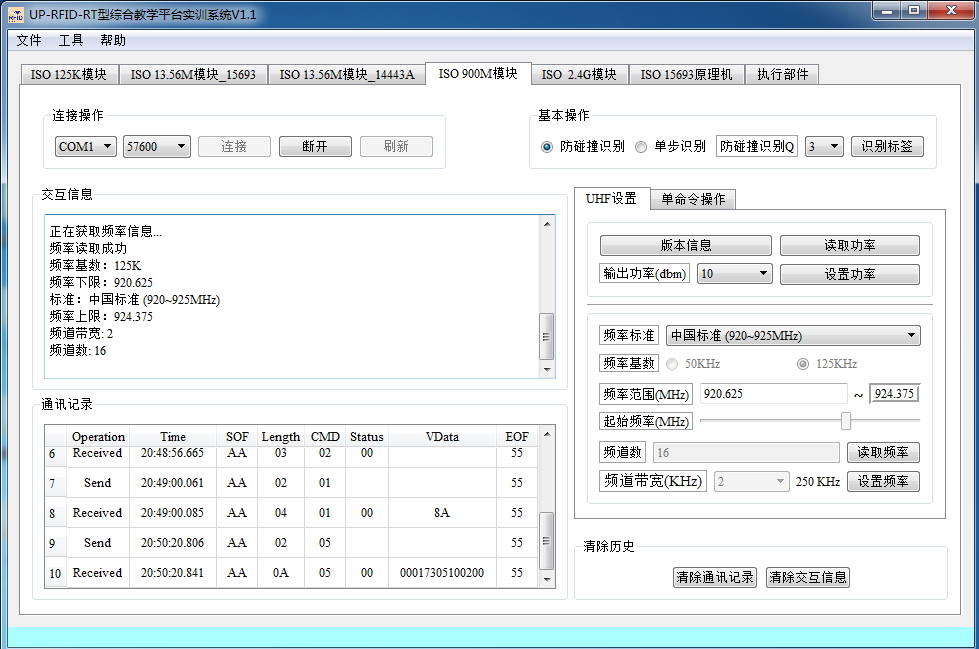


图4.3 读取频率信息图

### 单标签识别实验



图4.4 单标签单步识别识别图

### 多标签识别实验

对于卡重叠进行测试，当两张卡发生重叠的时候，只能读取到下面的卡。



图4.5 多标签单步识别图



图4.6 多标签防碰撞识别图

### 读写擦除操作实验



图4.7 读取内存数据图



图4.8 写入成功图

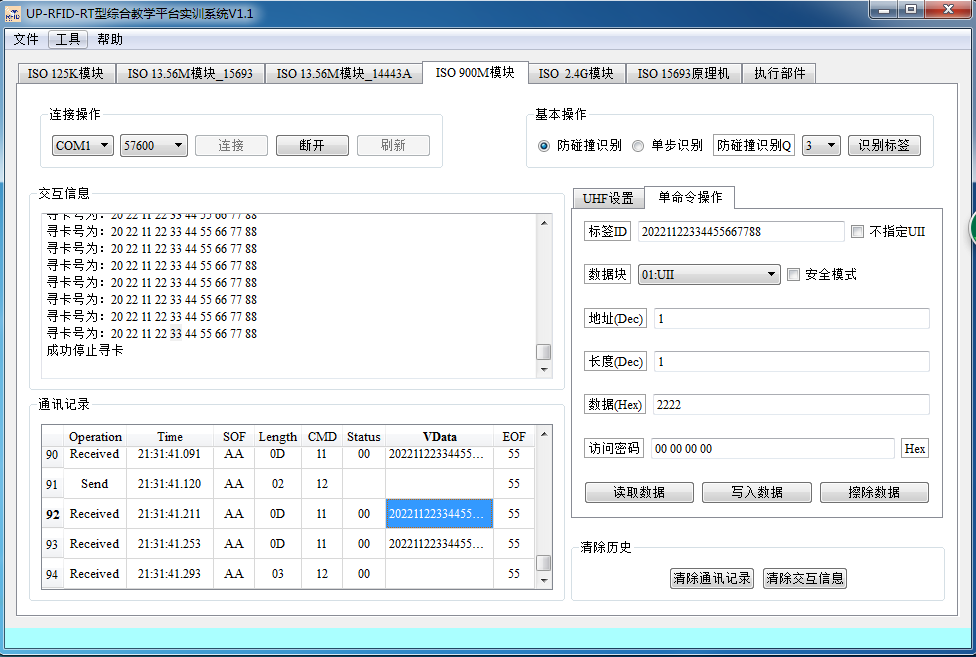


图4.9 卡号修改图

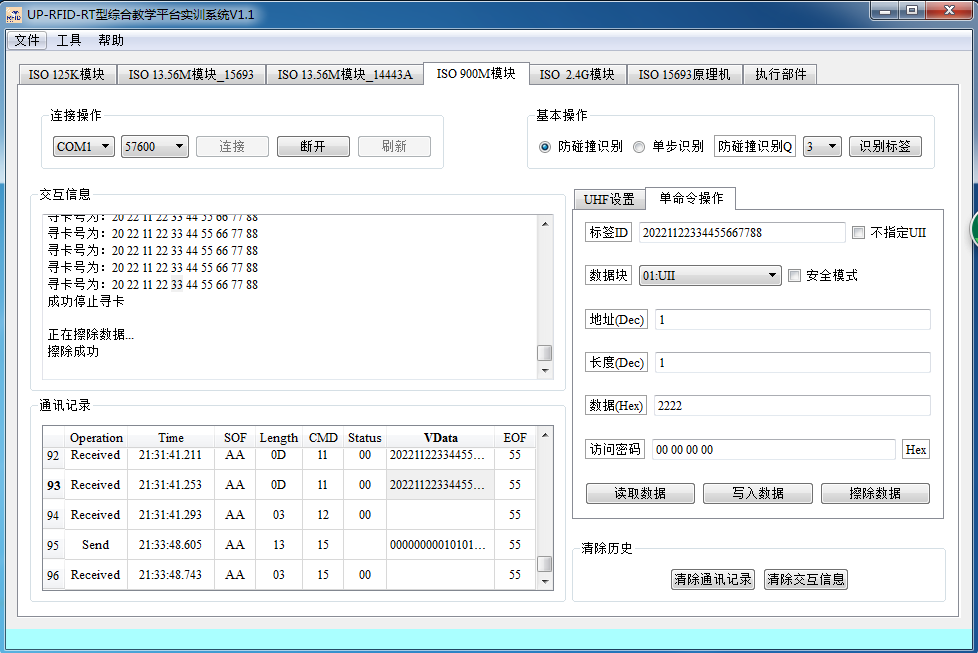


图4.10 擦除图



图4.11 修改0000图

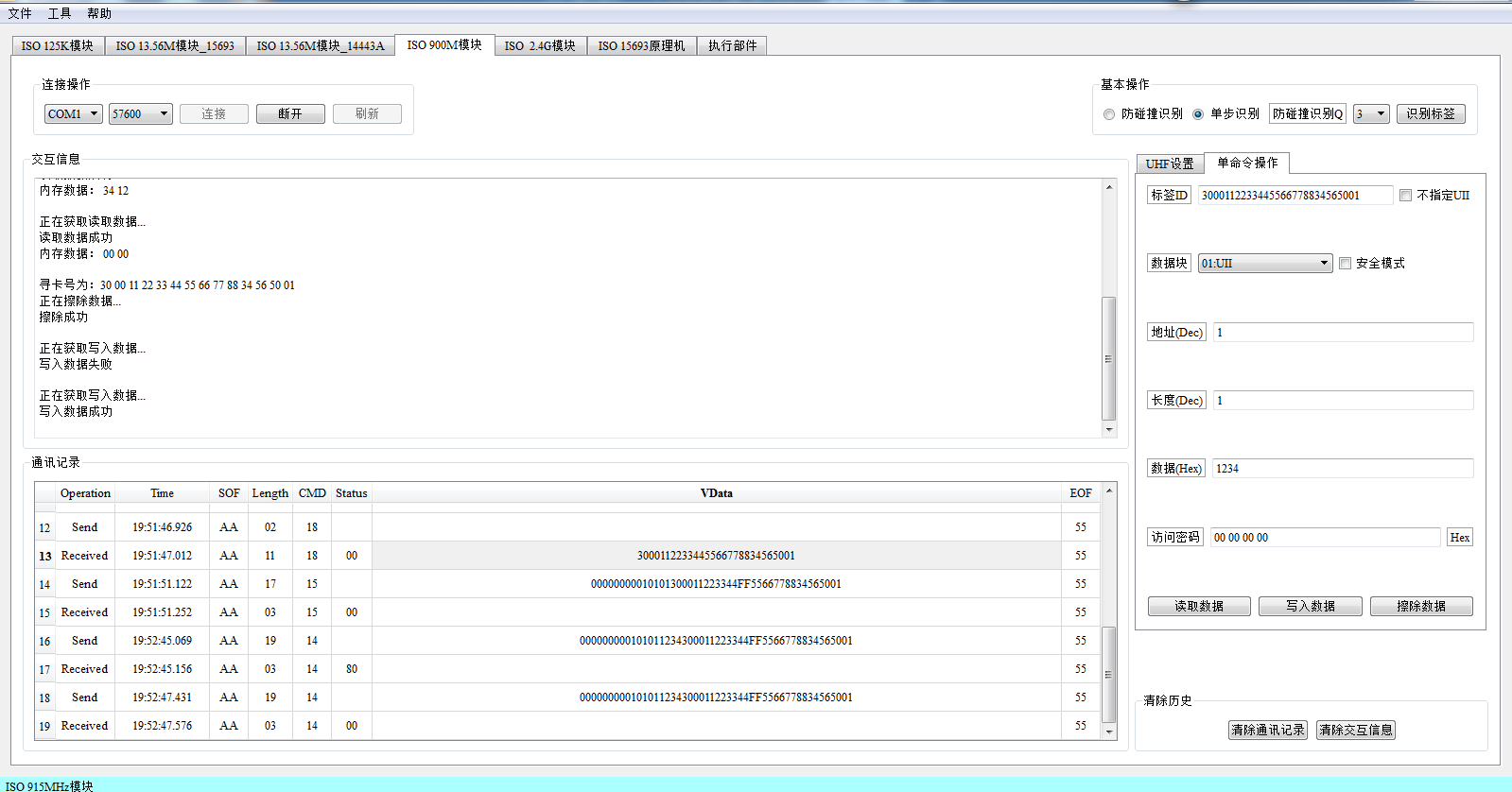


图4.12 读写擦除操作图

### UHF-900M开发实例



图4.13 UHF-900M图

## 核心源码说明

### 系统架构

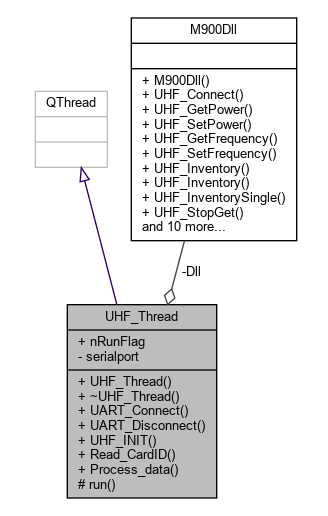


图4.14 UHF-900M接口图

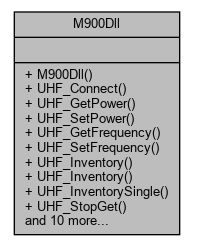


图4.15 M900DLL接口图

### UHF\_Thread接口定义

利用UHF\_Thread进行串口读写

#ifndef UHF\_THREAD\_H

#define UHF\_THREAD\_H

#include <QObject>

#include <QThread>

#include <QtSerialPort/QSerialPort>

#include <QtSerialPort/QSerialPortInfo>

#include "tools.h"

/\*添加dll链接库头文件

\* pro文件中添加：LIBS += -L D:\QtProject\900M\lib -l M900Dll\*/

#include "inc/m900dll.h"

#include "inc/m900dll\_global.h"

**class** UHF\_Thread:**public** QThread

{

    Q\_OBJECT

**public**:

**explicit** UHF\_Thread(QObject \*parent = 0);

    ~UHF\_Thread();

**bool** UART\_Connect(QString ComName, **int** Baudrate);

**bool** UART\_Disconnect();

**bool** UHF\_INIT();

**bool** Read\_CardID();

**void** Process\_data(**int** cmd, **char** data[], **int** length, **int** status);

**bool** nRunFlag;

**private**:

    M900Dll \*Dll;//dll链接库

    QSerialPort \*serialport;

signals:

**void** cardID(QString cardid);

**protected**:

**void** run();

};

#endif // UHF\_THREAD\_H

### 注册操作

判断是否文本框为空，此处进行字符串判断，并没有检测字符串的合法性，采用QMessageBox进行事件提醒。

同时对于数据库查询是否存在，如果存在进行更新，否则添加。未进行数据库异常处理。

**void** Regist\_Widget::Uhf\_Regist\_Button\_Click()

{

    //标签中显示的文本，用于在提示框中显示

**char** Label\_Name[][50] = {"卡号：","车牌号：", "金额：", "车型："};

    /\*如果文本框中存在空，则提示不能为空\*/

**for**(**int** i = 0; i < EDIT\_COUNT\_REGIST; i++)

    {

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

        {

**char** warning[256];

**sprintf**(warning,"%s不能为空",Label\_Name[i]);

            QMessageBox::warning(NULL, "warning", warning, QMessageBox::Yes, QMessageBox::Yes);

**return**;

        }

    }

    Sqlite sql;

    QSqlQuery Sqlite;

    /\*查询卡号是否已经存在\*/

**char** where[256];

**sprintf**(where,"cardID = '%s'",Edit[ID\_Regist]->text().toUtf8().data());

    Sqlite = sql.select("user", where);

**if**(Sqlite.next())

    {

        QMessageBox::warning(NULL, "warning", "已存在此卡", QMessageBox::Yes, QMessageBox::Yes);

**return**;

    }

    /\*将文本框中的信息添加到数据库\*/

    sql.add\_user(Edit[ID\_Regist]->text().toUtf8().data(), Edit[Plate\_number\_Regist]->text().toUtf8().data(), Types->currentText().toUtf8().data(), Edit[Balance\_Regist]->text().toInt());

    QMessageBox::warning(NULL, "warning", "注册成功", QMessageBox::Yes, QMessageBox::Yes);

    /\*清空文本框\*/

**for**(**int** i = 0; i < EDIT\_COUNT\_REGIST; i++)

    {

        Edit[i]->clear();

    }

}

## 思考题

1. 试从定性和定量两个方面分析读写器功率与标签读写距离的关系；
   1. 定性：功率越大读取标签的距离就越大，功率一定的情况下，距离越小越灵敏。
   2. 定量：采用dBm作为功率单位，所以功率的改变对读取的距离的影响为指数级别。
2. EPC 标签存储器分为哪几个区？各区有何功能？

分为Tag memory（标签内存）分为Reserved（保留），EPC（电子产品代码），TID（标签识别号）和User（用户）四个独立的存储区块（Bank）。

* 1. Reserved区：存储Kill Password（灭活口令）和Access Password（访问口令）。
  2. EPC区：存储EPC号码等。
  3. TID区：存储标签识别号码，每个TID号码应该是唯一的。
  4. User区：存储用户定义的数据。

1. EPC 标签可以通过哪些措施来保证各个存储区的信息安全？
   1. 状态安全

EPC标签通过有限状态机的思想来构建了七种状态，只有在指定的几个状态才能进行读写锁定等操作，同时进入操作的状态时，需要先经过Access Password的验证。

* 1. 数据安全

TID区：存储标签识别号码，每个TID号码应该是唯一的；User区：存储用户定义的数据。

1. EPC 和 TID 分别表示什么含义？二者结构有何特点？

EPC（Electronic Product Code），展品电子代码，对供应链中的对象进行全球唯一的标识；

TID，RFID电子标签识别号，是标签之间身份区别的标志，具有唯一性。

EPC码可以被复制，但TID是唯一的，具有一个Lock状态位，来防止出厂后被改写，才能被永久锁定。

## 实验体会与总结

通过试验箱对 Gen2 协议下标签指定存储区的数据读写，加深了对Gen2 协议下标签存储器结构的理解，了解了读写器功率和频率对电子标签读写的影响，了解了访问密码的用途和使用方法，了解了超高频读写器 API 函数的调用方法。

由于高频通信的硬件问题，造成许多卡不容易读出，此时可以考虑改变天线的朝向和相应角度进行读取。

对于Reader，尝试改变不同的读取参数(功率)，观察读取的距离，从而有了比较直观的体验。

实验代码对于不同的部分进行了解耦合和剥离，将工具类tools,sqlite等类进行了相应的区分，并创建线程进行串口读取，采用Timer机制进行轮训。但整个界面UI采用动态添加，在高频读取的时候，如果时间相应较频繁，其开销会较大，可考虑采用组件池的机制进行相应的组件管理。

前述实验熟悉了试验箱及串口通信和SQL/UI交互方式，为综合实验打下基础。

# 参考文献

1. 甘早斌，李开，鲁宏伟. 物联网识别技术及应用(普通高等教育“物联网工程专业”规划教材). 北京: 清华大学出版社, 2014.
2. 博创科技射频识别技术及应用实验指导书

、、