RFID 高级教学科研平台（II 型）

实 验 指 导 书

北京博创智联科技有限公司

2018-07-27

安全提示

非常感谢您购买博创科技产品！在打开包装箱后，请首先依据物品清单检查配件。若发现物品有所损坏、或是缺失情况，请尽快与您的当地经销商联络。

- 产品使用前，务必仔细阅读产品相关说明。
- 主板与电源连接时，请确认电源电压。
- 为了保证您的使用安全，请使用平台的专用电源。
- 接触平台主板前，应将手先置于接地金属物体上一会儿，以释放身体及手中的静电。
- 为避免人体被电击或产品损坏，在每次对主板、扩展卡进行拔插或重新配置时，须先关闭交流电源或将交流电源线从电源插座中拔出。
- 在对平台进行搬动前，先将交流电源线从电源插座中拔出。
- 当您需连接使用设备前，须检查确定设备电源线是否接好。
- 设备在使用过程中出现异常情况，请咨询和联系博创智联科技有限公司技术人员，电话：010-82110740 转 822/823，Email:support@up-tech.com。

版本声明

本文档为传感器原理与应用教学平台配套实验指导书。所述实验内容仅限传感器原理与应用教学平台使用。

日期	**修订版本**	**描述**	**编辑**	**备注**
2018-07-27	V1.0	Release Edition	Up-tech	@zyan

目录

第一章 RFID 高级教学科研平台（II 型）简介

RFID 高级教学科研平台（Ⅱ型）是北京博创智联科技有限公司研制的一款面向物联网工程专业核心课程《射频识别技术原理及应用》的综合教学实验平台。RFID 高级教学科研平台（Ⅱ型）表示 RFID 实验平台的型号说明。其中， UP 代表发行公司即北京博创智联科技有限公司， RFID 代表平台类型， Ⅱ型 代表区别于一代平台的升级版本。

1.1 平台设计思路

RFID 高级教学科研平台（Ⅱ型）是依据目前国内众多高校对物联网工程专业的最新要求，吸收国内、国外同类产品的优点，充分考虑高校 RFID 理论与实验教学的特点精心研制而成的一套综合教学实验系统。该系统覆盖了各种常用的 RFID 频段和 ISO 指令协议， 支持低频 125K、 高频 13.56M、 超高频 900M、微波 2.4G 四种 RFID 频段， 支持 ISO 18000-2、 ISO15693、 ISO 14443A/B、 ISO 18000-6C 等各种国际标准协议。

系统采用积木式、模块化的设计思想，每个频段对应一个独立的硬件模块，可以单独作为一个读写器使用，并配备了一套基于 STM32F407 的实训界面。通过实训界面与平台模块的相互通信，学生可以在平台显示屏上完成各个频段、各个协议的相关实验内容，帮助学生进一步深入理解 RFID 的技术原理和应用开发方法。

此外，系统还集成了 RFID 原理机、门禁闸机控制器。通过 RFID 原理机部分的实验，学生可以进一步学习和理解通信过程中的数据编码、载波信号、解调和调制方面的技术和原理。门禁闸机控制器主要是模拟门禁开关和闸机的设备，学生可以直接向闸机发送指令，从而模拟大楼或客房门禁管理系统或停车场收费管理系统。

1.2 产品外观

RFID 高级教学科研平台（Ⅱ型）实验箱箱体、箱体内部采用黑色， 平台底板均采用深蓝色。 打开 RFID 高级教学科研平台（Ⅱ型）实验箱， 取下实验箱上盖， 即可看到 RFID 高级教学科研平台（Ⅱ型）的全貌， 如图 1.3.1 所示。平台底板为蓝色， 模块为红色或者绿色， 各个模块均可拆下单独使用。

1.3 实验系统硬件结构

RFID 高级教学科研平台（Ⅱ型）主要有 RFID 原理机、低频 LF-125K 读卡器模块、高频 HF-13.56M（ ISO14443/ISO15693） 读卡器模块、超高频 UHF-900M 读卡器模块、微波有源 2.4G 读卡器模块、门禁闸机控制器模块、 7 寸真彩液晶触摸屏 组成。

图 1.3.2 为 125KHz 模块，用于简单的读取标签卡号。当该模块读取到不同的卡号时自动通过 RS232 接口发送出去。

图 1.3.3 为 13.56MHz 模块， 该模块支持 ISO 15693、 ISO 14443A 协议， 支持对 13.56MHz 的标签进行读写操作，包括读单块、 写单块、 读取 AFI、 写入 AFI、 锁定 AFI、 锁定数据块等一系列功能。

图 1.3.1 产品外观介绍

1、 低频 LF-125K 模块

图 1.3.2 RFID 高级教学科研平台（Ⅱ型）实验箱-125KHz 模块

2、 高频 HF-13.56M（ ISO14443A/ISO15693） 模块

图 1.3.3 RFID 高级教学科研平台（Ⅱ型）实验箱-13.56MHz 模块

3、超高频 UHF-900M 模块

图 1.3.4 所示为 900MHz 模块， 模块中间金属罩部分为核心部分， 金属罩有保护核心芯片和散热的作用，外部链接着可插拔的天线， 在进行读写卡的时候需要将卡靠近天线， 距离 1 到 2 厘米最佳（ 识别距离和功率有关， 请根据功率适当调节距离）。

图 1.3.4 RFID 高级教学科研平台（Ⅱ型）实验箱-900MHz 模块

4、微波有源 2.4G 模块

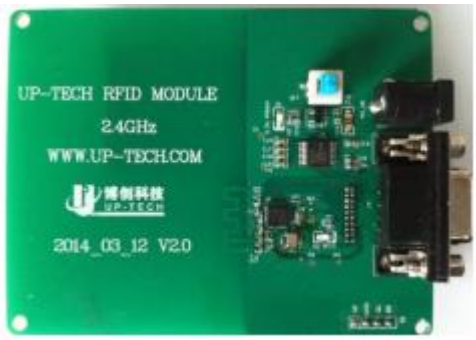

图 1.3.5 RFID 高级教学科研平台（Ⅱ型）实验箱-2.4GHz 模块

图 1.3.5 为微波 2.4GHz 模块。 2.4GHz 模块主要用在智能交通、 物流等领域。 结合有源标签， 可以在 5 米之外识别标签。 2.4GHz 模块与 125K 模块一样， 识别卡号后， 直接通过 RS232 接口将卡号发送出去。

5、微波有源 2.4G 标签

图 1.3.6 RFID 高级教学科研平台（Ⅱ型）实验箱-2.4GHz 有源标签

图 1.3.6 所示为微波 2.4GHz 的有源标签， 标签配合 2.4GHz 模块使用， 标签下面有一个纽扣电池，为标签提供电源， 标签周期性的上报自己的卡号。 标签上有电源开关， 实验完毕后最好关闭以节约电能。用其它厂商的 2.4GHz 有源标签模块。 需要烧写本平台指定的程序， 和接收模块相匹配方能使用。

6、RFID 原理机

图 1.3.7 所示为 RFID 原理机， 配备原理机的主要目的是让学生更深入地学习和理解 RFID 技术原理， 采用 ISO 15693 协议， 提供多个测试点， 用于波形观测、 原理分析， 原理机的显示屏用于显示当前读取到的卡号。同时，原理机还对外提供 RS232 接口，学生可以通过原理机模块设计和开发一些 RFID 的相关应用。

图 1.3.7 RFID 高级教学科研平台（Ⅱ型）实验箱-原理机模块

8、 门禁闸机控制器

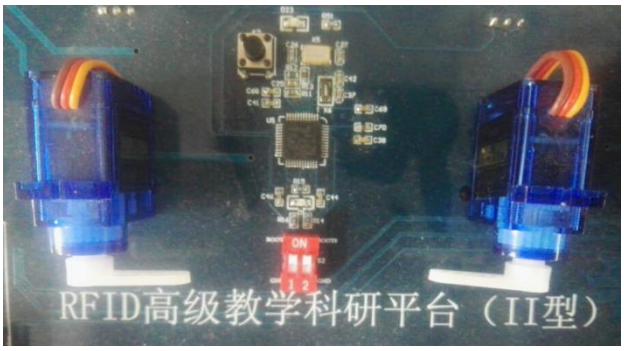

图 1.3.8 RFID 高级教学科研平台（Ⅱ型）实验箱-执行部件

图 1.3.8 所示为门禁闸机控制器， 中间黑色的芯片为 STM32F103 芯片， 上面的红色的拨码开关用于选择给 STM32 烧写程序的方式， 通过 STM32 实现闸机旋转的控制。

1.4 软件层次

图 1.4.1 工程文件夹与软件层次

软件设计采用分层架构，驱动设计由易到难。依据 MCU 开发功能分层，将 MCU 以及 MCU 片内外设归为 HAL(Hardware Abstract Layer，硬件抽象层)，将板载 Devices 依据功能归为 FML(Functional Module Layer，功能模块层)，将用户的应用程序以及更高层的 RTOS、FileSystem、GUI 等归为 AL(Application Layer，应用程序层)。并且在组织工程文件时，将 HAL 层的固件库和驱动包放到 Chip 文件夹，将板载的 Devices 驱动放到 Board 文件夹，将用户应用程序等放到 App 文件夹，将工程文件放到 Protect 文件夹，同时还会添加 Document 文件夹来放置说明文档，如有必要还会添加 Source Insight 工程文件夹 Readcode。这样的分层在 MCU 开发中非常有益，不仅仅在后期的代码维护和升级方面更具有显著优势，而且梳理清楚了各层次顺序、方便代码重用和移植。对于初学者来说，有效的分层、合理的架构，更能培养新手的系统意识，在接触和开发更大的系统工程时具有好的软件解决方案。

在平台的实验事例开发中，我们研发人员力求代码精简、注释规范。我们摒弃了市场上一些机构的大而杂的设计思路，认为初学者对嵌入式知识了解甚少，应当“说此就此”。因此我们的工程在整合好半导体厂家的固件库（HAL 层）后，每个单独驱动小程序都极其简练，希望老师和同学可以一次掌握各个驱动的要点。在注释方面，我们遵循本公司的代码注释规范，全中文注释，易于学习者理解接受。也希望学习者能够养成代码精简、注释规范的良好变成习惯。

总体来说，本实验平台无论是从硬件配置，还是软件设计方面来说，都是最先进、最优秀的。硬件方面的高配置，软件方面的分层架构和良好的代码规范设计，旨在为自动化、信息、电气工程等非计算机专业师生提供更实用易学、更贴近市场、更具成长力的嵌入式教学科研平台。

第二章 低频 LF-125K 实验及实例

2.1 低频 LF-125K 读卡实验

图 2.1.1 RFID 高级教学科研平台（Ⅱ型）-125KHz 模块

1、实验环境

- 硬件：RFID 高级教学科研平台（Ⅱ型）实验箱，PC 机
- 软件：Keil μVision5 IDE，低频 LF-125K 读卡器 emWin 测试程序

2、实验内容

- 了解 125k 的基本概念、国际标准、协议内容
- 了解 125k 的标准接口
- 了解 125k 的应用范围及领域

3、实验原理

3.1 LF-125K 技术基础

125KHz RFID 系统采用电感耦合方式工作。 由于应答器成本低、 非金属材料和水对该频率的射频具有较低的吸收率， 所以 125KHz RFID 系统在动物识别、 工业和民用水表等领域获得广泛应用。

3.2 e5551 应答器芯片

1、 e5551 芯片的性能和电路组成

（1） 主要技术性能

e5551 芯片是 Atmel 公司生产的非接触式、 无源、 可读写、 具有防碰撞能力的 RFID 器件， 中心工作频率为 125K 。 具有以下主要特性：

- 低功耗，低工作电压

- 非接触能量供给和读写数据
- 工作频率范围为 100~150KHz
- EEPROM 存储器容量为 264bit， 分 8 块， 每块 33 位
- 具有 7 块用户数据， 每块 32 位， 共 224 位
- 具有块写保护
- 采用请求应答实现防碰撞
- 完成块写和检验的时间小于 50ms
- 可编程选择传输速率和编码调制方式
- 可工作于密码方式

（2）内部电路结构

e5551 芯片的内部电路组成框架图如下， 该图给出了 e5551 芯片和读写器之间的耦合方式。 读写器向 e5551 芯片传送射频能量和读写命令， 同时接收 e5551 芯片以负载调制方式送来的数据信号。

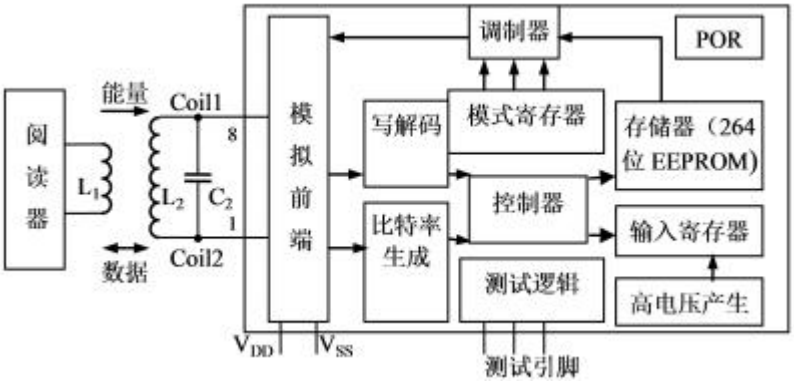

图 2.1.2 e5551 内部电路组成框架

e5551 芯片由模拟前端、 写解码、 比特率产生器、 调制器、 模式寄存器、 控制器、 测试逻辑、 存储器、编程用高压产生器等部分构成。

e5551 芯片在射频工作时， 仅使用 coil1（ 引脚 8） 和 coil2（ 引脚 1）， 外接电感 L2 和电容器 C2， 构成谐振回路。 在测试模式时， VDD 和 VSS 引脚为外加电压正端和地， 通过测试引脚实现测试功能。

① 模拟前端（ 射频前端）

模拟前端（ analog front end， AFE） 电路主要完成芯片对模拟信号的处理和变换， 包括电源产生、 时钟提取、 载波中断的检测、 负载调制等部分。

② 控制器

控制器主要完成 4 种功能：

A.在上电有效后及读期间， 用配置存储器数据装载模式寄存器， 以保证芯片设置方式工作；

B.控制对存储器的访问；

C.处理写命令和数据写入；

D.在密码模式中， 将接收操作码后的 32 位值与存储的密码进行比较和判别。

③ 比特率生成与写解码

比特率生成电路可产生射频的 8， 16， 32， 40， 50， 64， 100， 128 分频后的数据比特率。写解码电路在写操作期间解读有关写操作码， 并对写数据流进行检验。

④ 高压（ HV） 产生器

它在写入时产生对 EEPROM 编程时所需的高电压。

⑤ 模式寄存器

模式寄存器存储来自 EEPROM 块 0 的模式数据， 它在每块开始时被不断刷新。

⑥ 调制器

调制器由数据编码器和调制方式两级电路组成，如图 2.3.2 所示。其输入为来自存储器的二进制 NRZ 码，输出用于对载波的负载调制。

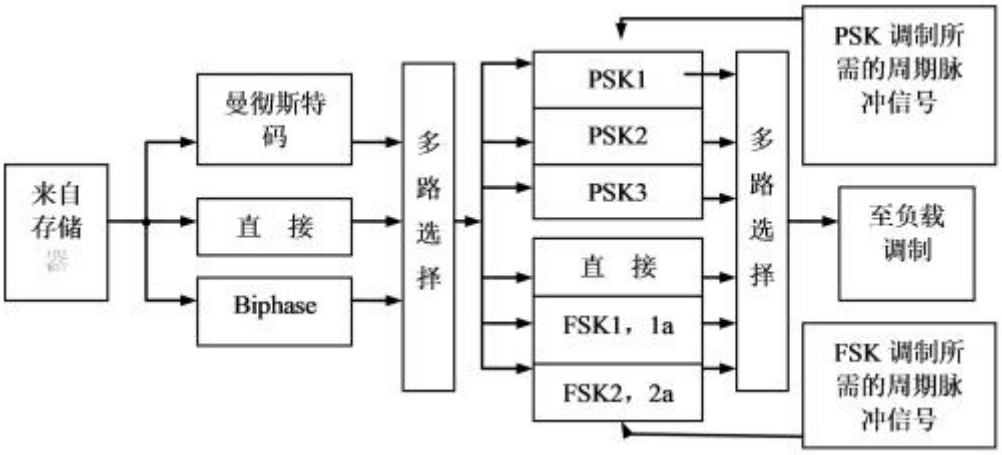

图 2.1.3 调制器电路组成

A.编码

♥曼彻斯特码： 逻辑 1 为倍频率 NRZ 码的 10， 逻辑 0 为倍频率 NRZ 码的 01；

♥Biphase： 每个位的开始电平跳变， 数位 0 时位中间附加一跳变

B.调制方式

PSK 调制的脉冲频率为 RF/2， RF/4 或 RF/8， RF 为载波频率 fc。 它的相位变化情况有以下：

♥PSK1： 数位从 1 变为 0 或从 0 变为 1 时， 相位改变 180；

♥PSK2： 每当数位 1 结束时， 相位改变 180；

♥PSK3： 数位从 0 变为 1（上升沿） 时， 相位改变 180；

FSK 调制有以下 4 种：

♥FSK1： 数位 1 和 0 的脉冲频率为 RF/8 和 RF/5;

♥FSK1a： 数位 1 和 0 的脉冲频率为 RF/5 和 RF/8;

♥FSK2： 数位 1 和 0 的脉冲频率为 RF/8 和 RF/10;

♥FSK2a： 数位 1 和 0 的脉冲频率为 RF/10 和 RF/8;

C.注意问题

下面的组合不可使用：

♥当编码为曼彻斯特码或 Biphase 码时，调制为 PSK2 或比特率为 RF/8 且脉冲频率为 RF/8 的 PSK 调制；

♥比特率为 RF/50 或者 RF/100 的 PSK 调制， PSK 的脉冲频率不为比特率的整数倍。

2、存储器

存储器 EEPROM 的结构如图， 它由 8 块构成， 每块 33 位， 第 0 位为锁存位， 共 264 位。所有 33 位都可被编程， 编程所需电压来自片内。 但若某块的锁存位被置 1， 则该块被锁存， 不能通过射频再次编程。

存储器 EEPROM 的结构如表 2.1.1 所示。

0	1~32	块
L	用户数据或密码（口令）	7
L	用户数据	6
L	用户数据	5
L	用户数据	4
L	用户数据	3
L	用户数据	2
L	用户数据	1
L	配置数据	0

表 2.1.1 存储器 EEPROM 结构

块 0 为芯片工作的模式数据， 它不能作为通常数据被传送， 块 1 至块 6 为用户数据； 块 7 为用户口令，若不需要口令保护， 则块 7 也可作为用户数据存储区。

存储器的数据以串行方式送出， 从块 1 的位 1 开始到最大块（ MAXBLK） 的位 32， MAXBLK 为用户设置的最大块号参数值。 各块的锁存位 L 不能被传送。

- 配置存储器

EEPROM 的块 0 用于存放配置数据。 其各位的编码含义如下页表。

- 初始化

电源上电后， e5551 芯片按配置数据进行初始化（ 需 256 个载波时钟周期， 约 2ms）， 采用所选用的编码调制方式工作。

配置存储器的配置数据编码如表 2.1.2 所示。

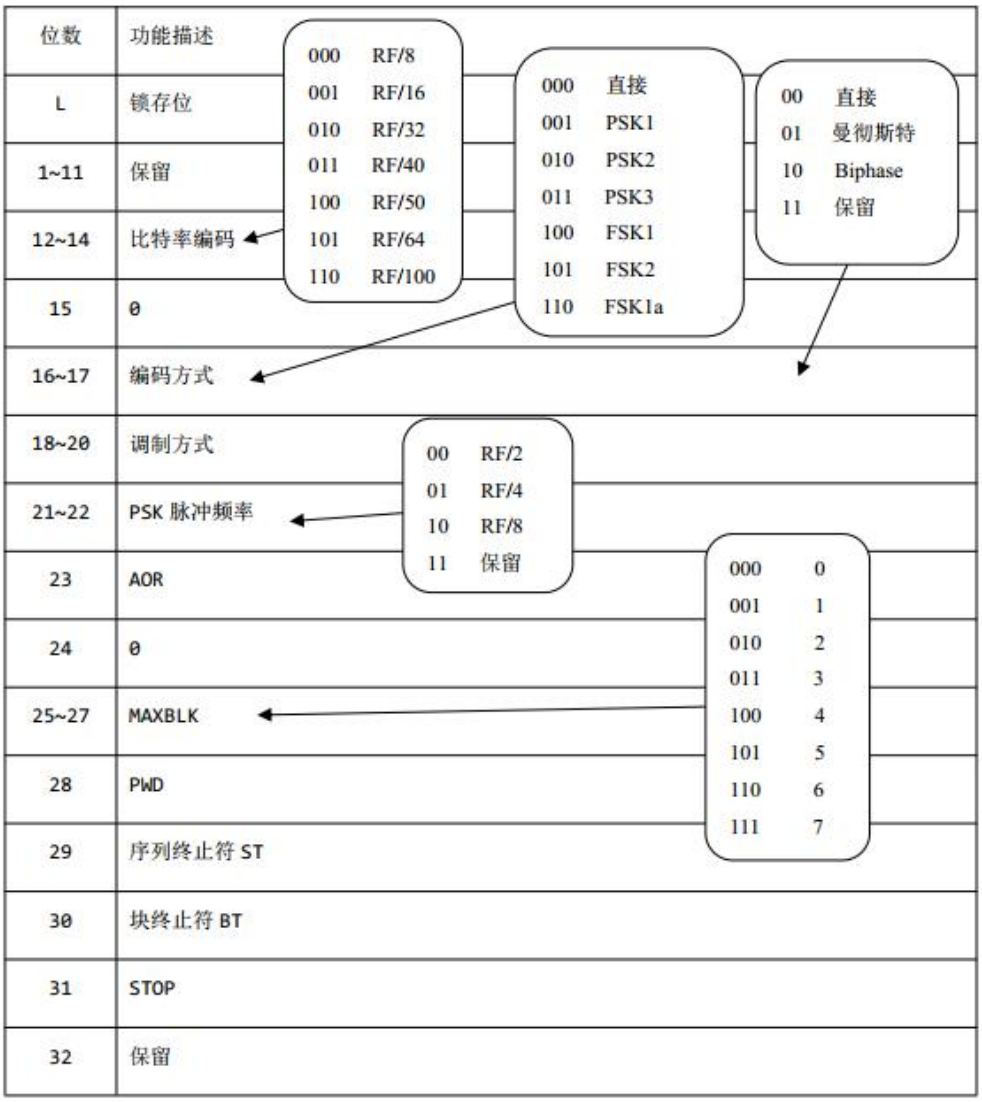

表 2.1.2 配置存储器的配置数据编码

4、实验步骤

1) 准备好硬件， RFID LF-125KHz 模块和配套 125K 标签。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 将标签放入读卡器天线辐射区内，可以读取卡号，GUI 显示如图 2.4.2 所示。

5) 125K 模块为卡号自动上报，所以只需要接通，即可看到卡号信息。

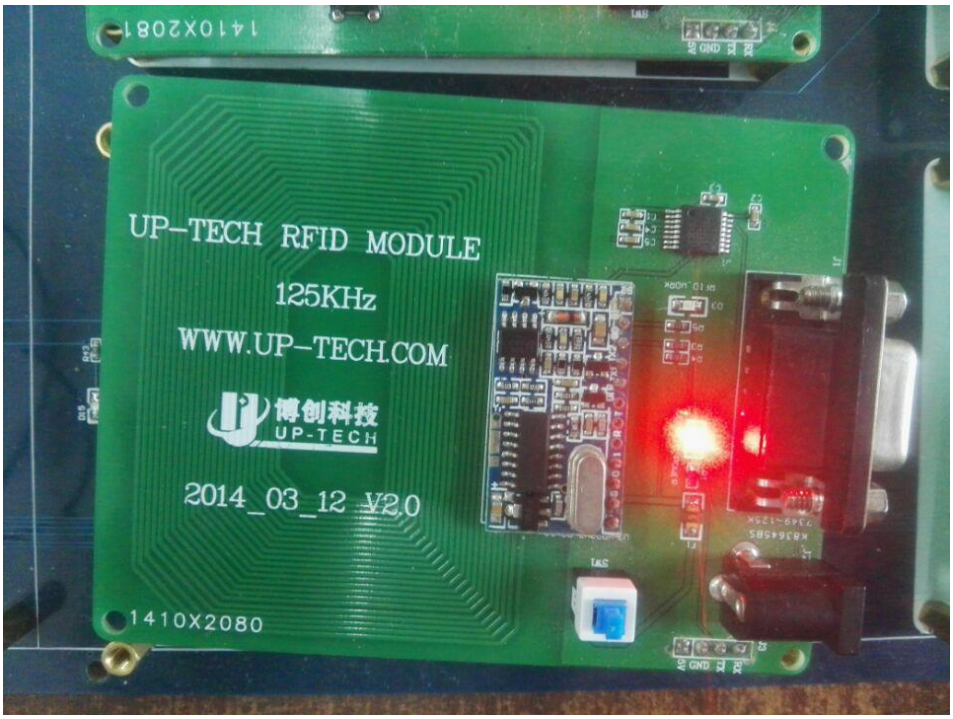

图 2.1.4 125KHz 模块选中状态

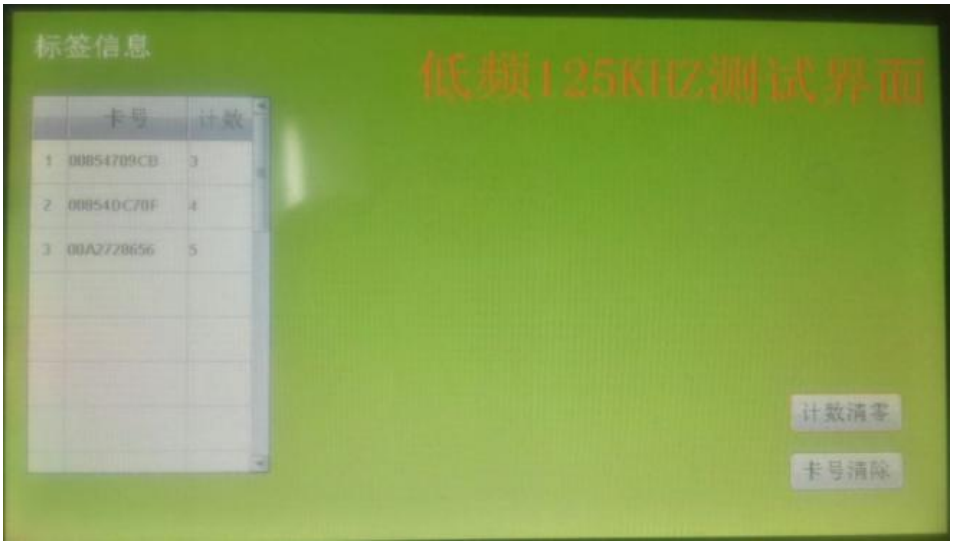

图 2.1.5 125KHz 测试界面

2.2 低频 LF-125K 开发实例

1、实验步骤

- 掌握 LF 125K RFID 模块串口通信协议
- 了解 LF 125K RFID 模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用
- 了解 125k 的应用范围及领域

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、实验内容

- 采用 RFID 高级教学科研平台（Ⅱ型）的 125KHz 模块作为读卡器,开发一个基于 emWin 的刷卡软件。

4、实验原理

STemWin 简介

SEGGER 公司做的 emWin 这个 GUI，也有叫μCGUI 的，这两个是一个东西。STemWin 是 SEGGER 公司授权给 ST（意法半导体）的。使用 ST 芯片的用户可以免费使用 STemWin。其实不光授权给了 ST，还有 NXP， Energy Micro 等。凡是使用这些芯片厂商生产的处理器都可以免费的使用 emWin。但是出于一定的保护措施，使用 STemWin 的库是不能用在其它芯片厂商的处理器上面的。因为在工程初始化 STemWin 前要使能 CRC 校验。如果没有使能，STemWin 是启动不起来的。 KEIL MDK 的安装目录里面也带有 emWin 软件包，这个软件包也不可以直接使用，用户需要给 KEIL MDK 注册 RL-ARM 才可以使用。

这里 STemWin 还针对 ST 的微控制器做了专门的优化，比如在使用 ST 的 F4XX 微控制器带 FPU 的芯片时， STemWin 在需要浮点处理的地方专门做了优化。

STemWin 特性

- 强大的消息机制

尽管消息机制是 STemWin 的核心，但是 STemWin 手册中讲解的确很少， 而且 STemWin 也是没有源码的，深入学习比较困难，不过没有关系，并不影响使用。

对于 STemWin 的消息机制，简单的理解就是这样：比如操作一个触控界面， 上面有按钮，滚动条，编辑框等控件， 当用户去触摸某个控件时都会触发窗口管理器去处理这些消息，并跳转到窗口回调函数的相应消息里面， 这些消息里面就是需要添加的功能。 比如想点击按钮后实现 LED 翻转， 可以在按钮所

在窗口回调函数的按钮消息中加入 LED 翻转功能就可以实现这种效果。 具体消息是如何传输的，用户不需要去管，只需在回调函数相应的消息里面加入功能就可以了。从这个角度来看， 消息机制还是比较容易掌握的。

对于初学者来说，明白了这点就可以了， 通过后面实际的例子来进一步加强认识。 另外，要实现消息机制就得有消息结构变量用来指示消息类型和一些其它相关的功能，下面讲解这部分知识。

❖ 丰富的控件及 API

STemWin 中提供了十几种控件，每种控件配备了大量 API 函数，可以满足用户的使用需求。

❖ 可视化编程简化开发流程

STemWin 提供一款官方模拟器——GUIBuilder。GUIBuilder 把 STemWin 中的控件图形化，用户可以在里面直接拖动控件并设置相应参数，可视化操作省去繁琐的代码编写，搭好界面后保存，模拟器会自动生成 C 代码文件，此时只需要把这个代码移植到核心板的 STemWin 程序中去。用户熟悉这一流程后，开发 STemWin 界面的速度会极快。

5、实验步骤

❖ 开发工具

我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

❖ 创建工程

我们直接使用现有的工程模板，这个模板已经集成了 RFID 高级教学科研平台（Ⅱ型）STemWin 正常运行所需要的驱动程序，并完成了 STemWin 向平台的移植，用户只需要在模板的基础上编写通信协议相关程序和进行界面开发工作。

如此设置旨在突出平台教学重点，略去 STemWin 运行需要驱动部分的开发及 STemWin 的移植，直接进入到射频通信和 STemWin GUI 开发环节，为广大师生降低学习成本。

❖ UI 设计

打开光盘中 emWin 工具的文件夹，找到 GUIBuilder 打开，如图 2.2.1 所示，用户拖动控件完成界面初步设计，能添加的控件在程序上部，直接拖下来即可。拖下来后还可以进行一些参数的设置，参数设置在左下角，可以进行数值的修改，左下角显示的参数不一定是控件的全部参数，用户可以右键点击控件，选项中可能会有一些新的参数，点击它们就可以添加到左下角进行参数修改了。

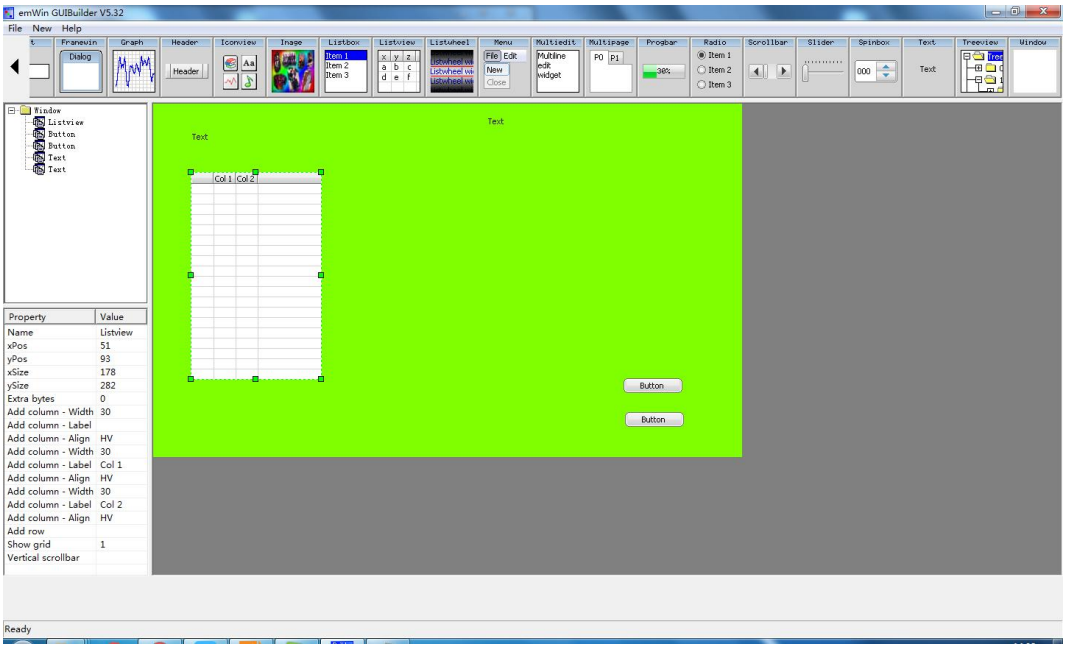

图 2.2.1 在 GUIBuilder 中完成可视化编程

在 GUIBuilder 中设计好界面左上角点击 File->Save...保存生成的 C 文件到 GUIBuilder.exe 所在文件夹下。如图 2.2.2 找到生成的 C 文件位置。

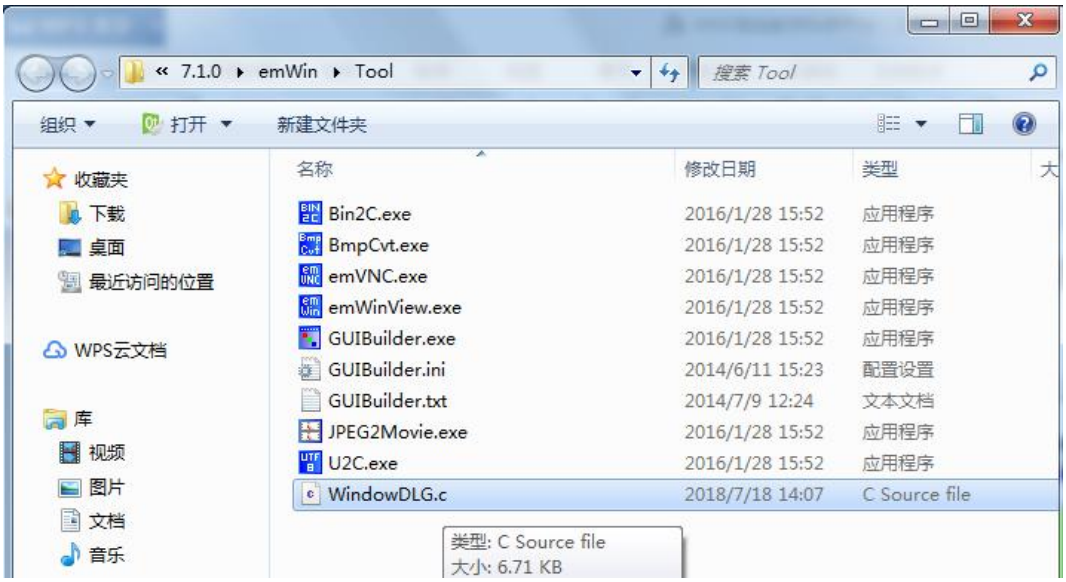

图 2.2.2 找到生成的 C 文件

把文件中除头文件外的所有代码复制粘贴到工程模板中的 MainTask.c 文件下，宏定义中的控件编号默认从“GUI_ID_USER + 0x00”开始，如果在一个程序中用到了多个窗口，这个编号就有可能会重复，需要用户手动修改，修改方法也很简单，GUI_ID_USER 是 STemWin 文件中宏定义的固定数值，用户只需要修改加号后面的数字，把重复编号的控件 ID 错开，不要忘记再把宏定义名字最后代表控件编号的数字做相应修改即可。

编译一下文件，发现一个错误。

```
.\Objects\emWin.axf: Error: L6218E: Undefined symbol GUI_MAKE_COLOR (referred from maintask.o).
Not enough information to list image symbols.
Finished: 1 information, 0 warning and 1 error messages.
".\Objects\emWin.axf" - 1 Error(s), 1 Warning(s).
Target not created.
Build Time Elapsed:  00:00:04
```

图 2.2.3 复制粘贴过去编译报错

定位到报错的函数处，发现是我们在模拟器中设置了背景颜色的地方，造成这种情况的原因是模拟器和我们使用的工程模板 emWin 版本不匹配，模拟器中的一些 API 函数在工程模板中可能没有，因此报错。这里我们要做的是删掉这个报错的函数，其中的参数保留。

编译文件通过，烧写到核心板，发现 LCD 屏是黑色的，没有出现模拟器中我们做的界面。这是因为尽管我们复制了整个窗口管理器及回调函数过去，仍需要在主函数“void MainTask(void)”中执行一步创建窗口的操作，不添加这行代码界面不会工作，创建窗口的函数模拟器已经给出，在模拟器代码最下方“WM_HWIN CreateWindow(void);”声明的就是这个函数。添加好代码，重复编译烧写步骤，可以看到模拟器中的界面已经显示到平台的 LCD 屏幕上面了。

```
179 /*********************************************************************
180 *
181 *       MainTask
182 *
183 *********************************************************************
184 */
185 void MainTask(void)
186 {
187     GUI_Init();
188     WM_SetDesktopColor(GUI_BLACK);      /* Automacally update desktop window */
189     WM_SetCreateFlags(WM_CF_MEMDEV);  /* Use memory devices on all windows to avoid flicker */
190     CreateWindow();
191     while(1)
192     {
193       GUI_Delay(1);
194     }
195 }
196
```

图 2.2.4 主函数中创建窗口

这时大家可能注意到一些问题，比如模拟器中无法显示中文，字体太小看不清，控件大小到了实际屏幕发现不合适想要进行调整等。不用担心，这些问题都可以通过修改代码进行调整。控件大小问题每个控件都提供了相应的 API 函数，具体用法可以查阅《emWin5 中文手册》，中文字体显示和字体大小修改可以通过自定义字库设置，下面会针对如何生成自定义字库做专门讲解。

❖ 自定义字库

emWin 自带字库不支持汉字，因此需要配置自定义字库文件，配置过程仅在此节进行说明，文档后续章节不再重复。

生成自定义字库用到软件 FontCvtNXP.exe，同样在 emWin 工具文件夹下。

（1）新建记事本，在里面输入所有你要用到的字符，另存为 Unicode 编码。

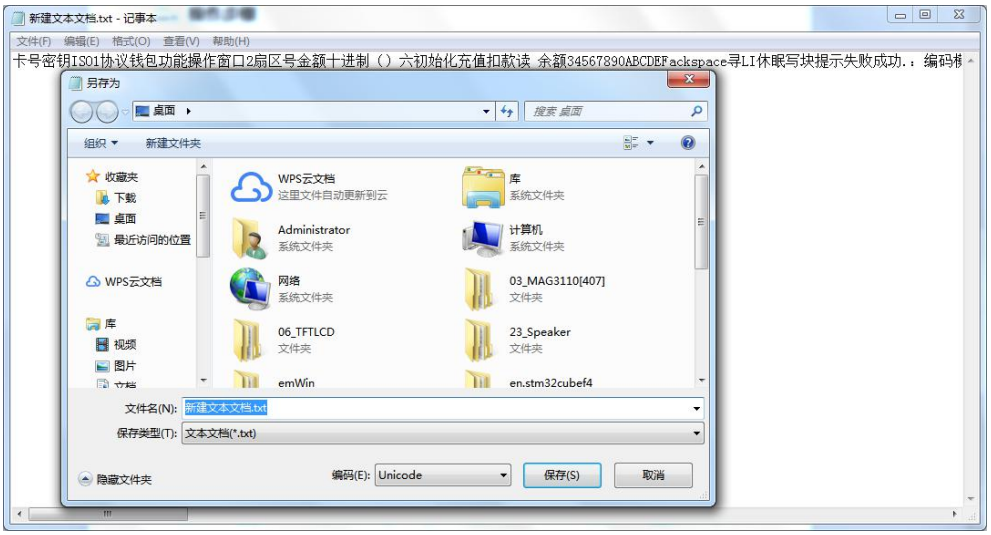

图 2.2.5 另存为选择 Unicode 编码

（2）打开 FontCvtNXP.exe。

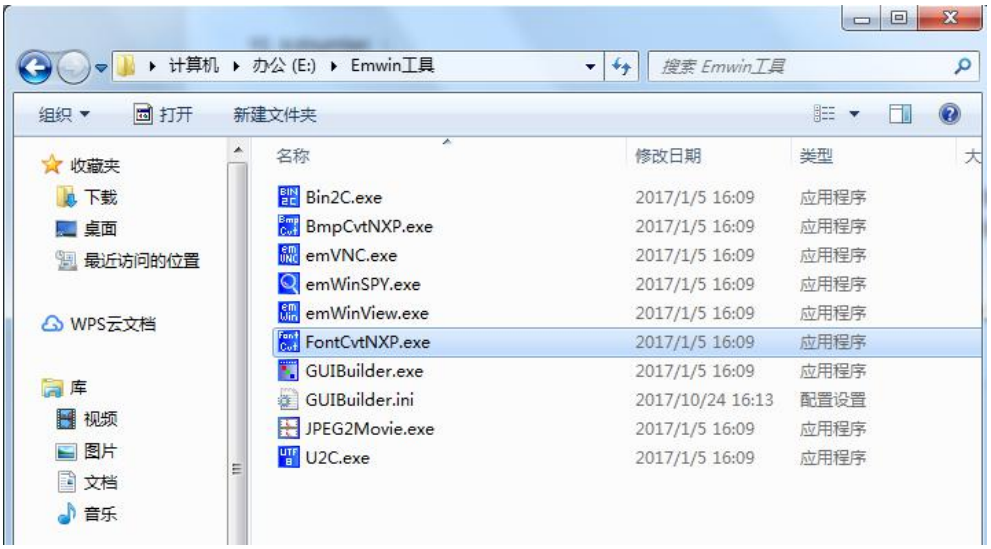

图 2.2.6 字体工具位置

（3）有多种模式和编码供用户选择，本节实验选择标准模式、”16 Bit UNICODE”编码。选好单击 OK。

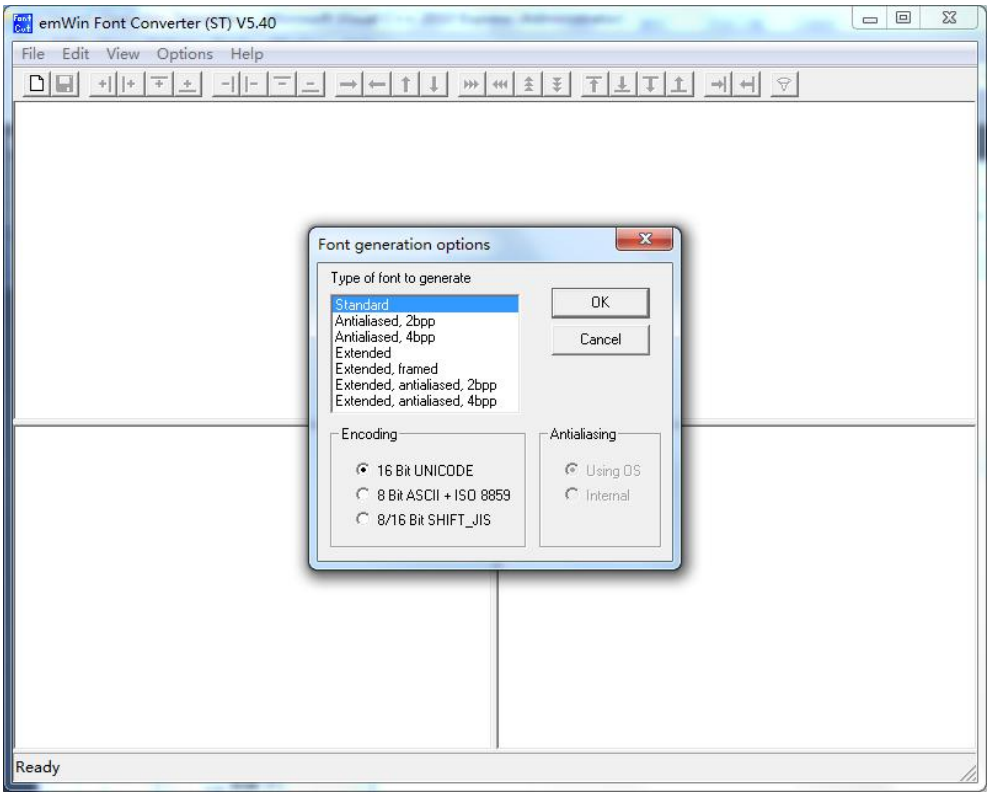

图 2.2.7 编码选择

（4）字体选择宋体，字形常规，大小 16，点阵模式选像素。单击确定。

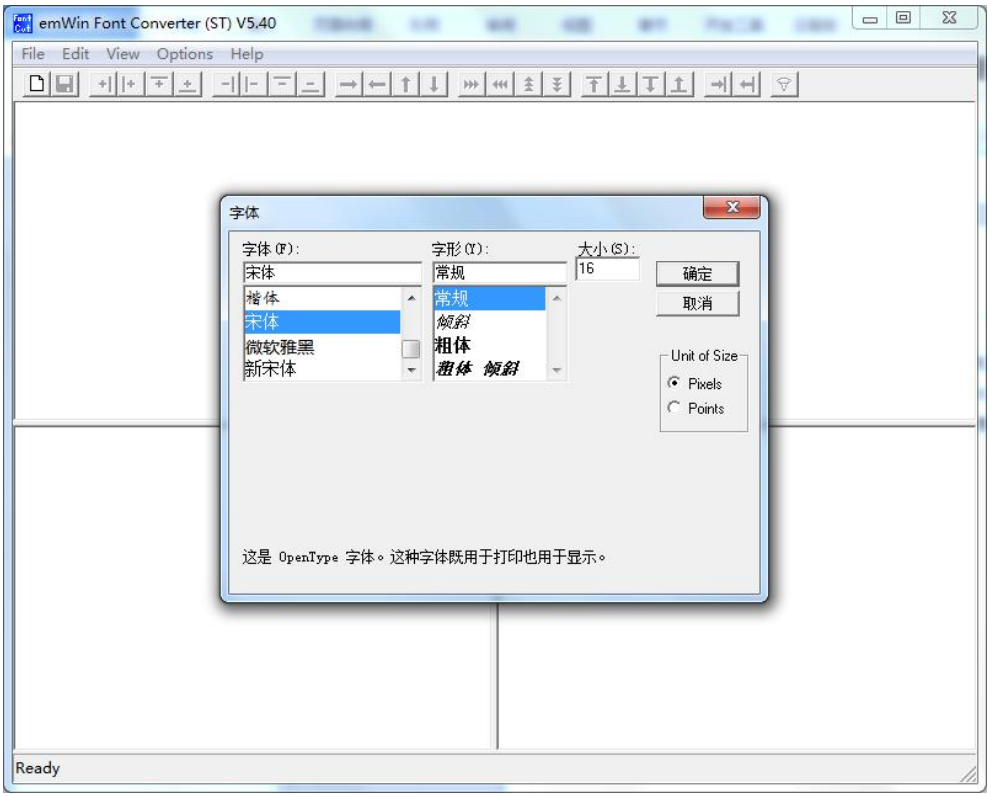

图 2.2.8 字体选择

（5）菜单栏”Edit”->’ ’Disable all characters”，失能所有字符，如图 2.2.9 软件上所有字符变灰。

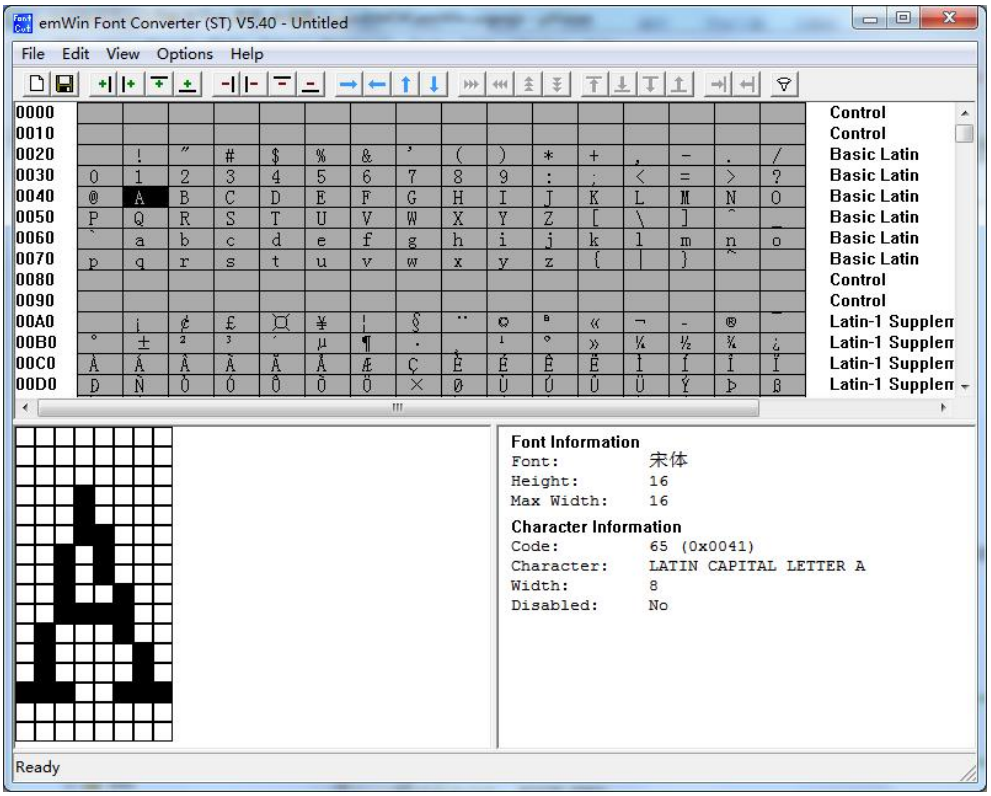

图 2.2.9 失能全部字符

（6）菜单栏”Edit”->’ ’Read pattern file...”，选择步骤 1 保存的.txt 文件，如图 2.2.10 文件里有的字符在软件里被使能（由灰变白）。

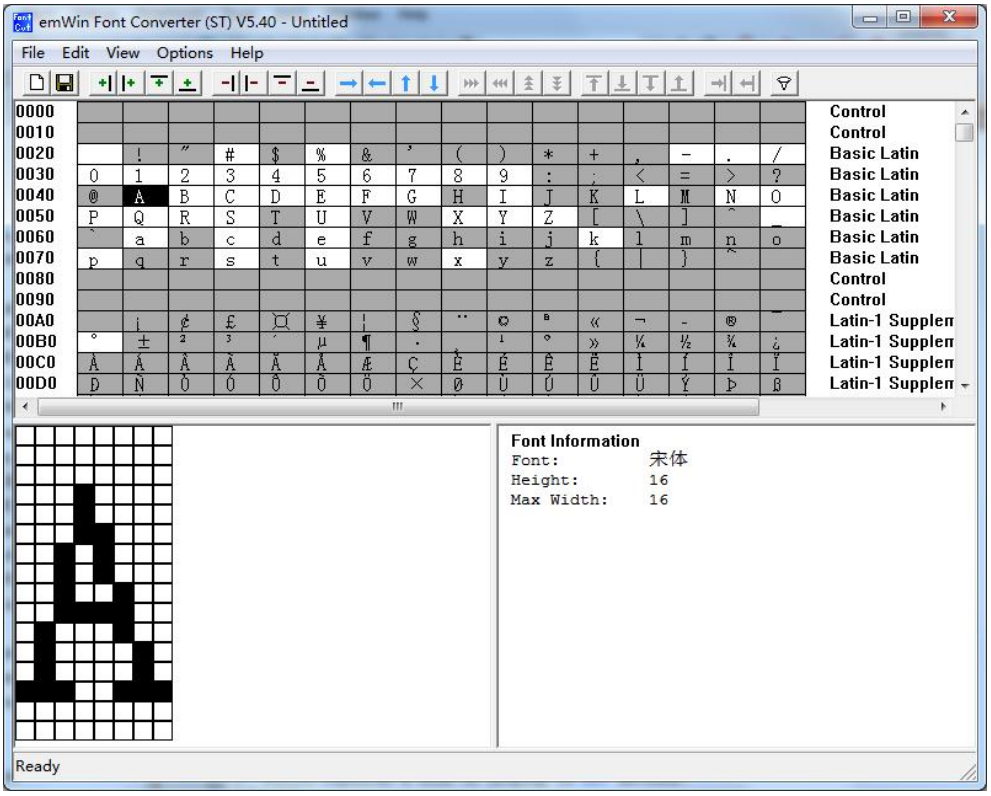

图 2.2.10 使能用户需要的字符

（7）另存为.c 文件，放到 App/fonts 文件夹下，并添加到 MDK 工程中。

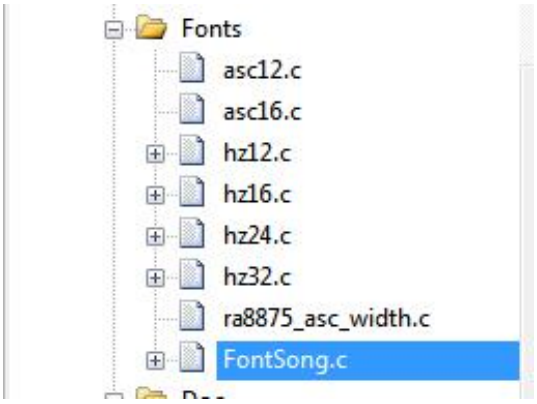

图 2.2.11 字体文件添加至 MDK 工程

（8）在使用前先使能 UTF-8 编码，”GUI_UC_SetEncodeUTF8();”写在使用字体前；在使用字体的文件开头加一句”extern GUI_CONST_STORAGE GUI_FONT GUI_Font 文件名;”，如字体文件叫”FontSong.c”，则加的语句是”extern GUI_CONST_STORAGE GUI_FONT GUI_FontFontSong;”，后面在使用控件字体设置函数时，字体参数项填写”&GUI_FontFontSong”调用用户生成的字库。

❖ 编码

先看通信部分，硬件上平台使用 ST16C554 扩展串口，125K 读卡器使用 16C554 的 CS1B 口，波特率 9600bps。波特率到 16C554 的头文件中修改，如图 2.2.12，修改 ST16C554_DLL_VAL 这个宏定义的数值，这个是分频值，可以看到上面注释中 115200bps 的分频值是 0x06，总频率除以分频值是波特率，因此分频值和波特率成反比，计算得出 9600bps 的分频值是 72。

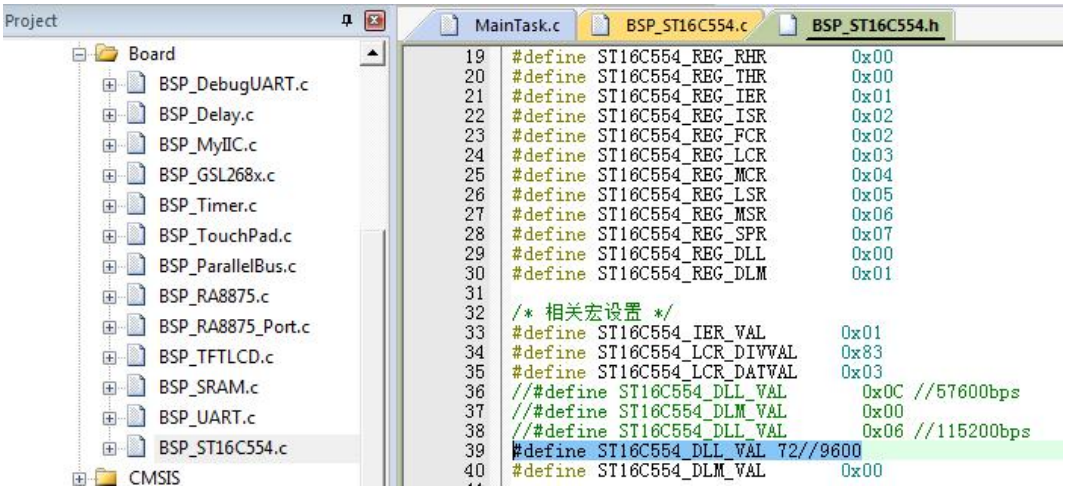

图 2.2.12 修改波特率

协议解析采用在线解析模式，即边接收边解析，直接在中断函数里完成。在线解析使用经典的状态机模式，结合 C 语言的 switch case 语句，方便快捷的在数据接收过程中完成状态跳转，本模块最后使用了异或校验，完成校验才会被认为是有效帧，否则直接丢弃。

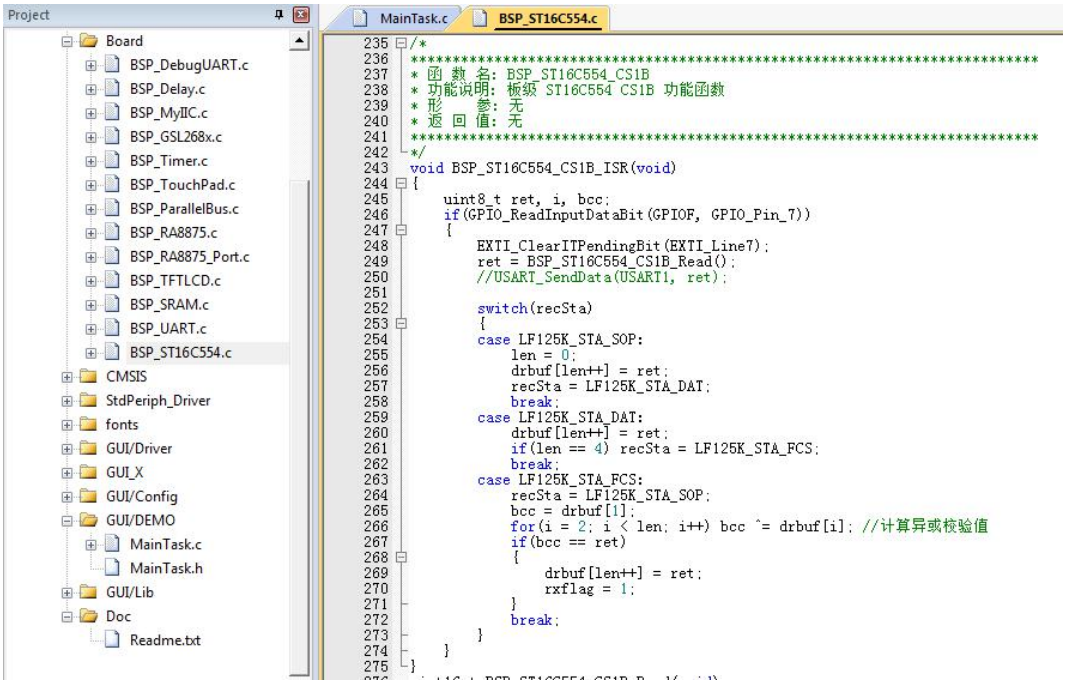

图 2.2.13 中断函数内在线解析

卡号计数在主函数实现，中断收到完整一帧在主函数中进行计数，不同的标签有独有的计数器，用于记录它们一共被用到几次。

```
          if(rxflag)                                    //读到新卡
          {
             cnt = 0;
             flg = 0;
             for(i = 0; i < 5; i++)
             {
                temp[cnt++] = num_to_char(drbuf[i]/16);
                temp[cnt++] = num_to_char(drbuf[i]%16);
             }
             temp[10] = '\0';
             for(i = 0; i < ct; i++)          //遍历刷出来的所有卡，出现卡号相同的在原基础上累加计数
             {
                if(same(temp, card[i].ID))
                {
                   card[i].num++;
                   sprintf(prtnum, "%d", card[i].num);
                   LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, i,
prtnum);
                    flg = 1;
                }
             }
             if(!flg)                                  //遍历所有卡没有出现相同卡号的就新增一行
             {
```

```
                strcpy(card[ct].ID, temp);
                card[ct].num++;
                sprintf(prtnum, "%d", ct+1);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 0, ct,
prtnum);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 1, ct,
card[ct].ID);
                sprintf(prtnum, "%d", card[ct].num);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, ct,
prtnum);
                ct++;
            }
            rxflag = 0;
         }
```

计数清零在回调函数中实现，清空所有计数器，所有卡号显示的次数均显示为 0。

```
      case GUI_ID_BUTTON0:
            switch(NCode)
            {
            case WM_NOTIFICATION_CLICKED:
               for(i = 0; i < ct; i++)
               {
                  if(card[i].num) card[i].num = 0;
                  LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, i,
"0");
               }
               break;
            case WM_NOTIFICATION_RELEASED:
               break;
            case WM_NOTIFICATION_MOVED_OUT:
               break;
            }
            break;
```

卡号清除在回调函数中的另一个按键消息中完成，先把屏幕上显示卡号和数字的位置更新为空，再把卡号计数器 ct 清零。

```
   case GUI_ID_BUTTON1:
            switch(NCode)
            {
            case WM_NOTIFICATION_CLICKED:
               for(i = 0; i < ct; i++)
               {
                  LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 0, i,
"");
```

```
                    LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 1, i,
"");
                    LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, i,
"");
                    card[i].num = 0;
                }
                ct = 0;
                break;
            case WM_NOTIFICATION_RELEASED:
                break;
            case WM_NOTIFICATION_MOVED_OUT:
                break;
            }
            break;
```

界面初始化中各种初始化的 API 函数用户自行查阅《emWin5 中文手册》中相关控件的部分就可以了，这里着重强调一下列表控件表头格式的更改，实验中可能让表头显示中文并更改大小等一系列操作，初学者往往不能成功，这是因为他们仍然是对 LISTVIEW 这个控件在进行操作。实际上，列表控件的表头是一个单独的控件 HEADER，用户需要用 LISTVIEW 中的 API 函数获取表头控件 HEADER 的句柄，再用 HEADER 控件的 API 函数进行想要的修改操作。下面举一个实验中设置表头字体的例子：

```
hHeader = LISTVIEW_GetHeader(WM_GetDialogItem(pMsg->hWin, GUI_ID_LISTVIEW0));//获取句柄
HEADER_SetFont(hHeader, &GUI_FontFONT3);//用 HEADER 句柄设置字体
```

再补充说明一下 LISTVIEW 控件中滚动条，尽管 LISTVIEW 的 API 中有自动添加滚动条，但是一般还是手动把滚动条控件加到列表控件上去，原因在于手动添加可以获取滚动条句柄，用户可以对这个滚动条进行各种修改操作，自动添加的滚动条由于没有句柄，不合适的地方我们没有办法进行修改。这一思想在 emWin 的界面设置中非常关键，想要进行任何修改，第一步就是想办法获取相应的句柄，没有句柄没办法进行任何操作，软件只有通过句柄才能知道用户的“想法”，定位到要修改的位置。

```
    //LISTVIEW_SetAutoScrollV(hItem, 1);
    hScroll = SCROLLBAR_CreateAttached(hItem, GUI_ID_VSCROLL);
    SCROLLBAR_SetWidth(hScroll, 15);
```

上面代码中第一句是自动为 LISTVIEW 添加滚动条，我们把它注释掉了并没有使用。下面一句手动添加滚动条，相应的 API 函数会返回这个滚动条的句柄，把这个句柄保存下来，在最后就可以进行对滚动条宽度的修改了。

第三章 高频 HF-13.56M（ISO14443） 实验及实例

3.1 ISO14443 相关实验

1、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： 13.56M ISO14443 模块测试软件

2、实验内容

- 了解 14443 的基本概念、 国际标准、 协议内容
- 了解 14443 的标准接口
- 了解 14443 的应用范围及领域
- 了解 14443 指令操作
- 掌握 14443 的相关的命令的操作

3、实验原理

3.1 ISO14443 协议

ISO14443 协议是 Contactless card standards（ 非接触式 IC 卡标准） 协议。 英文版原版由 4 个部分组成：

物理特性、 频谱功率和信号接口、 初始化和防冲突算法、 通讯协议。

3.2 物理特性

1) 范围

ISO/IEC14443 的这一部分规定了邻近卡（ PICC） 的物理特性。 它应用于在耦合设备附近操作的 ID － 1 型识别卡。

ISO/IEC14443 的这一部分应与正在制定的 ISO/IEC14443 后续部分关联使用。

2) 标准引用

下列标准中所包含的条文， 通过在本标准中引用而构成为本标准的条文。 本标准出版时， 所示版本均为有效。 所有标准都会被修订， 使用 ISO/IEC14443 这一部分的各方应探讨使用下列最新版本标准的可能性。

ISO 和 IEC 的成员修订当前有效国际标准的纪录。

ISO/IEC7810： 1995， 识别卡——物理特性。

ISO/IEC10373， 识别卡——测试方法。

1) 定义缩略语和符号

下列定义适用于 ISO/IEC14443 的这一部分：

- 集成电路 Integrated circuit(s)（ IC）： 用于执行处理和/或存储功能的电子器件。
- 无触点的 Contactless： 完成与卡的信号交换和给卡提供能量， 而无需使用微电元件（即： 从外部接口设备到卡上的集成电路之间没有直接路径）。
- 无触点集成电路卡 Contactless integrated circuit(s) card： 一种 ID－1 型卡类型（如 ISO/IEC7810 中所规定）， 在它上面有集成电路， 并且与集成电路的通信是用无触点的方式完成的。
- 邻近卡 Proximity card（PICC）： 一种 ID－1 型卡， 在它上面有集成电路和耦合工具， 并且与集成电路的通信是通过与邻近耦合设备电感耦合完成的。
- 邻近耦合设备 Proximity coupling device（PCD）： 用电感耦合给邻近卡提供能量并控制与邻近卡的数据交换的读/写设备。

2) 物理特性

- 一般特性

邻近卡应有根据 ISO/IEC7810 中规定的 ID－1 型卡的规格的物理特性。

- 尺寸

邻近卡的额定尺寸应是 ISO/IEC7810 中规定的 ID－1 型卡的尺寸。

- 附加特性

1、 紫外线

ISO/IEC14443 的这一部分排除了大于海平面普通日光中的紫外线的紫外线水平的防护需求， 超过周围紫外线水平的防护应是卡制造商的责任。

2、 X－射线

卡的任何一面曝光 0.1Gy 剂量， 相当于 100KV 的中等能量 X—射线（每年的累积剂量）， 应不引起卡的失效。

注 1： 这相当于人暴露其中能接受的最大值的年累积剂量的近似两倍。

3、 动态弯曲应力

按 ISO/IEC10373 中描述的测试方法（短边和长边的最大偏移为 hwA=20mm， hwB=10mm） 测试后，邻近卡应能继续正常工作。

4、 动态扭曲应力

按 ISO/IEC10373 中描述的测试方法（旋转角度为 15° ） 测试后， 邻近卡应能继续正常工作。

5、 可变磁场

在下表给出的平均值的磁场内暴露后， 邻近卡应能继续正常工作。

频率范围（MHz）	平均磁场强度（A/m）	平均时间（minutes）
0.3——3.0	1.63	6
3.0——30	4.98/f	6
30——300	0.163	6

注：f—频率（MHz）

表 3.1.1 邻近卡受磁场影响条件

磁场的最高值被限制在平均值的 30 倍。

在 12A/m、 13.56MHz 的磁场中暴露后， 邻近卡应能继续正常工作。

6、 可变电场

在下表给出的平均值的电场内暴露后， 邻近卡应能继续正常工作。

7、 静态电流

按 ISO/IEC10373（ IEC1000－4－ 2： 1995） 中描述的测试方法（ 测试电压为 6KV） 测试后， 邻近卡应能继续正常工作。

8、 静态磁场

在 640KA/m 的静态磁场内暴露后， 邻近卡应能继续正常工作。

警告： 磁条上的数据内容将被这样的磁场擦去。

9、 工作温度

在 0℃ 到 50℃ 的环境温度范围内， 邻近卡应能正常工作。

频率范围（MHz）	平均电场强度（V/m）	平均时间（minutes）
0.3——3.0	0.614	6
3.0——30	1842/f	6
30——300	61.4	6

注：f—频率（MHz）

表 3.1.2 邻近卡受电场影响条件

电场的最高值被限制在平均值的 30 倍。

3.3 频谱功率和信号接口

1) 范围

ISO/IEC14443 的这一部分规定了需要供给能量的场的性质与特征， 以及邻近耦合设备（ PCDs） 和邻近卡（ PICCs） 之间的双向通信。

ISO/IEC14443 的这一部分应与 ISO/IEC14443 的其它部分关联使用。

ISO/IEC14443 的这一部分并不规定产生耦合场的方法， 也没有规定如何符合因国家而异的电磁场辐射和人体辐射安全的条例。

2) 标准引用

下列标准中所包含的条文， 通过在本标准中引用而构成为本标准的条文。 本标准出版时， 所示版本均为有效。

所有标准都会被修订， 使用 ISO/IEC14443 这一部分的各方应探讨使用下列标准最新版本的可能性。ISO 和 IEC 的成员修订当前有效国际标准的纪录。

ISO/IEC14443－1： 识别卡——无触点集成电路卡——邻近卡——第一部分： 物理特性。

ISO/IEC10373， 识别卡——测试方法。

3) 缩略语和符号

缩略语	符号含义
ASK	移幅键控
BPSK	二进制移相键控
NRZ－L	不归零，（L 为电平）
PCD	邻近耦合设备
PICC	邻近卡
RF	射频
fc	工作场的频率（载波频率）
fs	副载波调制频率
Tb	位持续时间

表 3.1.3 缩略语和符号

4) 术语和定义

ISO/IEC14443－2 中给出的定义和下列定义适用于本国际标准：

- 位持续时间 Bit duration

  一个确定的逻辑状态的持续时间， 在这段时间的最后， 一个新的状态位将开始。

- 二进制相移键控 Binary phase shift keying

  相移键控， 此处相移 180° ， 从而导致两个可能的相位状态。

- 调制系数 Modulation index

  定义为（a－b)/(a+b）， 其中 a， b 分别是信号幅度的最大， 最小值。

- 不归零 NRZ－L

  在位持续时间内，一个逻辑状态的位编码方式，它以在通信媒介中的两个确定的物理状态之一来表示。

- 副载波 Subcarrier

  以载波频率 fc 调制频率 fs 而产生的 RF 信号。

5) 邻近卡的初始化对话

邻近耦合设备和邻近卡之间的初始化对话通过下列连续操作进行：

—PCD 的射频工作场激活 PICC

—邻近卡静待来自邻近耦合设备的命令

—邻近耦合设备命令的传送

—邻近卡响应的传送

这些操作使用下面段落中规定的射频功率和信号接口。

6) 功率传输

邻近耦合设备产生一个被调制用来通信的射频场， 它能通过耦合给邻近卡传送功率。

- 频率

射频工作场频率（ fc） 是 13.56MHz± 7kHz。

- 工作场

最小未调制工作场的值是 1.5A/mrms， 以 Hmin 表示。

最大未调制工作场的值是 7.5A/mrms， 以 Hmax 表示。

邻近卡应持续工作在 Hmin 和 Hmax 之间。

从制造商特定的角度说（ 工作容限）， 邻近耦合设备应产生一个大于 Hmin， 但不超过 Hmax 的场。

另外， 从制造商特定的角度说（ 工作容限）， 邻近耦合设备应能将功率提供给任意的邻近卡。

在任何可能的邻近卡的状态下，邻近耦合设备不能产生高于在 ISO/IEC14443－1 中规定的交变电磁场。

7) 信道接口

耦合 IC 卡的能量是通过发送频率为 13.56MHz 的阅读器的交变磁场来提供。由阅读器产生的磁场必须在 1.5A/m~7.5A/m 之间。 国际标准 ISO14443 规定了两种阅读器和近耦合 IC 卡之间的数据传输方式：A 型和 B 型。

一张 IC 卡只需选择两种方法之一。 符合标准的阅读器必须同时支持这两种传输方式， 以便支持所有的 IC 卡。 阅读器在” 闲置“的状态时能在两种通信方法之间周期的转换。

PCD->PICC	A 型	B 型
调制	ASK 100%	ASK 10%（键空度 8%～12%）
位编码	改进的 Miller 编码	NRZ 编码
同步	位级同步（帧起始，帧结束标记）	每个字节有一个起始位和一个结束位
波特率	106KdB	106kdB

表 3.1.4 阅读器(PCD） 到卡（ PICC） 的数据传输

PICC->PCD	A 型	B 型
调制	用振幅键控调制 847kHz 的负载调制的负载波	用相位键控调制 847kHz 的负载
位编码	Manchester 编码 NRZ 编码	NRZ 编码
同步	1 位"帧同步"（帧起始，帧结束标记）	每个字节有一个起始位和一个结束位
波特率	106KdB	106kdB

表 3.1.5 卡（ PICC） 到阅读器（ PCD） 的数据传输

3.4 初始化和防碰撞算法

1) 范围

ISO/IEC14443 的这一部分规定了邻近卡（PICCs） 进入邻近耦合设备（PCDs） 时的轮寻， 通信初始化阶段的字符格式， 帧结构， 时序信息。 REQ 和 ATQ 命令内容， 从多卡中选取其中的一张的方法，初始化阶段的其它必须的参数。

这部分规定同时适用于 A 型 PICCs 和 B 型 PICCs。

2) 标准引用

下列标准中所包含的条文， 通过在本标准中引用而构成为本标准的条文。 本标准出版时， 所示版本均为有效。

所有标准都会被修订， 使用 ISO/IEC14443 这一部分的各方应探讨使用下列标准最新版本的可能性。ISO 和 IEC 的成员修订当前有效国际标准的纪录。

ISO/IEC 3309:1993 信息技术系统间的远程通信和信息交换数据链路层的控制帧结构。

ISO/IEC 7816-3： 1997 识别卡接触式集成电路卡第三部分电信号和传输协议。

ISO/IEC 14443-2 识别卡非接触式集成电路卡第二部分频谱功率和信号接口

ITU-T 推荐 V.41

3) 术语和定义

ISO/IEC14443－3 中给出的定义和下列定义适用于本国际标准：

- 防碰撞循环（Anticollision loop)

  在多个 PICCs 中， 选出需要对话的卡的算法

- 可适用的（Applicative)

  属于应用层或更高层的协议， 将在 ISO/IEC 1443-4 中描述。

- 位碰撞检测协议（Bit collision detetion protocol)

  帧内的位检测防碰撞算法。

- 数据块(Block)一系列的数据字节构成数据块。

- 异步数据块传输（Block-asynchronous transmission)

  在异步数据块传输， 数据块是包括帧头和帧尾的数据帧。

- 字节（Byte)

  八个 bits 构成一个字节。

- 字符串

  在异步通信中，一个字符串包括一个开始位，8 位的信息，可选的寄偶检验位，结束位和时间警戒位。

- 碰撞

  两个 PICCs 和同一个 PCD 通信时， PCD 不能区分数据是属于那一个 PICC。

- 能量单位

  在 ISO/IEC14443 的这个部分中， 1etu=128/fc 容差为 1%

- 时间槽协议

PCD 建立与一个或多个 PICCs 通信的逻辑通道， 它利用时间槽处理 PICC 的响应， 与时间槽的 ALOHA 相似。

4) 缩略语和符号

ATQ	对请求的应答
ATQA	对 A 型卡请求的应答
ATQB	对 B 型卡请求的应答
ATR	对重新启动的请求的应答
ATS	对选择请求的应答
ATQ-ID	对 ID 号请求的应答
CRC	环检验码
RATS	对选择应答请求
REQA	对 A 型卡的请求
REQB	对 B 型卡的请求
REQ-ID	请求 ID 号
RESEL	重新选择的请求

表 3.1.6 缩略语和符号

5) 轮询

为了检测到是否有 PICCs 进入到 PCD 的有效作用区域， PCD 重复的发出请求信号， 并判断是否有响应。 请求信号必须是 REQA 和 REQB， 附加 ISO/IEC14443 其它部分的描述的代码。 A 型卡和 B 型卡的命令和响应不能够相互干扰。

6) A 型卡的初始化和防碰撞

当一个 A 型卡到达了阅读器的作用范围内， 并且有足够的供应电能， 卡就开始执行一些预置的程序后，IC 卡进入闲置状态。 处于“闲置状态”的 IC 卡不能对阅读器传输给其它 IC 卡的数据起响应。 IC 卡在“闲置状态”接收到有效的 REQA 命令， 则回送对请求的应答字 ATQA。 当 IC 卡对 REQA 命令作了应答后， IC 卡处于 READY 状态。 阅读器识别出： 在作用范围内至少有一张 IC 卡存在。 通过发送 SELECT 命令启动“二进制检索树”防碰撞算法， 选出一张 IC 卡， 对其进行操作。

- PICC 的状态集
  - 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。
  - 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。
  - 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。
  - 激活状态： PCD 通过防碰撞已经选出了单一的卡。
  - 结束状态
- 命令集

PCD 用于管理与 PICC 之间通信的命令如表 3.1.7 所示：

REQA	对 A 型卡的请求
WAKE-UP	唤醒
ANTICOLLISION	防碰撞
SELECT	选择
HALT	结束

表 3.1.7 A 卡指令集

7) B 型卡的初始化和防碰撞

当一个 B 型卡被置入阅读器的作用范围内， IC 卡执行一些预置程序后进入“闲置状态” ， 等待接收有效的 REQB 命令。 对于 B 型卡， 通过发送 REQB 命令， 可以直接启动 Slotted ALOHA 防碰撞算法， 选出一张卡， 对其进行操作。

- PICC 状态集
  - 调电状态： 由于载波能量低， PICC 没有工作。
  - 闲置状态： 在这个状态， PICC 已经上电， 监听数据帧， 并且能够识别 REQB 信息。

当接收到有效的 REQB 帧的命令， PICC 定义了单一的时间槽用来发送 ATQB。 如果是 PICC 定义的第一个时间槽， PICC 必须发送 ATQB 的响应信号， 然后进入准备—已声明子状态。 如果不是 PICC 定义的第一个时间槽， PICC 进入准备—已请求子状态。

- 准备—已请求子状态：在本状态下，PICC 已经上电，并且已经定义单一的时间槽来发送 ATQB。它监听 REQB 和 Slot-MARKER 数据帧。
- 准备—已声明子状态： 在本状态下， PICC 已经上电， 并且已经发送了对 REQB 的 ATQB 响应。它监听 REQB 和 ATTRIB 的数据帧。
- 激活状态： PICC 已经上电， 并且通过 ATTRIB 命令的前缀分配到了通道号， 进入到应用模式。它监听应用信息。
- 停止状态： PICC 工作完毕， 将不发送调制信号， 不参加防碰撞循环。
- 命令集：

  管理多极点的通信通道的 4 个基本命令

  REQB 对 B 型卡的请求

  Slot-MARKER 时间槽产生命令

  ATTRIB PICC 选择命令的前缀

  DESELECT 去选择

3.5 传输协议

1) 范围

ISO/IEC14443 的这一部分规定了非接触的半双工的块传输协议并定义了激活和停止协议的步骤。 这部分传输协议同时适用于 A 型卡和 B 型卡。

2) 标准引用

下列标准中所包含的条文， 通过在本标准中引用而构成本标准的条文。 本标准出版时， 所示版本均为有效。 所有标准都会被修订， 使用 ISO/IEC14443 这一部分的各方应探讨使用下列标准最新版本的可能性。ISO 和 IEC 的成员修订当前有效国际标准的纪录。

ISO/IEC 7816-4： 识别卡 接触式集成电路卡 第四部分 产业内部交换命令

3) 术语和定义

- 数据块（ Block)

特殊格式的数据帧。 符合协议的数据格式， 包括 I-blocks,R-blocks 和 S-blocks.

- 帧格式（ frame format)

ISO/IEC 1444303 定义的。 A 型 PICC 使用 A 类数据帧格式， B 型 PICC 使用 B 类数据帧格式。

4) A 型 PICC 的协议激活

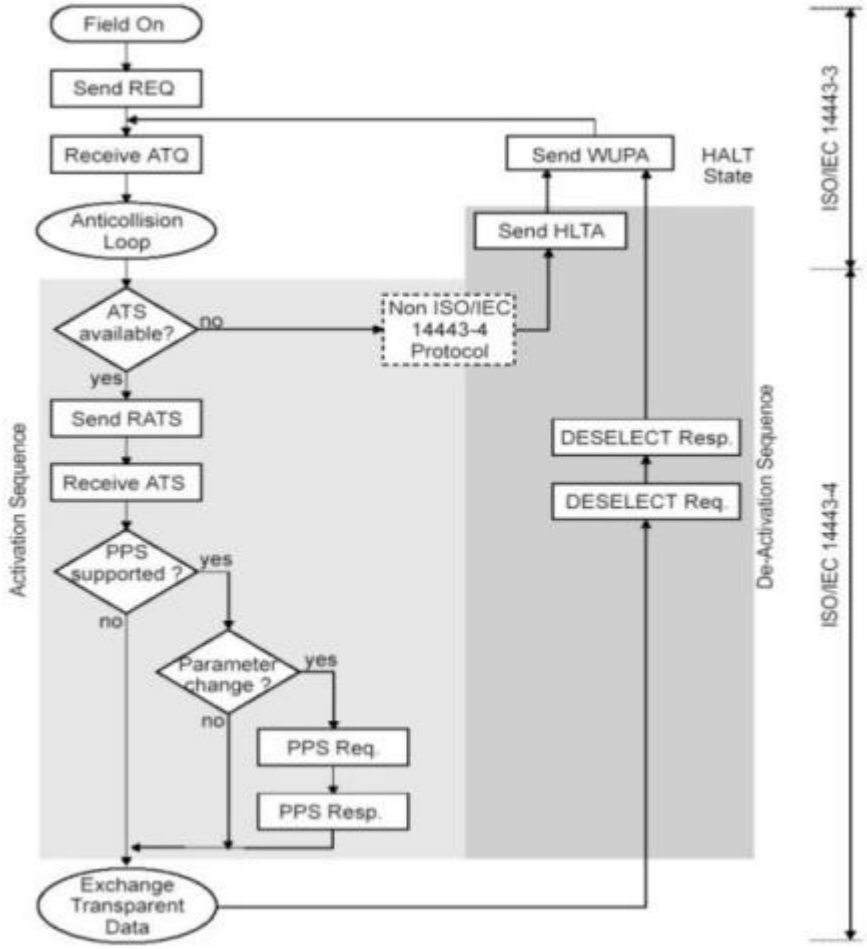

Figure 1 — Activation overview of a PICC Type A

图 3.1.1 A 型卡的协议激活流程

5) 缩略语和符号

PPS	协议和参数的选择
R-block	接收准备块
R(ACK)	R-block 包含正的确认
R(NAK)	R-block 包含负的确认
RFU	保留，将来使用
S-block	管理块
SAK	选择确认
WUPA	A 型卡的唤醒命令
WTX	等待时间扩展

表 3.1.8 缩略语和符号

6) B 型 PICC 的协议激活

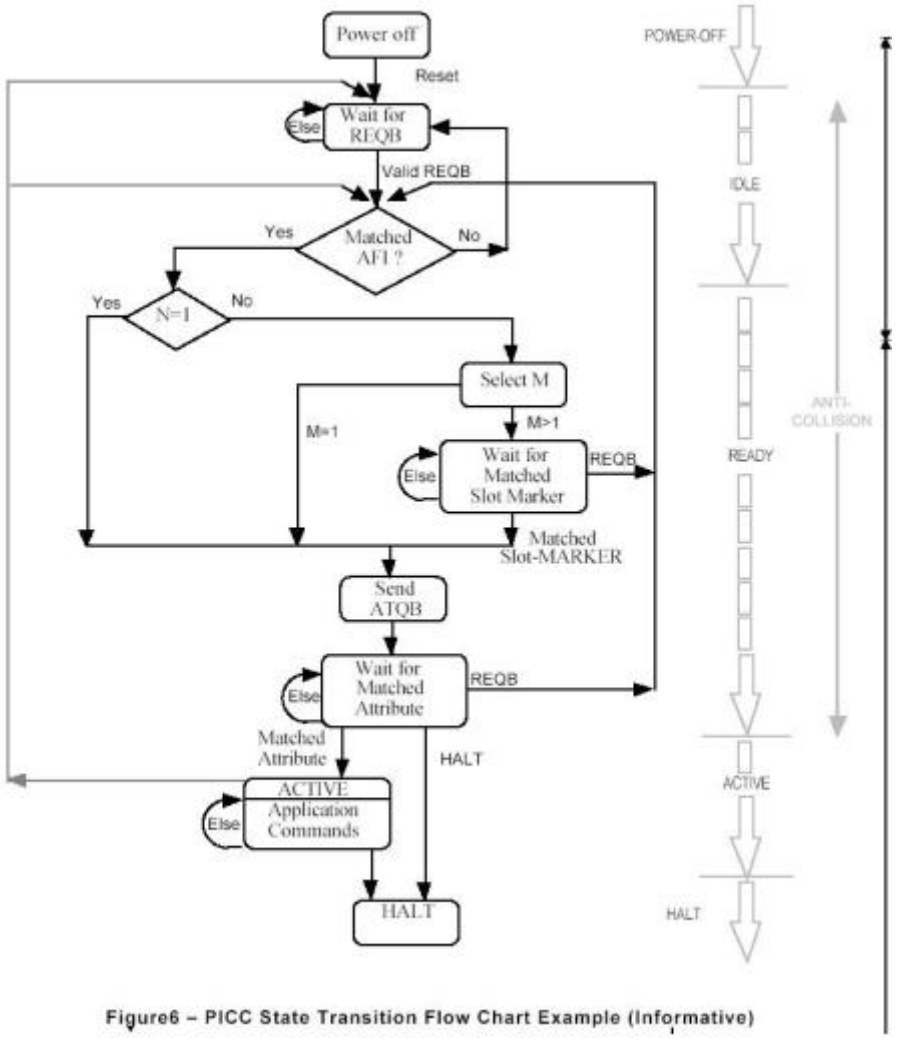

Figure6 – PICC State Transition Flow Chart Example (Informative)

图 3.1.2 B 型卡的协议激活流程

7) 半双工的传输协议

8) A 型和 B 型 PICC 的协议去激活

4、实验步骤

1) 准备好硬件， RFID 13.56MHz 模块(ISO14443 协议)和 14443 标签。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 功能测试

- 选中 14443 模块

点击功能按键中右下角的 14443，选择读卡器为 14443 模式，13.56M 读卡器是 ISO14443 和 ISO15693 共用的，因此做 14443 实验前必须做这一步，让读卡器知道要识别的是 14443 标签。模式选择成功会有 OK 的提示，失败则提示 failed。如图 3.1.3 选择成功的提示。

图 3.1.3 读卡器选择 14443 模式成功

- 寻所有卡操作

将 14443 标签置于 HF-13.56MHz 模块天线辐射区内，点击寻卡.ALL，此时发送寻卡命令， 14443 模块识别标签并将卡号反馈回来，显示在屏幕左上角的卡号处。

图 3.1.4 寻卡成功

- 寻未休眠卡

将 14443 标签置于 HF-13.56MHz（ISO14443 协议） 模块天线辐射区内， 点击寻卡.IDL。

这里需要后面的休眠和唤醒命令结合使用， 验证这些命令的功能。如果寻卡成功，和上图 3.1.4 显示效果相同。

- 休眠操作

将 14443 标签置于 HF-13.56M（ ISO14443A 协议） 模块天线辐射区内， 点击休眠， 成功返回 OK，失败返回 failed。休眠的卡再点击寻卡.IDL 就不能读取到卡号了，不过此时用寻所有卡的寻卡.ALL 依然可以读取到卡号，不过这并不能接触标签的休眠状态，用户可以再自行尝试点击寻卡.IDL，会发现依然寻不到卡号。点击休眠按键右侧的唤醒可以接触标签的休眠状态，再点寻卡.IDL 就可以读取到卡号了。

- 内存读写

内存读写是对标签操作， 用于读写标签存储器中的不同地址的数据信息， 在进行内存读写之前需要先寻卡， 在卡号区域选择一张卡进行内存读写。 点击读写操作窗口中的内存读写按钮。

根据 14443 标签的相关资料，标签存储器分为 16 个扇区，每个扇区分为 4 个块，每个块有 16 个字节， 其中第 0 扇区的第 0 块为卡序列号， 为只读的， 不可写入。 数据块 1 和数据块 2 为用户数据， 您可以随意写入和读取， 数据块 3 为密码 A（6 字节）、 存取控制码（4 字节）、 密码 B（6 字节） 的信息， 最好别随意修改， 否则可能使该扇区不能用（比如您随意写了个密码， 后面操作需要验证密码才能正常操作， 这个时候当你忘记密码， 该扇区就不能读写信息了）。

接着是扇区 1 到扇区 15， 这些扇区数据块 3 和扇区 0 的数据块 3 一样， 是密码信息， 数据块 0 和扇区 0 的数据块 2、 3 一样， 您可以随意读写内容。实验程序提供了 0-4 扇区和 0-3 块供用户选择。

上文提到数据块 3 存放的是密码数据，因此不要随意修改，这里额外补充一些内容：读取密钥 A 会发现其一直显示为 0， 这是因为密钥 A 是不可见的， 所以它表现为 0， 实际值与您的操作相关，要注意到不可见和不可修改的区别，密钥 A 的数据是可以修改的，而在修改后仍然显示为 0。如果用户擅自修改了密钥 A 而又忘记了修改内容，这个扇区有关密钥 A 的所有操作就都无法使用了，密码对各个扇区是独立的，密钥 A 和密钥 B 的初始数值均为 FFFFFFFFFFFF。

操作窗口 1 是写标签界面，如图 3.1.5 选中的是扇区 4 块 2，向块 2 写入 8888，操作窗口 1 是只写的，读卡按键不可点击。

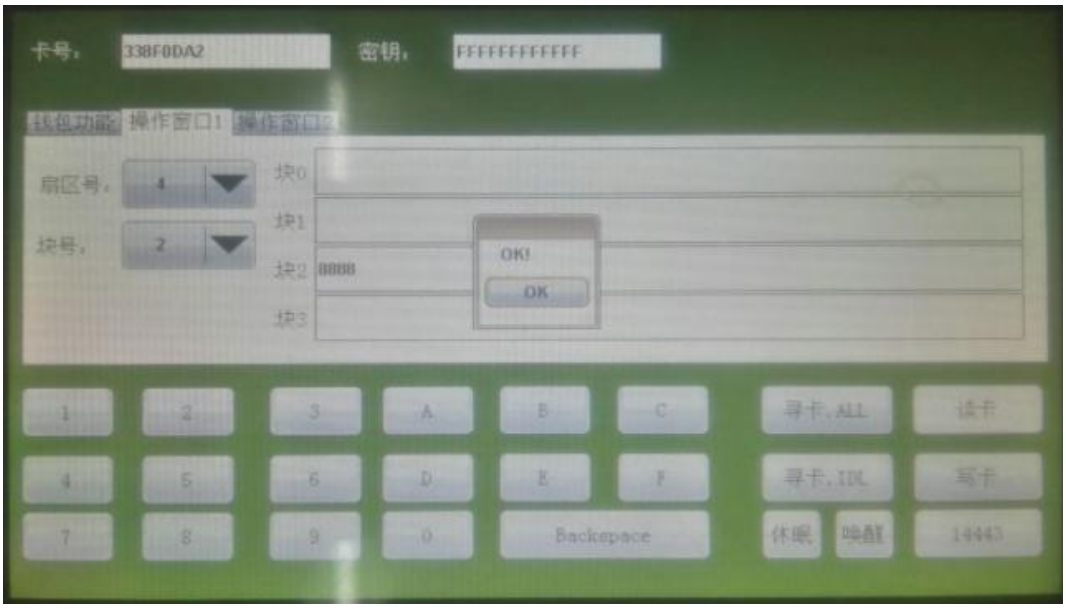

图 3.1.5 写入标签

操作窗口 2 是读标签界面，如图 3.1.6 选中我们刚刚写入的区域，点击读卡，可以看到前四位是写入的 8888，后面没有填写的内容均补 0，操作窗口 2 是只读的，写卡按键不可点击。

图 3.1.6 读取标签

- 钱包功能

点击多页控件上的钱包功能操作页面，先选择一个可以随意读写的区域，我们这里仍然选择扇区 4 块 2 操作，实际上进行的仍然是是内存读写，支持初始化、充值、扣款、读余额，必须正确输入密钥，操作金额在十进制下的编辑框输入，右边会即时把十进制数转化成十六进制数，之所以这么设计，是因为我们在内存中操作的都是十六进制数，增加这么一步转化工作，用户在对照源码进行学习的时候可以更加直观，便于理解。

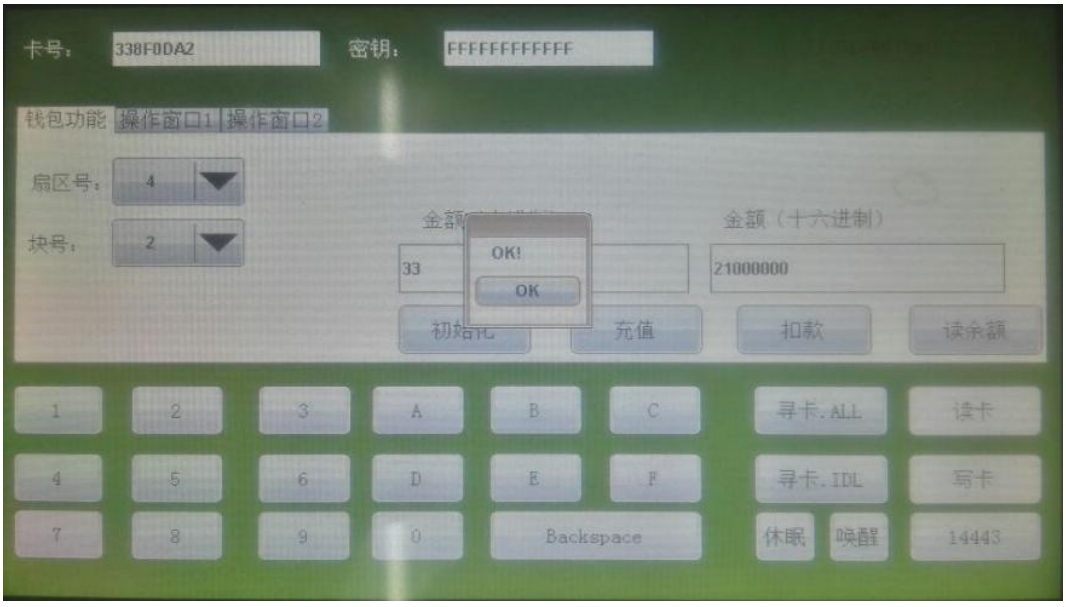

图 3.1.7 钱包功能演示

3.2 高频 HF-13.56M 14443A 开发实例

1、实验目的

- 掌握 HF 13.56MHz ISO-14443A 串口通信协议
- 了解 HF 13.56MHz RFID 模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用
- 了解 13.56MHz RFID 的应用范围及领域

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、 实验内容

- 14443 标签寻卡
- 14443 标签内存区域读写数据
- 基于 14443 标签开发的钱包功能实例

4、实验原理

4.1 13.56MHz RFID 应用领域

1、 图书档案管理系统的应用

2、 瓦斯钢瓶的管理应用

3、 服装生产线和物流系统管理和应用

4、 三表预收费系统

5、 酒店门锁的管理和应用

6、 大型会议人员通道系统

7、 物流与供应链管理解决方案

8、 医药物流与供应链管理

9、 智能货架的管理

10、 校园一卡通

11、 电子钱包（对标签有要求）

4.2 13.56MHz RFID 符合的国际标准

a) ISO/IEC 14443 近耦合 IC 卡， 最大的读取距离为 10cm。

b) ISO/IEC 15693 疏耦合 IC 卡， 最大的读取距离为 1M。

c) ISO/IEC 18000-3 该标准定义了 13.56MHz 系统的物理层， 防冲撞算法和通讯协议。

d) 13.56MHz ISM Band Class 1 定义 13.56MHz 符合 EPC 的接口定义。

4.3 M1 卡性能指标

a) MIFARE S50 卡的容量为 1K 字节的 EEPROM， 分为 16 个扇区， 每个扇区为 4 块， 以块为存取单位，每块 16 个字节。

b) MIFARE S70 卡的容量为 4K 字节的 EEPROM，前 2K 共分 32 个扇区，每个扇区 4 块，后 2K 共分为 8 个扇区， 每个扇区 16 块， 以块为存取单位， 每块 16 个字节。

c) 每个扇区有独立的密码和访问控制机制。

d) 每张卡有唯一的序列号， 4 字节。

5、 实验步骤

❖ 开发工具选择

我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

❖ 创建工程

直接使用光盘中的 STemWin 模板工程。

❖ UI 设计

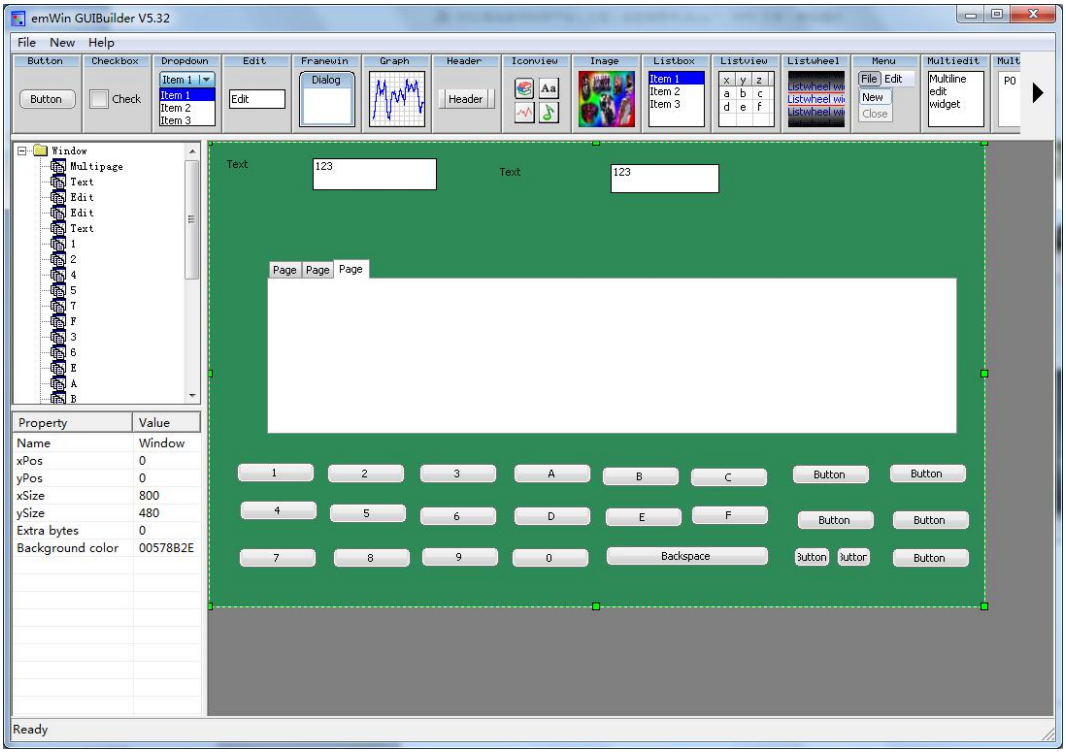

图 3.2.1 GUIBuilder 中完成粗略设计

“多页”控件中需要添加的控件只能从程序代码入手，直接在模拟器上加是不可以的，因为直接加在

模拟器上的控件是以最开始创建的 Window 控件为背景的，而此处我们的目的是让这些控件以“多页”控件中不同的页为背景显示不同控件，这需要在不同的窗口管理器中添加相应代码，后续会有代码实例的展示。

❖ 编码

接收数据采用在线解析，即中断边接收边解析，与之相对的离线解析是接收完完整的一帧数据后，再对这帧数据进行解析。两种方法各有利弊，本平台实验传送的数据规模较小，一律采用在线解析就可以满足要求。帧格式查看《13.56MHz_14443 串口通讯协议》，最后使用了异或校验，需要注意的是有哪些位参与了校验，一般来讲帧头、长度位、帧尾均不参与校验。

```
void BSP_ST16C554_CS1D_ISR(void)
{
   uint8_t ret, i, flag;
   if(GPIO_ReadInputDataBit(GPIOF, GPIO_Pin_9))
   {
      EXTI_ClearITPendingBit(EXTI_Line9);
      ret = BSP_ST16C554_CS1D_Read();

            if(FLAG)
            {
                FLAG = 0;
                return;
            }
      switch(recSta)
      {
      case RC632_STA_SOP1:
         len = 0;
         if(ret == 0xaa)
         {
            drbuf[len++] = ret;
            recSta = RC632_STA_SOP2;
         }
         break;
      case RC632_STA_SOP2:
         if(ret == 0xbb)
         {
            drbuf[len++] = ret;
            recSta = RC632_STA_LEN1;
         }
         else recSta = RC632_STA_SOP1;
         break;
      case RC632_STA_LEN1:
         dataLen = ret;
         drbuf[len++] = ret;
         recSta = RC632_STA_LEN2;
```

```
            break;
        case RC632_STA_LEN2:
            drbuf[len++] = ret;
            recSta = RC632_STA_DAT;
            break;
        case RC632_STA_DAT:
            drbuf[len++] = ret;
            if(drbuf[len-1] == 0xaa) FLAG = 1;
            if(len == dataLen + 3)
                recSta = RC632_STA_FCS;
            break;
        case RC632_STA_FCS:
            recSta = RC632_STA_SOP1;
            flag = drbuf[4];
            for(i = 5; i < 3 + dataLen; i++)
            {
                flag ^= drbuf[i];
            }
            dataLen = 0;
            if(flag == ret)
            {
                drbuf[len++] = ret;
                rxflag = 1;
            }
        }
    }
}
```

主函数的超级循环中主要完成一些需要即时更新的工作，GUI_Delay(1)是负责刷屏的函数，并且自带延时，括号中的数字表示延时的时长，我们在程序中填写的是 1，延时的时间没有具体单位，它的单位取决于用户进行触摸检测的时长，这个时长是用户自定义定时器设置的，如果想修改，可以去工程 App 文件夹下修改 main.c 文件中定时器初始化函数 BSP_Timer_Init()括号中的数值。我们在此处对这个延时的处理方式是填写最小的数 1，相当于仅让函数执行刷屏的功能，延时时间极短，具体情况想增加延时在循环中添加已经初始化好的精确延时函数 BSP_Delay_ms()即可。

一系列消息框执行函数，在用户进行操作后，会弹出消息框进行反馈，在此举一个例子：

```
        if(flag1 == 1)
        {
            GUI_ExecCreatedDialog(_CreateMessageBox("14443 OK!", "", 0,
&GUI_Font16B_ASCII));
            flag1 = 0;
        }
        else if(flag1 == 2)
        {
```

```
            GUI_ExecCreatedDialog(_CreateMessageBox("14443 failed!", "", 0,
&GUI_Font16B_ASCII));
            flag1 = 0;
        }
```

钱包功能中的进制转换，直接获取用户输入的 10 进制数，获取到的是字符串形式，再对字符串进行字符串解析转化成 16 进制，需要一提的是，此处如果用格式化输出函数 sprintf 可以很方便的得到 16 进制表示，但和我们手动写的转化函数表现形式略有不同，之所以手动写一个，是因为手动转化出的这种格式更加适合 14443A 的通信协议，转化结果可以直接用于通信，用户如果有疑惑可以自行查阅我们的通信协议或自己尝试写 14443A 数据接收发送的程序，很容易理解这个意思：

```
        //10 进制转 16 进制输出
        EDIT_GetText(WM_GetDialogItem(hWin1,ID_EDIT_2), buf, sizeof(buf));
        if(buf[0] != '\0')
        {
            sscanf(buf, "%d", &res);
            xianshijine = res;
            for(i = 0; i < 4; i++)
            {
                jine16[i*2+1] = tran(res%16);
                Data[i*2+1] = res % 16;
                res /= 16;
                jine16[i*2] = tran(res%16);
                Data[i*2] = res % 16;
                res /= 16;
            }
        }
        else
        {
            for(i = 0; i < 8; i++)
            {
                jine16[i] = 0;
                Data[i] = 0;
            }
            xianshijine = 0;
        }
        jine16[8] = '\0';
        EDIT_SetText(WM_GetDialogItem(hWin1,ID_EDIT_3),jine16);
        buf[0] = '\0';
```

获取密钥，密钥是我们设置的一个全局变量，在超级循环中不断读取该编辑框中的字符串进行更新：

```
        //获取密钥
        EDIT_GetText(WM_GetDialogItem(hWin,ID_EDIT_1), key, sizeof(key));
```

读卡和写卡的使能与失能依据条件是多页控件选择到哪一页，先用多页控件的 API 函数获取页面选择

信息，再针对信息更新：

```
                res = MULTIPAGE_GetSelection(WM_GetDialogItem(hWin, ID_MULTIPAGE_0));
               if(res == 0)
               {
                   WM_DisableWindow(WM_GetDialogItem(hWin,ID_BUTTON_duka));
                   WM_DisableWindow(WM_GetDialogItem(hWin,ID_BUTTON_xieka));
               }
               else if(res == 1)
               {
                   WM_DisableWindow(WM_GetDialogItem(hWin,ID_BUTTON_duka));
                   WM_EnableWindow(WM_GetDialogItem(hWin,ID_BUTTON_xieka));
               }
               else if(res == 2)
               {
                   WM_DisableWindow(WM_GetDialogItem(hWin,ID_BUTTON_xieka));
                   WM_EnableWindow(WM_GetDialogItem(hWin,ID_BUTTON_duka));
               }
```

输入信息用到了软键盘，下面讲解一下 emWin 软键盘功能是如果做出来的。

先简单解释下什么是计算机中的“焦点”，“焦点”可以理解为用户在计算机上关注的区域，放到 emWin 中可以有更加形象的说法，我们说哪个控件被选中、可以被操作，哪个控件就有焦点。例如按键，按键按下说明按键被选中，焦点在按键上；对于编辑框，只有焦点在编辑框上的时候，才可以往里面输入文本。emWin 的控件都是自带焦点的，这样就会出现一个问题，用户想点击按键向编辑框输入内容，可是在点击按键的时候，焦点会到按键上，编辑框没有被选中无法完成输入。解决的办法非常简单，emWin 提供了按键失焦（失能控件焦点）的 API 函数 BUTTON_SetFocussable()，这样选中编辑框再点按键，焦点就不会移走了。

下面是软键盘消息传递部分代码，先找到按键的编号 ID，确定哪些按键是软键盘的按键，直接获取按键上的字符传输。通过函数 GUI_SendKeyMsg()把获取的字符传递到当前焦点（编辑框）上。

```
          switch(NCode) {
          case WM_NOTIFICATION_CLICKED:
             Pressed = 1;
          case WM_NOTIFICATION_RELEASED:
             if ((Id >= GUI_ID_USER + 10 && Id <= GUI_ID_USER + 18) || (Id >= GUI_ID_USER +
0 && Id <= GUI_ID_USER + 7)) {
                int Key;
                if (Id != GUI_ID_USER + 7) {
                   char acBuffer[10];
                   BUTTON_GetText(pMsg->hWinSrc, acBuffer, sizeof(acBuffer)); /* Get the
text of the button */
                   Key = acBuffer[0];
                } else {
                   Key = GUI_KEY_BACKSPACE; /* Get the text from the array */
```

```
                }
                GUI_SendKeyMsg(Key, Pressed); /* Send a key message to the focussed window
*/
            }
            break;
        }
```

“多页”中的三个页面分别用三个窗口管理器配置，然后在主界面中通过“多页”的函数创建，这样用户选到哪个界面就会显示哪个。

主界面中创建三个“多页”界面，每页的句柄需要一个全局变量单独记录，便于对不同页控件进行操作：

```
    hWin1 = CreateWindowPage1();
    MULTIPAGE_SetFont(hWin, &GUI_FontFontSong);
    MULTIPAGE_AddEmptyPage(hWin, hWin1, "钱包功能");

    hWin2 = CreateWindowPage2();
    MULTIPAGE_AddEmptyPage(hWin, hWin2, "操作窗口 1");

    hWin3 = CreateWindowPage3();
    MULTIPAGE_AddEmptyPage(hWin, hWin3, "操作窗口 2");
```

第四章 高频 HF-13.56M（ISO15693） 实验及实例

4.1 ISO15693 相关实验

1、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： RFID 高级教学科研平台（Ⅱ型）模块测试软件

2、实验内容

- 了解 15693 的基本概念
- 了解 15693 的应用范围及领域
- 了解 15693 相关指令
- 掌握相关命令的使用

3、实验原理

ISO15693 协议

ISO15693 是针对射频识别应用的一个国际标准， 该标准定义了工作在 13.56Mhz 下智能标签和读写器的空气接口及数据通信规范， 符合此标准的标签最远识读距离达到 2 米。

工作频率： 工作频率范围为 13.56MHz±7KHz

工作磁场： 工作场最小值 0.15A/m,最大值 5A/m

调制：

VCD 到 VICC

支持两种幅值调制方式 100%ASK 和 10%ASK

VICC 到 VCD

采用副载波调制方式， 支持 FSK 的双副载波调试方式和 ASK 的单副载波两种调制方式。

数据编码及速率：

1、 VCD 到 VICC1

即读写器到标签的编码方式采用脉冲位置调制， 支持两种编码方式， 分别为 256 选 1 模式和 4 选 1 模式。 当为 256 选 1 模式时通信速率 1.54KBIT/S， 当为 4 选 1 模式时的通信速率为 26.48kbits/s。

2、 VICC 到 VCD

标签到读写器的数据编码采用曼彻斯特编码方式， 根据信号调试的方式不同， 通信速率也不同， 如下表 4.1 所示， 标签支持高速和低速两种通信速度：

数据速率	单副载波	双副载波
低	6.62kbits/s(fc/2048)	6.67kbits/s(fc/2032)
高	26.48kbits/s(fc/512)	26.69kbits/s(fc/508)

表 4.1.1 数据编码及速率

防冲突和传输协议

1、数据元素

1.1、 UID 唯一标识符

64 位唯一标识符， 在防冲突周期中， 读写器和标签间交互过程中用来识别标签的身份号码。

1.2、防冲突

读写器和标签之间通过标签的唯一标识符 UID 来进行防冲突循环处理， 采用的防冲突算法为 16 时隙方式和单时隙方式。

1.3、指令

指令	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

表 4.1.2 ISO15693 协议指令

1.4、AFI 应用标识

AFI 表示读写器锁定的应用类型， 仅选择符合应用类型的标签

1.5、DSFID 数据存储格式标识识

DSFID 指明了标签的数据存储格式

1.6、标签的状态

Power off 状态： 在标签未进入到有效磁场区域时标签处于 Power off 状态

Ready 状态： 被激活后选择表示未设立时， 处理任何请求

Quiet 状态： 不处理任何标签清点指令， 可接受直接寻址的命令

Select 状态： 仅响应选择标识符设置的请求

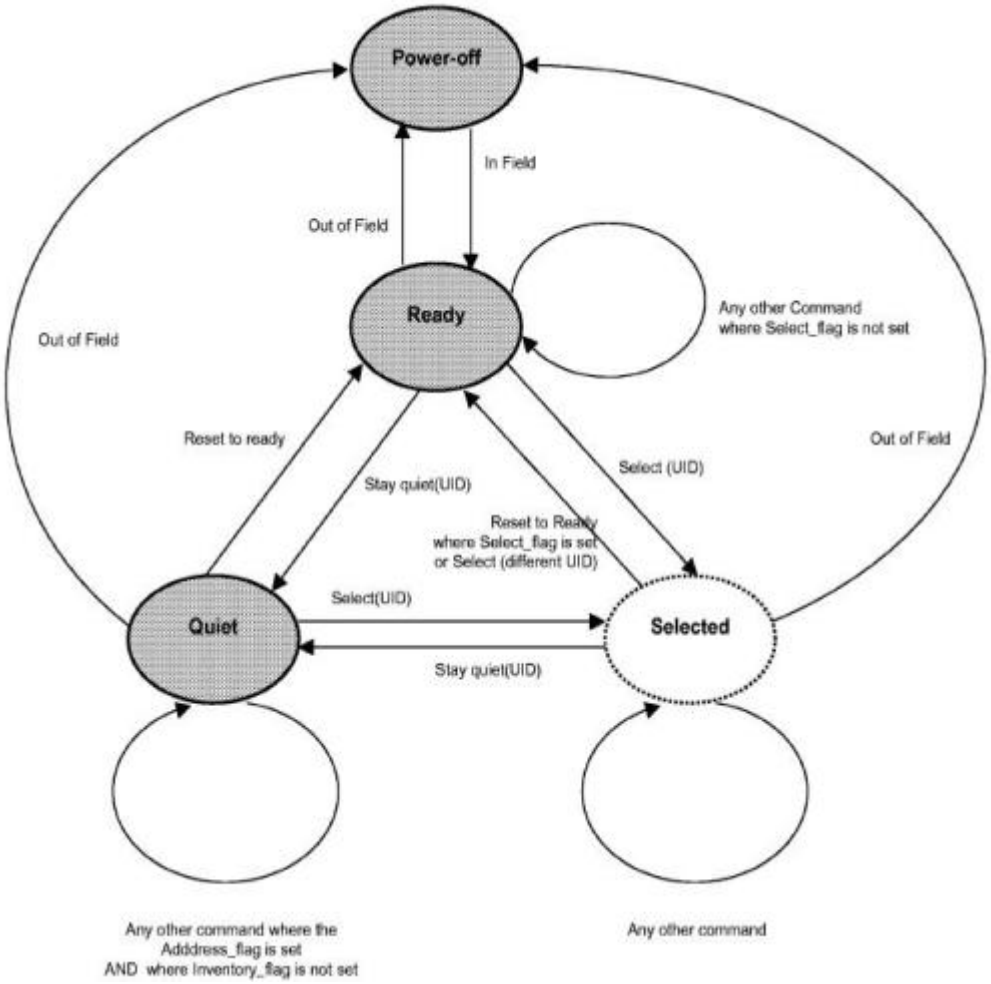

图 4.1.1 VICC 状态转换图

4、实验步骤

1) 准备好硬件， RFID 13.56MHz 模块(ISO15693 协议)和 15693 标签。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源

供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 功能测试

- 选中 15693 模块

点击按键 ISO15693，选择读卡器为 15693 模式，13.56M 读卡器是 ISO14443 和 ISO15693 共用的，因此做 15693 实验前必须做这一步，让读卡器知道要识别的是 15693 标签。模式选择成功会有 OK 的提示，失败则提示 failed。如图 4.1.2 选择成功的提示。

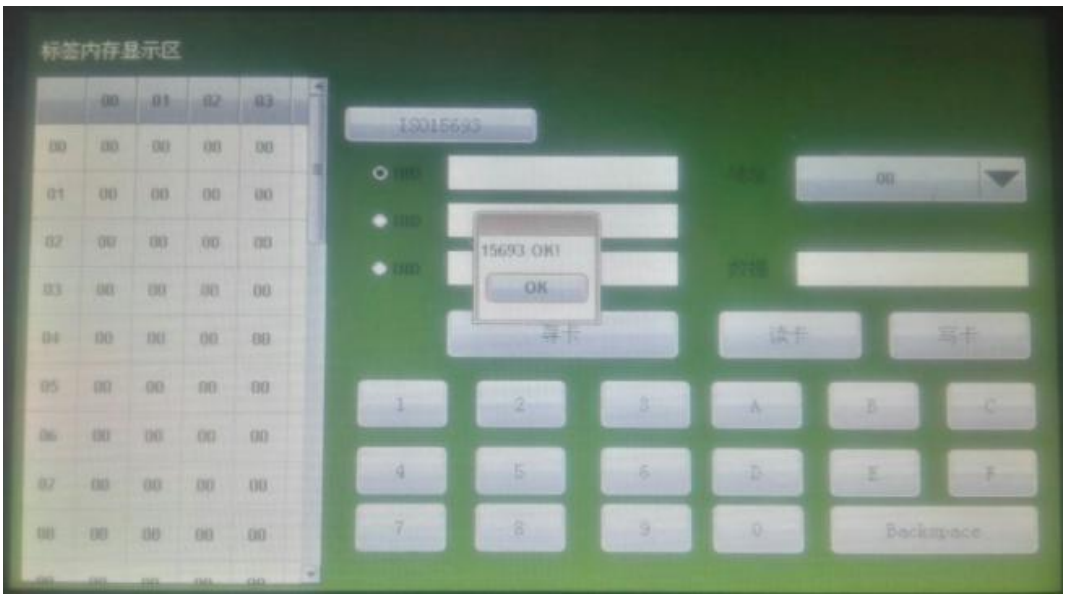

图 4.1.2 选中读卡器 15693 模式

- 寻卡

选中 15693 模式成功后才可以进行寻卡操作，与 14443A 协议不同的是，15693 协议支持同时寻多张卡显示多个标签号，如图 4.1.3，放置两张 15693 标签在 13.56M 读卡器上，可以同时寻到两张卡并把卡号显示出来。

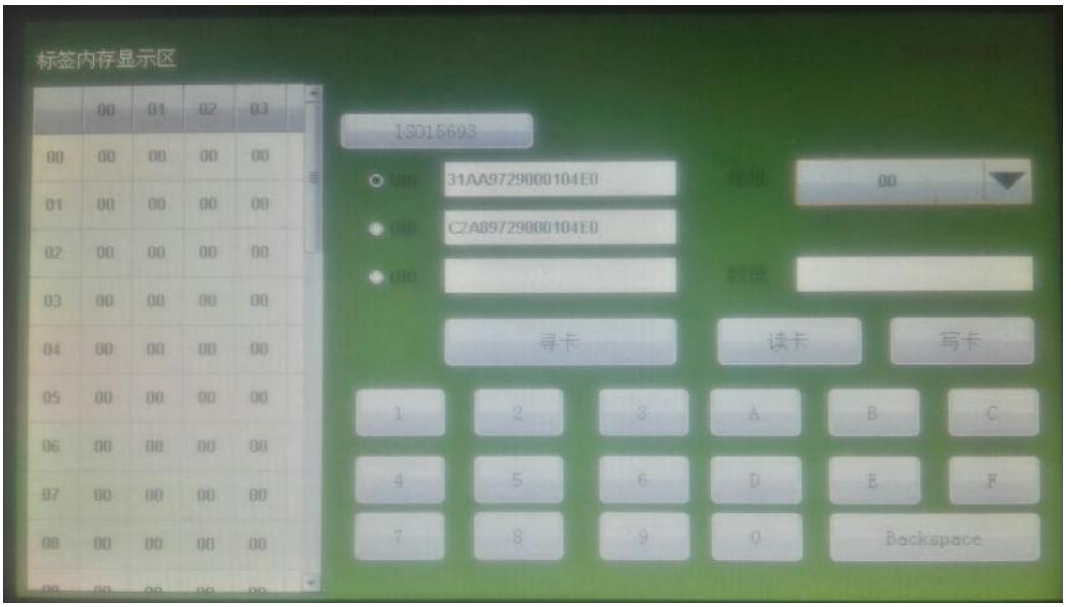

图 4.1.3 同时寻多张卡

- 读单个数据块

寻到多张卡后，用户可以通过点击不同的 UID 选择不同的卡进行内存操作。实验提供了 00-04 五个地址供用户体验，图 4.1.4 读选中卡的 03 地址，读出的数据显示在数据栏和标签内存显示区的自身位置，形象直观，易于新手掌握。

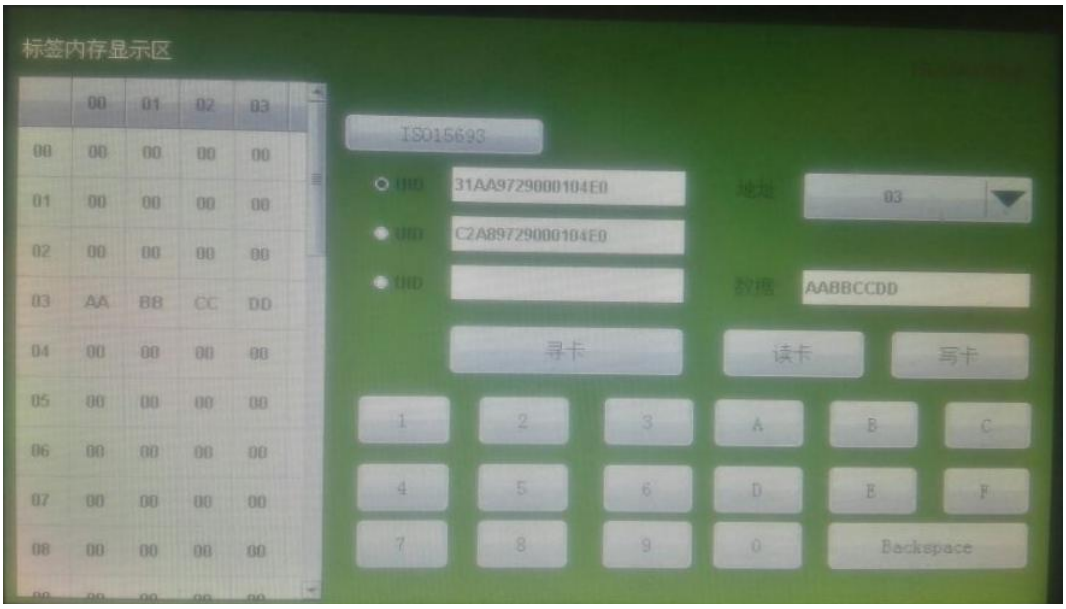

图 4.1.4 读卡

- 写单个数据块

写卡和读卡区别不大，选中要操作的标签号和地址，在数据栏输入想写的内容，点击写卡即刻写入。需要注意的是，写卡成功后标签内存显示区不会刷新，想看地址上新的数据需要再读卡。

4.2 高频 HF-13.56M 15693 开发示例

1、实验目的

- 掌握 HF 13.56MHz RFID 模块 ISO-15693 串口通信协议
- 了解 HF 13.56MHz RFID 模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用
- 了解 13.56MHz RFID 的应用范围及领域

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、实验内容

- 15693 标签寻卡
- 15693 标签内存读写数据

4、实验原理

1356M_15693 串口通信协议

通信协议的目的是当上位机与下位机模块进行通信时， 使下位机模块能够有效识别出上位机传输的命令， 使上位机能够正确读取出下位机模块的参数和状态。 上位机向下位机模块发送指令时， 必须使用请求帧协议的格式， 将指令和数据通过串口发送到下位机模块上。 下位机模块也只能识别请求帧协议格式的数据， 其他数据无法正确解析。 当下位机模块向上位机返回数据时， 发送的是响应帧格式的数据， 上位机只要按照响应帧的格式就可以将数据解析出来。

1356M_15693 的请求帧和响应帧的格式采用同样的格式。

帧头	协议长度	设备号	命令	状态/标志	数据	校验和
SOF	Length	Dev_ID	CMD	Statuc/Flag	VData	FCS
16bit	16bit	16bit	16bit	8bit	...	8bit

表 4.2.1 协议格式

SOF： 主机（ 如 F407） 与 13.56MHz 模块通信的命令帧起始字。

Length： 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： 设备编号， 用于扩展使用。

CMD： 命令字， 如寻卡命令字为 0x1000。

Statuc/Flag： 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： 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。

1356M_15693 具体命令请阅读“13.56MHz_15693 串口通讯协议.pdf” 文档。

5、实验步骤

❖ 开发工具选择

我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

❖ 创建工程

直接使用光盘中的 STemWin 模板工程。

❖ UI 设计

同之前一样，我们先在 GUIBuilder 上完成初步设计，细节可微调。

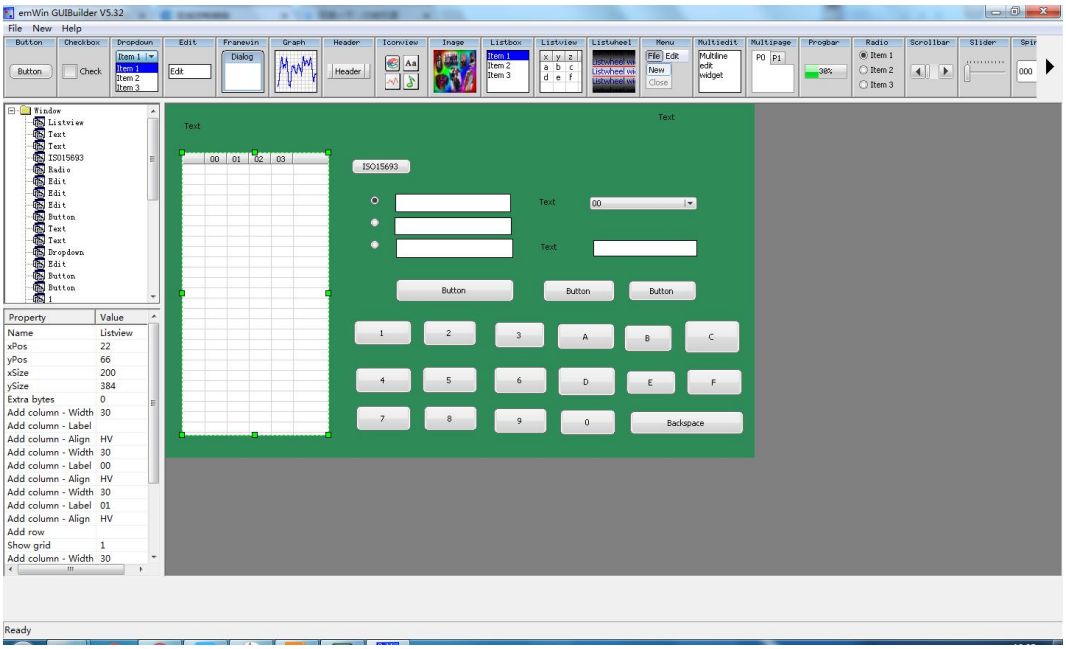

图 4.2.1 模拟器设计 UI

❖ 编码

通信方面按照帧格式逐节解析，最后采用异或校验。

```
void BSP_ST16C554_CS1D_ISR(void)
{
    uint8_t ret, i, flag;
    if(GPIO_ReadInputDataBit(GPIOF, GPIO_Pin_9))
    {
        EXTI_ClearITPendingBit(EXTI_Line9);
```

```
        ret = BSP_ST16C554_CS1D_Read();
//        USART_SendData(USART1, ret);
        printf("%x\r\n", ret);
        if(FLAG)
        {
          FLAG = 0;
          return;
        }
        switch(recSta)
        {
        case RC632_STA_SOP1:
            len = 0;
            if(ret == 0xaa)
            {
                drbuf[len++] = ret;
                recSta = RC632_STA_SOP2;
            }
            break;
        case RC632_STA_SOP2:
            if(ret == 0xbb)
            {
                drbuf[len++] = ret;
                recSta = RC632_STA_LEN1;
            }
            else recSta = RC632_STA_SOP1;
            break;
        case RC632_STA_LEN1:
            dataLen = ret;
            drbuf[len++] =  ret;
            recSta = RC632_STA_LEN2;
            break;
        case RC632_STA_LEN2:
            drbuf[len++] = ret;
            recSta = RC632_STA_DAT;
            break;
        case RC632_STA_DAT:
            drbuf[len++] = ret;
                      if(drbuf[len-1] == 0xaa) FLAG = 1;
            if(len == dataLen + 3)
                recSta = RC632_STA_FCS;
            break;
        case RC632_STA_FCS:
            recSta = RC632_STA_SOP1;
            flag = drbuf[4];
```

```
            for(i = 4; i < 3 + dataLen; i++)
            {
                flag ^= drbuf[i];
            }
            dataLen = 0;
            if(flag == ret)
            {
                drbuf[len++] = ret;
                rxflag = 1;
            }
        }
    }
}
```

单选按钮和下拉列表处理方式类似，都是用一个全局变量记录选择哪项，再根据全局变量的值选择对应位置输出或者内存操作。两个都用到了控件状态改变的消息，这些消息的机制是如果控件状态发生变化则触发消息，所以程序刚加载进去不会触发这个消息，全局变量得不到数值，需要在程序中提前把全局变量的数值初始化为控件初始状态的值。

```
        case ID_DROPDOWN_0:
            switch(NCode)
            {
            case WM_NOTIFICATION_CLICKED:

                break;
            case WM_NOTIFICATION_RELEASED:
                break;
            case WM_NOTIFICATION_SEL_CHANGED:
                dizhi = DROPDOWN_GetSel(WM_GetDialogItem(hWin, ID_DROPDOWN_0));
                break;
            }
            break;
        case ID_RADIO_0:
            switch(NCode)
            {
            case WM_NOTIFICATION_CLICKED:
                break;
            case WM_NOTIFICATION_RELEASED:
                break;
            case WM_NOTIFICATION_VALUE_CHANGED:
                UID = RADIO_GetValue(WM_GetDialogItem(hWin, ID_RADIO_0));
                break;
            }
            break;
```

第五章　超高频 UHF-900M 实验及实例

5.1 UHF 相关实验

1、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： RFID 高级教学科研平台（Ⅱ型）模块测试软件

2、实验内容

- 了解 UHF 的基本概念、 国际标准、 协议内容
- 了解 UHF 的标准接口
- 了解 UHF 的应用范围及领域
- 掌握对功率和功放相关命令的操作

3、实验原理

UHF 协议

超高频射频识别系统的协议目前有很多种， 主要可以分为两大协议制定者： 一是 ISO（国际标准化组织）； 二是 EPC Global。 ISO 组织目前针对 UHF（超高频） 频段制定了射频识别协议 ISO 18000－6，而 EPCGlobal 组织则制定了针对产品电子编码（ Electronic Product Code）超高频射频识别系统的标准。目前， 超高频射频识别系统中的两大标准化组织有融合的趋势， EPC Class 1 Generation 2 标准可能会变成 ISO18000-6 标准的 Type c。本文主要讨论的是针对 ISO 18000－6 标准的射频识别系统，本节讨论的是 ISO 18000－6 协议中与系统架构相关的物理层参数。

ISO 18000-6 目前定义了两种类型： Type A 和 Type B。 下面对这两种类型标准在物理接口、 协议和命令机制方面进行分析和比较。

1． 物理接口

ISO 18000-6 标准定义了两种类型的协议—Type A 和 Type B。 标准规定： 读写器需要同时支持两种类型， 它能够在两种类型之间切换， 电子标签至少支持一种类型。

（1） Type A 的物理接口

Type A 协议的通信机制是一种“读写器先发言”的机制， 即基于读写器的命令与电子标签的应答之间交替发送的机制。 整个通信中的数据信号定义为以下四种： “0”， “1”， “SOF”， “EOF”。

通信中的数据信号的编码和调制方法定义为：

① 读写器到电子标签的数据传输

读写器发送的数据采用 ASK 调制， 调制指数为 30%（误码不超过 3%）。

数据编码采用脉冲间隔编码， 即通过定义下降沿之间的不同宽度来表示不同的数据信号。

② 电子标签到读写器的数据传输

电子标签通过反向散射给读写器传输信息， 数据速率为 40kbits。 数据采用双相间隔码来进行编码，是在一个位窗内采用电平变化来表示逻辑， 如果电平从位窗的起始处翻转， 则表示逻辑“1”； 如果电平除了在位窗的起始处翻转， 还在位窗的中间翻转， 则表示逻辑“0”。

（ 2） Type B 的物理接口

Type B 的传输机制也是基于“读写器先发言”， 即基于读写器命令与电子标签的应答之间交换的机制。

① 读写器到电子标签的数据传输

采用 ASK 调制， 调制指数为 11%或 99% ， 位速率规定为 10kbits 或 40kbits， 由曼彻斯特编码来完成。具体来说就是一种 on-offkey 格式， 射频场存在代表“1”， 射频场不存在代表“0”。 曼彻斯特编码是在一个位窗内采用电平变化来表示逻辑“1”（ 下降沿） 和逻辑“0”（ 上升沿） 的。

② 电子标签到读写器的数据传输

同 TypeA 一样，通过调制入射并反向散射给读写器来传输信息，数据速率为 40kbits，同 Type A 采用一样的编码。

2． 协议和命令

（ 1） Type A 协议和命令

① 命令格式

由读写器发送给电子标签的数据按照如表 5.1.1 所示的帧格式组成。

静默	帧头	命令	帧尾

表 5.1.1 帧格式

开始的静默（ Quiet） 是一段持续时间至少为 300ps 的无 调制载波， SOF 是帧开始标志。 在发送完 EOF 结束标志以后， 读写器必须继续维持一段时间的稳定载波来提供电子标签应答的能量。

命令包含下列各部分区域， 见表 5.1.2。

保留（RFU flage）	命令码	命令标志	参数	数据区	CRC16 或 CRC15

表 5.1.2 Type A 读写器的命令格式

RFU 位， 保留作为协议的扩展； 命令码的长度是 6 位； 命令标志的长度是 4 位； 使用 CRC16 或者 CRC5 取决于命令的位数， 可在不同长度的命令中分别采用不同位数的 CRC 编码。

电子标签的应答格式见表 5.1.3， 应答包含下列区域： 帧头、 标志位、 一个或更多的参数、 数据、16 位的 CRC 编码。

帧头	标志（flags）	参数（Parameters）	数据	CRC

表 5.1.3 电子标签的应答格式

② 数据和参数

在 Type A 协议的通信中可能会用到以下的数据内容和参数信号， 如表 5.1.4 所示。

数据段	说明
UID	电子标签 64 未的唯一标识符，具体分配：高 8 位定义为“E0”，接着是 8 位的 IC 制造商码和 48 位的唯一序列号（只有 Get system_information 命令才返回完整 UID）
SUID	作为 UID 的子集，被用在冲突识别过程的绝大部分命令和应答中，它是一个 40 位的识别符：8 位 IC 制造商码和 UID 序列号的第 32 位
AFI	1 字节编码，定义了所有应用类及子类，该标志符被用于指定电子标签的目标应用类型，可以被编辑或锁定
DSFID	1 字节编码，定义了电子标签存储器中的数据结构，可以被编辑或锁定。

表 5.1.4 数据段说明

命令标志段： 一个 4 位的数据， 用来规定电子标签的工作和数据段的有效性。 其中 1 位的标志定义命令是否使用在防冲突过程中， 其它三位根据具体情况有不同的定义。

数据段：定义了电子标签的识别码和数据结构，另外，为加快识别过程，还定义了一个较短的识别码。

③ 存储器寻址

Type A 可以寻址最多可达 256 个块， 每个块最多可以包含 256 位容量。 整个电子标签的存储容量最多可达 64K 位。

④ 通信中的一些时序规定

电子标签应该在无电或者电源不足的情况下保持它的状态至少 300μs，特别是当电子标签处于静默状态时， 电子标签必须保持该状态至少 2s， 可以用复位（Reset_to_ready） 命令退出该状态。

电子标签从读写器接收到一个帧结束（EOF） 以后， 需要等待帧结束（EOF） 的下降沿开始计时的一段时间后才开始回发， 等待的时间根据时隙延迟标志确定， 一般在 150μs 以上。

读写器对于一个特定的电子标签的应答必须在一个特定的时间窗口里发送， 这个时间从电子标签的最后一个传输位结束后的第 2 位和第 3 位的边界开始， 持续 2．75 个电子标签位。

读写器在发送命令以前至少 3 位内不得调制载波。 读写器在电子标签最后一个传输位结束后的第 4 个位时内发送命令帧的第一个下降沿。

（2） Type B 协议和命令

① 命令格式

Type B 中， 读写器命令包含下列各区域， 见表 5.1.5。

帧头探测段	帧头	分割符	命令	参数	数据	CRC

表 5.1.5 Type B 读写器的命令格式

帧头探测段是一个至少持续 400μs 的稳定无调制载波（相当于 16 位数据的传输）。

帧头是 9 位曼彻斯特“0”， NRZ 格式就是 010101010101010101。

分割符是用来区分帧头和有效数据的，共定义了五种，经常使用第一种 5 位分割符（ 110011 10 10）。

命令和参数段没有作明确定义。

CRC 采用 16 位 CRC 编码。

Type B 中， 电子标签的应答格式见表 5.1.6。

静默	返回帧头	数据	CRC

表 5.1.6 Type B 电子标签的应答格式

静默是电子标签持续 2 字节的反向散射（ 40kbits 的速率相当于 400μs 的持续时间）。

返回帧头是一个 16 位数据“000001 01 01 01 01 01 01 01 0001 10 11 0001”。

CRC 采用 16 位数据编码。

② 数据和参数

在 Type B 协议的通信中可能用到以下的数据内容和参数信号。

电子标签包含一个唯一独立的 UID 号， 包含一个 8 位的标志段（ 低四位分别表示 4 个标志， 高四位保留， 通常为“0”）。

64 位 UID 包含 50 位的独立串号、 12 位的 Foundry code 和一个两位的校验和。

③ 存储器寻址

电子标签通过一个 8 位的地址区寻址， 因此它总共可以寻址 256 个存储器块， 每个块包含一个字节数据， 整个存储器将可以最多保存 2K 位数据。

存储器的 0～17 块被保留用作存储系统信息， 18 块以上的存储器用作电子标签中普通的应用数据存储区。

每个数据字节包含响应的锁定位， 可以通过 Lock 命令将该锁定位锁定， 可以通过 Query lock（ 查询锁定） 命令读取锁定位的状态， 电子标签的锁定位不允许被复位。

④ 通信中的一些时序规定

电子标签向存储器写操作的等待阶段，读写器需要向电子标签提供至少 15ms 的稳定无调制载波。在写操作结束以后， 读写器需要发送 10 个“01”信号。 同时， 在读写器的命令之间发生频率跳变时， 或者读写器的命令和电子标签的应答之间发生跳变时， 在跳变结束后也需要读写器发送 10 个“01”信号。电子标签将使用反向调制技术回发数据给读写器， 这就需要整个回发过程中读写器必须向电子标签提供稳定的能量，同时检测电子标签的应答。 在电子标签发送完应答以后， 至少需要等待 400μs 才能再次接收读写器的命令。

Q 值算法

在 Q 值算法中，阅读器首先发送 Query 命令，该命令中含有一个参数 Q（取值范围 0～15），接收到

命令的标签可在[0，2Q－1]范围内（称为帧长）随机选择时隙，并将选择的值存入标签的时隙计数器中，只有计数器为 0 的标签才能响应，其余标签保持沉默状态。当标签接收到阅读器发送的 QueryRep 命令时，将其时隙计数器减 1，若减为 0，则给阅读器发送一个应答信号。标签被成功识别后，退出这轮盘存。当有两个以上标签的计数器都为 0 时，它们会同时对阅读器进行应答，造成碰撞。阅读器检测到碰撞后，发出指令将产生碰撞的标签时隙计数器设为最大值（2Q－1），继续留在这一轮盘存周期中，系统继续盘存直到所有标签都被查询过，然后阅读器发送重置命令，使碰撞过的标签生成新的随机数。

根据上一轮识别的情况，阅读器发送 Query－Adjust 命令来调整 Q 的值，当标签接收到 Query－Adjust 命令时，先更新 Q 值，然后在[0，2Q－1]范围内选择随机值。EPCClass1Gen2 标准中提供了一种参考算法来确定 Q 值的范围.其中：Qfp 为浮点数，其初值一般设为 4．0，对 Qfp 四舍五入取整后得到的值即为 Q；C 为调整步长，其典型取值范围是 0．1<C<0．5，通常当 Q 值较大时，C 取较小的值，而当 Q 值较小时，C 取较大的值。

当阅读器发送 Query 命令进行查询时，若应答标签数等于 1，则 Q 值不变；若应答标签数等于 0（空闲时隙），则减小 Q 值；若应答标签数大于 1（冲突时隙），则增大 Q 值。

该算法在参数 C 的辅助下对 Q 值进行动态调整，但是 C 太大会造成 Q 值变化过于频繁，导致帧长调整过于频繁，C 太小又不能快速地实现最优帧长的选择。因此，研究者们对 Q 值的调整进行了各种优化。

4、实验步骤

1) 准备好硬件， 准备好硬件， RFID 900MHz 模块和 Gen2 标签， RFID 900M 模块接上配套天线，（ 注： 上电之前必须先接好天线）。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 功能测试

- 识别标签

点击单步识别，F407 会发送一次寻卡指令；点击识别标签，F407 会连续发送寻卡指令直到用户点击停止，如图 5.1.2。

图 5.1.1 和图 5.1.2 的设置仅用在识别一张卡的时候，当同时有多张卡需要识别，需要点上防碰撞识别。防碰撞识别 Q 值是防碰撞识别中的一个参数，在上节实验原理中有 Q 值算法的说明，简单来说，Q 值的选择，决定了识别多张标签卡的效率，合理的 Q 值是让所有标签卡恰好分配在 Q 的取值范围内（[0,2Q-1]），既没有空闲时隙，又没有冲突时隙，最大化查询效率。由于实验提供的标签卡较少，Q 值的选择范围仅提供了 0-3（可以分配六张卡）。同时针对 Q 值的计算，有很多改良算法，有兴趣的同学可以查阅相关论文学习。

图 5.1.1 单步识别

图 5.1.2 识别标签

点击内存窗口切换到内存操作。

- 读取数据

读取标签数据分两种情况： 指定 UII 和不指定 UII。 指定 UII 是必须指定一个 UII 才能读取数据，如果有多张卡则读取指定卡号的数据， 一般只有一张卡的时候使用不指定 UII 进行读取。 图 5.1.3 所示为读取标签 User 数据块的数据。 如果不指定 UII， 请勾选不指定 UII， 如果需要指定 UII， 请将 UII 填写到标签 ID 处。

900M 标签的数据分四块， 由两位二进制位编号， 00 为 Reserved 区， 01 为 UII 区， 10 为 TID 区， 11 为 User 区。下面是地址， 不同的数据块有不同的地址长度， 都以 0 开始。 长度为您要读取的

长度， 默认采用 8 个 0 为访问密码， 默认密码直接写入程序的发送指令， 用户无需填写。 在读取的时候地址和长度需要输入正确， 例如 UII 数据块， 最大长度是 8 个字， 因此您的地址可以输入 0,1…,7，而您的长度可以对应的输入 8,7…1， 也就是必须满足地址加长度不能超过 8。 其它数据块也类似， 只不过长度有所变化，其中 User 为 32， TID 为 12， Reserved 为 4。

图 5.1.3 把滚动条拖到后面可以看到读取到的数据，指定 UII 会显示标签 ID 号，不指定 UII 不会显示。

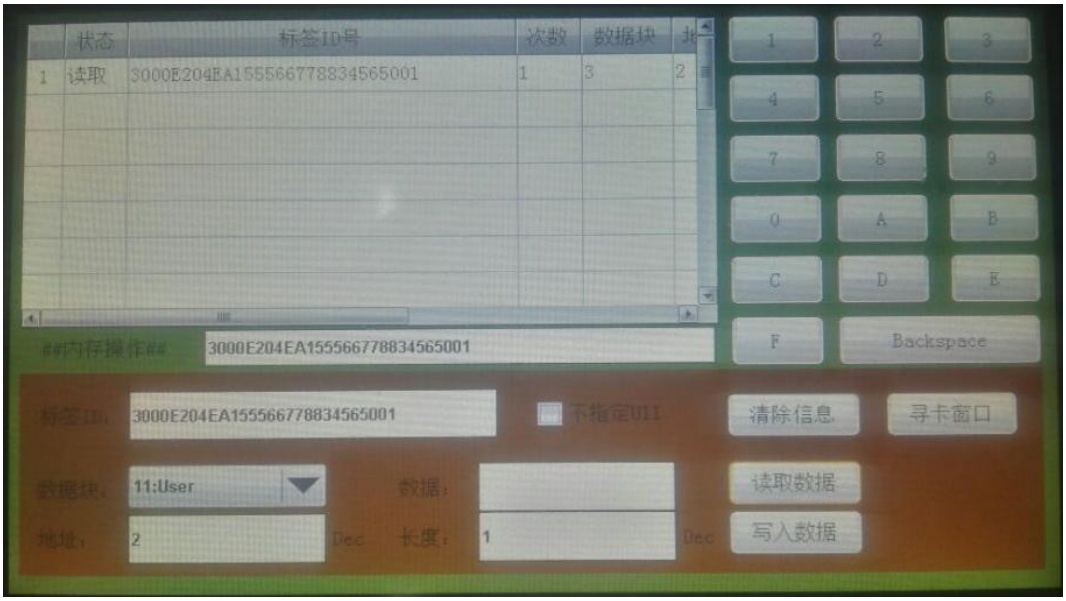

图 5.1.3 指定 UII 读取数据

- 写入数据

写入数据和读取数据类似， 但是长度只能是 1， 也就是只能单个字进行写入。 如果您需要写入多个字是可以逐个写入。 写入如图 5.1.4 所示：

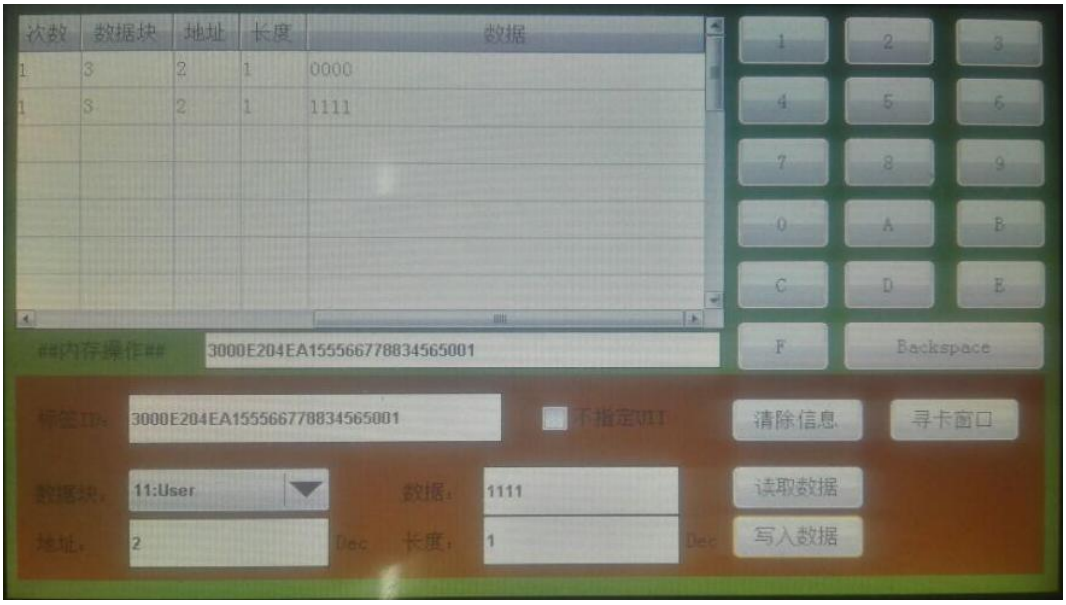

图 5.1.4 指定 UII 写入数据

写入的时候需要注意， 当修改 UII 数据块地址 1 处的数据， 会引起卡号的变更， UII 的第一个字为协议控制字， 定义如表 5.1.7 所示：

位	Bit15~Bit7	Bit6~Bit5	Bit4~Bit0
功能	NSI（未使用）	未定义	以word（两个字节）为单位的PC和UII的总体长度

注： UII 从低位开始传输

表 5.1.7 PCBits 字段的定义

如果您不想修改 UII 的长度， 请不要修改地址 1 的数据。

5.2 超高频 UHF-900M 开发实例

1、实验目的

- 掌握 UHF 900MHz RFID 模块串口通信协议
- 了解 UHF 900MHz RFID 模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用
- 了解 900 MHz RFID 的应用范围及领域

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、实验内容

- 熟悉 900M 标签卡号读取
- 了解防碰撞检测算法
- 了解 900M 标签内存读取

4、实验原理

900M 串口通信协议

通信协议的目的是当上位机与下位机模块进行通信时， 使下位机模块能够有效识别出上位机传输的命令， 使上位机能够正确读取出下位机模块的参数和状态。 上位机向下位机模块发送指令时， 必须使用请求帧协议的格式， 将指令和数据通过串口发送到下位机模块上。 下位机模块也只能识别请求帧协议格式的数据， 其他数据无法正确解析。 当下位机模块向上位机返回数据时， 发送的是响应帧格式的数据， 上位机只要按照响应帧的格式就可以将数据解析出来。具体串口协议请参照串口协议部分的 900MHz 串口通讯协议文档。

5、实验步骤

- 开发工具选择

  我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

- 创建工程

  直接使用光盘中的 STemWin 模板工程。

- UI 设计

同之前一样，我们先在 GUIBuilder 上完成初步设计，细节可微调。这里需要补充一点，模拟器上无法在一个窗口同时显示两个 Window 控件，需要分层创建，再在主界面中调用子界面。

❖ 编码

波特率 57600，通信协议部分结尾使用的固定数值，没有加入校验。

```
void BSP_ST16C554_CS2A_ISR(void)
{
   uint8_t ret;
   if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_1))
   {
      EXTI_ClearITPendingBit(EXTI_Line1);
      ret = BSP_ST16C554_CS2A_Read();
      if(ret == 0xff) return;
      switch(recSta)
      {
      case RC632_STA_SOP:
         rxflag = 0;
         len = 0;
         if(ret == 0xaa)
         {
            drbuf[len++] = ret;
            recSta = RC632_STA_LEN;
         }
         break;
      case RC632_STA_LEN:
         dataLen = ret;
         drbuf[len++] =  ret;
         recSta = RC632_STA_DAT;
         break;
      case RC632_STA_DAT:
         drbuf[len++] = ret;
         if(len == dataLen+1)
            recSta = RC632_STA_EOF;
         break;
      case RC632_STA_EOF:
         recSta = RC632_STA_SOP;
         dataLen = 0;
         drbuf[len++] = ret;
         if(ret == 0x55)
            rxflag = 1;
         break;
      }
   }
}
```

内存操作界面和之前的实验代码大同小异，不再多做介绍，重点讲解一下寻卡窗口的识别标签功能。

点击识别标签系统会连续发送寻卡指令，此时用户无法做其他操作，只有点击停止才可以继续进行实验别的操作，下面是 emWin 实现这部分功能的代码：

```
    case ID_BUTTON_shibiebiaoqian:
          switch(NCode)
          {
          case WM_NOTIFICATION_CLICKED:
             hWin3 = CreateWindowPage3();
             WM_DisableWindow(WM_GetDialogItem(hItem,ID_BUTTON_qingchuxinxi));
             WM_DisableWindow(WM_GetDialogItem(hItem,ID_BUTTON_danbushibie));
             WM_DisableWindow(WM_GetDialogItem(hItem,ID_BUTTON_neicunchuangkou));
             tingzhi = 1;
             break;
          case WM_NOTIFICATION_RELEASED:
             break;
          case WM_NOTIFICATION_MOVED_OUT:
             break;
          }
          break;
```

看到点击识别标签按键，首先新建了一个窗口，显而易见新窗口是停止按键所在的位置，后面把其他可以点击的按键失能，这样看起来就完成了“禁用”其他功能。

```
    case ID_BUTTON_tingzhi:
          switch(NCode)
          {
          case WM_NOTIFICATION_CLICKED:
             GUI_EndDialog(hItem, 0);
             WM_EnableWindow(WM_GetDialogItem(hWin1,ID_BUTTON_qingchuxinxi));
             WM_EnableWindow(WM_GetDialogItem(hWin1,ID_BUTTON_danbushibie));
             WM_EnableWindow(WM_GetDialogItem(hWin1,ID_BUTTON_neicunchuangkou));
             tingzhi = 0;
             break;
          case WM_NOTIFICATION_RELEASED:
             break;
          case WM_NOTIFICATION_SEL_CHANGED:
             break;
          }
          break;
```

上面是新窗口停止功能的代码，用户点击停止说明要停止寻卡，界面需要恢复正常。所以首先关闭停止按键所在窗口，接着使能那些之前被我们失能的功能按键即可。

第六章　有源 2.4G 实战及实例

6.1 有源 2.4G 相关实验

图 6.1.1 RFID 高级教学科研平台（Ⅱ型）-2.4GHz 模块

1、实验环境

- 硬件：RFID 高级教学科研平台（Ⅱ型），PC 机，J-Link，有源 2.4G 标签
- 软件：Keil μVision5 IDE，低频 LF-125K 读卡器 emWin 测试程序

2、实验内容

- 了解有源 2.4G RFID 的技术原理
- 了解有源 2.4G RFID 系统的基本结构
- 了解有源 2.4G RFID 的应用范围及领域

3、实验原理

3.1 什么是有源 RFID

有源 RFID，又称为主动式 RFID（Active tag），依据电子标签供电方式的不同进行划分的电子标签的一种类型，通常支持远距离识别。电子标签可以分为有源电子标签(Active tag)、无源电子标签(Passive tag)和半无源电子标签(Semi-passivetag)。有源电子标签内装有电池，无源射频标签没有内装电池，半无源电子标签(Semi-passive tag)部分依靠电池工作。

3.2 有源 RFID 如何构成

RFID 是一种简单的无线系统，由两个基本器件组成，询问器（或阅读器）和很多应答器（或标签），同时辅以天线、外围网络、中间件、管理系统，从而形成完整的 RFID 应用系统。

3.3 有源电子标签如何构成

有源 RFID 电子标签由中心处理器（MCU）、通讯芯片和外围电路组成。

3.4 有源阅读器如何构成

有源 RFID 阅读器有中心处理器（ MCU）、 通讯芯片、 接口电路、 存储单元和外围电路组成， 可以实现接收数据的解析、 处理和分析。

3.5 有源电子标签如何工作

RFID 标签俗称电子标签， RFID 标签中存储一个唯一编码， 其地址空间大大高于条码所能提供的空间，因此可以实现单品级的物品编码。标签上电后，按照预设的规则周期性的进行信号发射，当 RFID 标签进入读写器的作用区域， 阅读器获取到标签发射出来的信息， 即完成了对标签的识别过程。

3.6 有源阅读器如何工作

阅读器是对 RFID 标签进行读/写操作的设备， 主要包括射频模块和数字信号处理单元两部分。 读写器是 RFID 系统中最重要的基础设施， 一方面， RFID 标签返回的电磁信号通过天线进入读写器的射频模块中转换为数字信号， 再经过读写器的数字信号处理单元对其进行必要的加工整形， 最后从中解调出返回的信息， 完成对 RFID 标签的识别或读/写操作； 另一方面， 上层中间件及应用软件与读写器进行交互， 实现操作指令的执行和数据汇总上传。

在上传数据时， 读写器会对 RFID 标签原子事件进行去重过滤或简单的条件过滤， 将其加工为读写器事件后再上传， 以减少与中间件及应用软件之间数据交换的流量， 因此在很多读写器中还集成了微处理器和嵌入式系统， 实现一部分中间件的功能， 如信号状态控制、 奇偶位错误校验与修正等。

3.7 决定有源 RFID 的主要参数

1） 工作频段

2） 读写器读取距离

3） 防碰撞性能(读写器同时读取标签数量）

4） 读写器灵敏度

5） 标签存储器容量

6） 标签电池寿命、 发射功率、 接收灵敏度

7） 标签尺寸、 形状、 防护（防水、 防尘、 防腐、 防爆性） 等级

8） 抗干扰能力（同频信号干扰下是否正常工作）

9） 稳定性（标签漏读率）

10） 安全性（加密方式）

11） 扩展性（是否支持 RSSI、 TDOA 等算法定位、 传感器结合）

4、实验步骤

1) 准备好硬件， 有源 2.4G 读卡器和有源 2.4G 标签。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 打开 2.4G 有源标签电源，并把标签放在 2.4G 读卡器附近，屏幕上会显示卡号同时计数，点击右下角按键可以计数清零或卡号清除，可以记录多张卡。

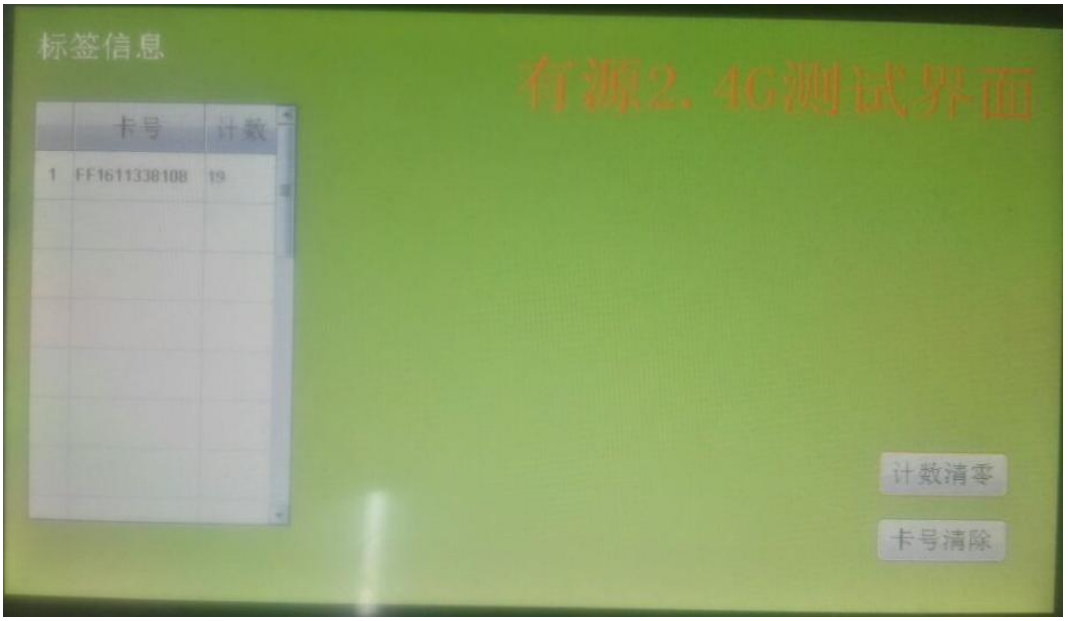

图 6.1.2 有源 2.4G 测试界面

6.2 有源 2.4G 开发实例

1、实验目的

- 掌握有源 2.4G RFID 模块串口通信协议
- 了解有源 2.4G RFID 模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用
- 了解有源 2.4G 的应用范围及领域

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、 实验内容

- 有源 2.4G 标签寻卡

4、实验原理

有源 2.4G RFID 标签周期性（2 秒） 的上报 ID 信息， 2.4G RFID 读卡器识别标签 ID 并上报给 F407 核心板。 2.4G RFID 主要应用在人员资产定位领域。

5、实验步骤

- 开发工具选择

  我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

- 创建工程

  直接使用光盘中的 STemWin 模板工程。

- UI 设计

  本实验的 UI 非常简单，在 GUIBuilder 上可轻易完成。

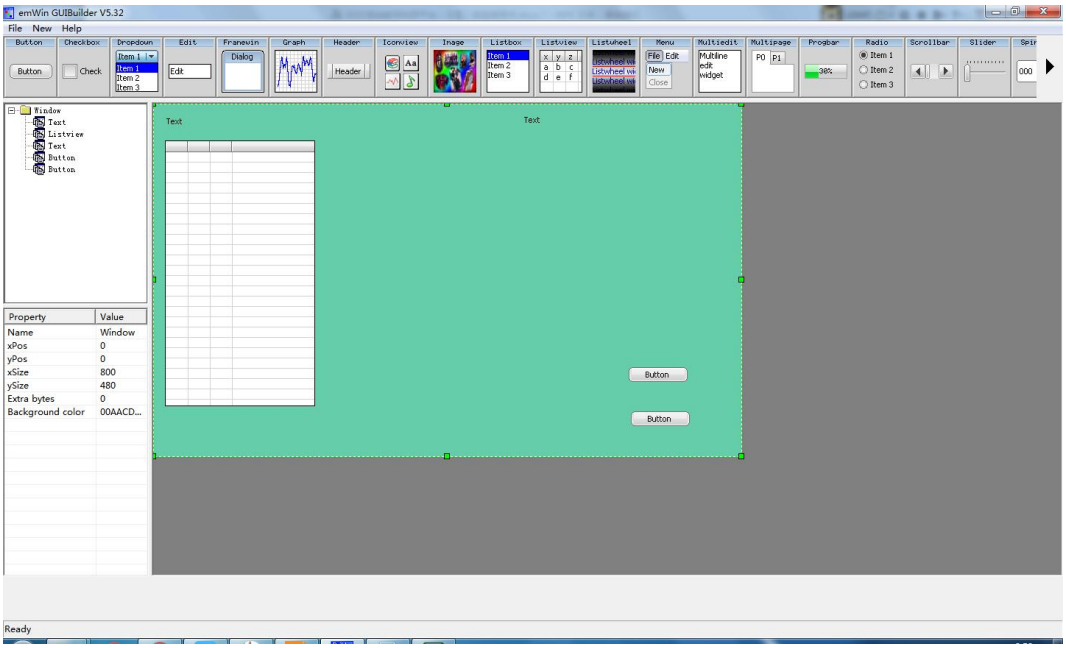

图 6.2.1 GUIBuilder UI 设计

❖ 编码

有源 2.4G 标签卡号自动上报，无校验，按长度直接接收。

```
void BSP_ST16C554_CS1C_ISR(void)
{
    uint8_t ret;
    if(GPIO_ReadInputDataBit(GPIOF, GPIO_Pin_8))
    {
        EXTI_ClearITPendingBit(EXTI_Line8);
        ret = BSP_ST16C554_CS1C_Read();
        //USART_SendData(USART1, ret);

        switch(recSta)
        {
        case LF125K_STA_SOP:
            drbuf[len++] = ret;
            recSta = LF125K_STA_DAT;
            break;
        case LF125K_STA_DAT:
            drbuf[len++] = ret;
            if(len == 5) recSta = LF125K_STA_FCS;
            break;
        case LF125K_STA_FCS:
            recSta = LF125K_STA_SOP;
            drbuf[len++] = ret;
            rxflag = 1;
            break;
```

```
        }
            printf("%d %x\r\n", len, ret);
    }
}
```

由于要显示多个卡号，需要在主函数中查找当前读到的卡号是否在前面出现过，如果出现过则出现位置的计数加一，没有出现过则添加新的一行。

```
            cnt = 0;
            flg = 0;
            rxflag = 0;
            for(i = 0; i < 12; i++)
            {
                temp[cnt++] = num_to_char(drbuf[i]/16);
                temp[cnt++] = num_to_char(drbuf[i]%16);
            }
            temp[12] = '\0';
            for(i = 0; i < ct; i++)
            {
                if(Same(temp, ka[i].kahao))
                {
                    ka[i].num++;
                    num_to_str(ka[i].num);
                    LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, i,
prtnum);
                    flg = 1;
                }
            }
            if(!flg)
            {
                strcpy(ka[ct].kahao, temp);
                ka[ct].num++;
                num_to_str(ct+1);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 0, ct,
prtnum);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 1, ct,
ka[ct].kahao);
                num_to_str(ka[ct].num);
                LISTVIEW_SetItemText(WM_GetDialogItem(hWin,GUI_ID_LISTVIEW0), 2, ct,
prtnum);
                ct++;
            }
            len = 0;
```

第七章 原理机应用实验及实例

7.1 ISO15693 原理机相关实验

图 7.1.1 RFID 高级教学科研平台（Ⅱ型）-RFID 原理机模块

1、实验环境

- 硬件：RFID 高级教学科研平台（Ⅱ型），PC 机，J-Link，15693 标签
- 软件：Keil μVision5 IDE，RFID 原理机 emWin 测试程序

2、实验内容

- 熟悉 15693 协议
- 掌握原理机串口通信协议
- 了解 15693 的应用场景
- 深入了解 15693 的工作原理

3、实验步骤

1) 准备好硬件， RFID 原理机和 15693 标签。

2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。

3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。

4) 功能测试

- 寻卡

选择任一编码模式， 将 15693 标签放在 RFID 原理机 读卡区域， 点击寻卡， 系统会读出 15693 标签卡号并显示在屏幕上。

➢ 读单个数据块

必须先寻卡，再选择地址，地址提供 00-04 供用户体验。点击读卡，会读出相应地址的数据，显示在数据栏和屏幕左侧标签内存显示区的对应位置。

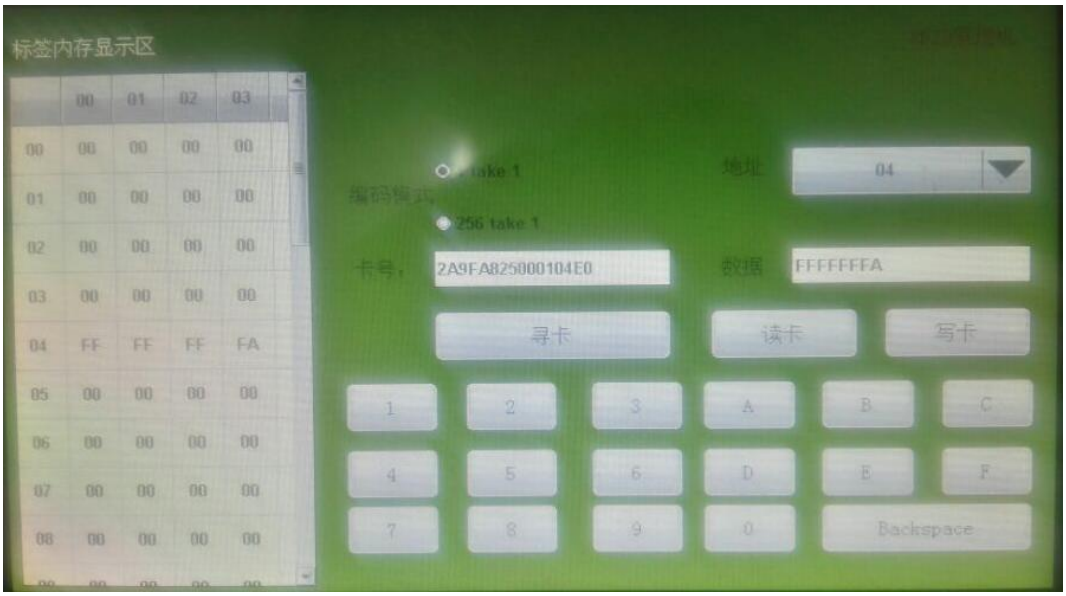

图 7.1.2 RFID 原理机寻卡和读卡

➢ 写单个数据块

先寻卡，选择要写入的地址，在数据栏填写八字节要写入的数据，点击写卡进行写入。

7.2 原理机应用开发实例

1、实验目的

- 掌握 ISO 15693 模块串口通信协议
- 了解原理机模块的读卡特性
- 掌握 STemWin 窗口管理器的使用方法
- 了解 STemWin 的消息机制
- 学会查阅 STemWin 的 API 并使用

2、实验环境

- 硬件： RFID 高级教学科研平台（Ⅱ型）实验箱， PC 机
- 软件： Keil μVision5 IDE

3、实验内容

- 15693 标签寻卡
- 15693 标签内存读写数据

4、实验原理

原理机串口通信协议

通信协议的目的是当上位机与下位机模块进行通信时， 使下位机模块能够识别出上位机传输的命令，使上位机能够正确读取出下位机模块的参数和状态。 上位机向下位机模块发送指令时， 必须使用请求帧协议的格式， 将指令和数据通过串口发送到下位机模块上。 下位机模块也只能识别请求帧协议格式的数据，其他数据无法正确解析。 当下位机模块向上位机返回数据时， 发送的是响应帧格式的数据， 上位机只要按照响应帧的格式就可以将数据解析出来。

帧头	协议长度	标志位	命令	数据	校验和	帧尾
SOF	Length	Flag	CMD	VData	FCS	EOF
8bit	8bit	8bit	8bit	...	8bit	8bit

表 7.2.1 命令帧协议格式

SOF： 帧头， 固定为 0x02。

Length： 一帧命令的长度， 从 SOF（ 包含） 开始到 EOF（ 包含） 结束为止的总长度。

Flag： 标志位， 包含编码模式、 数据率、 载波信息。

CMD： 命令字节。

VData： 可变长的数据值。

FCS： 校验和， 从 Length（ 包含） 字节到 FCS（ 不包含） 开始字节的所有字节的和模 256。

EOF： 帧尾， 固定为 0x03。

帧头	协议长度	标志位	命令
CMD	Flag	VData	CRC16/CCITT
8bit	8bit	8bit	8bit

表 7.2.2 响应帧协议格式

CMD： 命令字节。

Flag： 标志， 标志请求帧是否正确执行。

VData： 响应帧负载。

CRC16/CCITT： 16 位的 CRC 校验值。

原理机具体命令请阅读“13.56MHz 原理机串口通讯协议.pdf” 文档。

CRC16\CCITT

RFID 原理机 接收数据用到 CRC16\CCITT 校验，下面来解释一下这种校验码的计算方法。

循环冗余校验码（CRC）的基本原理是：在 K 位信息码后再拼接 R 位的校验码，整个编码长度为 N 位，因此，这种编码也叫（N，K）码。对于一个给定的（N，K）码，可以证明存在一个最高次幂为 N-K=R 的多项式 G(x)。根据 G(x)可以生成 K 位信息的校验码，而 G(x)叫做这个 CRC 码的生成多项式。 校验码的具体生成过程为：假设要发送的信息用多项式 C(X)表示，将 C(x)左移 R 位（可表示成 C(x)*2R），这样 C(x)的右边就会空出 R 位，这就是校验码的位置。用 C(x)*2R 除以生成多项式 G(x)得到的余数就是校验码。

任意一个由二进制位串组成的代码都可以和一个系数仅为‘0’和‘1’取值的多项式一一对应。例如：代码 1010111 对应的多项式为 x6+x4+x2+x+1，而多项式为 x5+x3+x2+x+1 对应的代码 101111。

CRC16\CCITT：多项式 x16+x12+x5+1（0x1021），初始值 0x0000，低位在前，高位在后，结果与 0x0000 异或。

5、实验步骤

❖ 开发工具选择

我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

❖ 创建工程

直接使用光盘中的 STemWin 模板工程。

❖ UI 设计

本实验的 UI 非常简单，在 GUIBuilder 上可轻易完成。

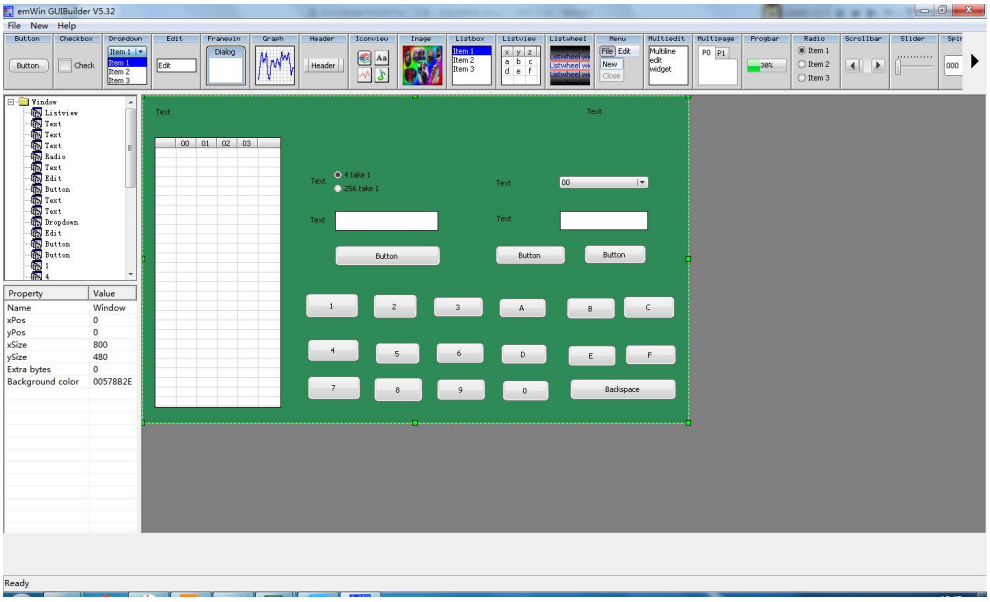

图 7.2.1 GUIBuilder UI 设计

❖ 编码

RFID 原理机采用离线解析，中断内先存一个定长的字符串（足够长，保证能从中解析到完整的数据帧），程序再对这个字符串进行解析。这样做的好处是，当数据帧很长的时候，在线解析边接收边解析存在两者速度不匹配的问题，解析速度较慢而接收速度很快，会出现丢帧的情况，而离线解析可以有效避免这种情形发生。

中断接收字符串代码：

```
void BSP_ST16C554_CS1A_ISR(void)
{
    uint8_t ret;
    if(GPIO_ReadInputDataBit(GPIOF, GPIO_Pin_6))
    {
        EXTI_ClearITPendingBit(EXTI_Line6);
        ret =  BSP_ST16C554_CS1A_Read();
        drbuf[len++] = ret;
        if(len > 4000) len = 0;
        printf("%x ", ret);
    }
}
```

校验用到了 CRC16\CCITT 算法，计算在实验原理小节有详解，在此直接给出 C 语言代码：

```
uint16_t crc = 0xffff;

uint16_t crc16_ccitt(uint8_t data,uint16_t crc1)
{
    uint16_t ccitt16 = 0x8408;//0x8408 为 0x1021 的逆序值,而 0x1021 为生成多项式 0x11021
    int i;
```

```
    crc1 ^= data; /*新的数据与原来的余数相加（加法就是异或操作） */
    for(i=0; i<8; i++)
    {
        if(crc1 & 0x0001) /*最低位为 1， 减去除数 */
        {
            crc1 >>= 1;
            crc1 ^= ccitt16;
        }
        else /*最低位为 0， 不需要减去除数 */
        {
            crc1 >>= 1; /*直接移位*/
        }
    }
    return crc1;
}
```

解析函数部分稍显复杂，用户应仔细阅读相关通信协议部分，对应代码进行分析：

```
void jieXi(int mode)
{
    int i, j;
    uint8_t ret;
    rxflag = 0;
    for(i = 0; i < 4000; i++)
    {
        if(drbuf[i] == cmd) rxflag = 1;
        if(rxflag)
        {
            ret = drbuf[i];
            switch(recSta)
            {
            case CMD:
                if(ret != cmd)
                {
                    rxflag = 0;
                    break;
                }
                Flag1 = 0;
                Flag2 = 0;
                LEN = 0;
                ANS[LEN++] = ret;
                crc = 0xffff;
                recSta = FLAG;
                break;
            case FLAG:
                ANS[LEN++] = ret;
```

```
                crc = crc16_ccitt(ret, crc);
                recSta = DATA;
                break;
            case DATA:
                ANS[LEN++] = ret;
                crc = crc16_ccitt(ret, crc);
                if(LEN == 11)
                {
                    if(ANS[2] & 0x01)
                        recSta = DSFID;
                    else
                    {
                        if(ANS[2] & 0x02)
                            recSta = AFI;
                        else
                        {
                            if(ANS[2] & 0x04)
                                recSta = VICC;
                            else
                            {
                                if(ANS[2] & 0x08)
                                    recSta = ICR;
                                else
                                    recSta = FCS;
                            }
                        }
                    }
                }
                break;
            case DSFID:
                ANS[LEN++] = ret;
                crc = crc16_ccitt(ret, crc);
                if(ANS[2] & 0x02)
                    recSta = AFI;
                else
                {
                    if(ANS[2] & 0x04)
                        recSta = VICC;
                    else
                    {
                        if(ANS[2] & 0x08)
                            recSta = ICR;
                        else
                            recSta = FCS;
```

```
                }
            }
            break;
        case AFI:
            ANS[LEN++] = ret;
            crc = crc16_ccitt(ret, crc);
            if(ANS[2] & 0x04)
                recSta = VICC;
            else
            {
                if(ANS[2] & 0x08)
                    recSta = ICR;
                else
                    recSta = FCS;
            }
            break;
        case VICC:
            ANS[LEN++] = ret;
            crc = crc16_ccitt(ret, crc);
            Flag1++;
            if(Flag1 == 2)
            {
                if(ANS[2] & 0x08)
                    recSta = ICR;
                else
                    recSta = FCS;
            }
            break;
        case ICR:
            ANS[LEN++] = ret;
            crc = crc16_ccitt(ret, crc);
            recSta = FCS;
            break;
        case FCS:
            ANS[LEN++] = ret;
            Flag2++;
            if(Flag2 == 1)
                recSta = FCS;
            else if(Flag2 == 2)
            {
                recSta = CMD;
                for(j = 0; j < 5000; j++) drbuf[j] = 0;
                len = 0;
                if(mode == 1) flag2 = 1;
```

```
                    if(mode == 2) flag3 = 1;
                }
                break;
            }
        }
        if(mode == 1 && flag2 == 1) break;
        if(mode == 2 && flag3 == 1) break;
    }
}
```

两种编码模式区分在信息标志 Flag 位数值不同，应用单选按钮的数值改变消息更新 Flag 数值。

```
    case ID_RADIO_0:
          switch(NCode)
          {
          case WM_NOTIFICATION_CLICKED:
             break;
          case WM_NOTIFICATION_RELEASED:
             break;
          case WM_NOTIFICATION_MOVED_OUT:
             break;
          case WM_NOTIFICATION_VALUE_CHANGED:
             if(RADIO_GetValue(WM_GetDialogItem(hWin, ID_RADIO_0)))
                Flag = 0x06;
             else
                Flag = 0x02;
             break;
          }
          break;
```

其余部分和 15693 标签的开发实例类似，不再多做赘述。

第八章 执行模块实例

1、实验目的

- 了解 RFID 技术的实际应用场景

2、实验环境

- 硬件：RFID 高级教学科研平台（Ⅱ型），PC 机，J-Link
- 软件：Keil μVision5 IDE，RFID 原理机 emWin 测试程序

3、实验内容

- 模拟地铁门禁闸机系统让用户体验 RFID 技术在现实的应用

4、实验步骤

1) 准备好硬件， 执行器部件直接安装在 RFID 平台上。
2) 将 STM32F407 核心板插到 RFID 高级教学科研平台（Ⅱ型）底板上，连接 J-Link，连接 5V 电源供电。
3) 打开光盘对应核心板的工程，编译无误后，可以通过 MDK 的 Download 按钮下载程序写到核心板。
4) 模拟地铁闸机通道，在显示屏上点击打开门禁闸机打开，点击关闭门禁闸机关闭。

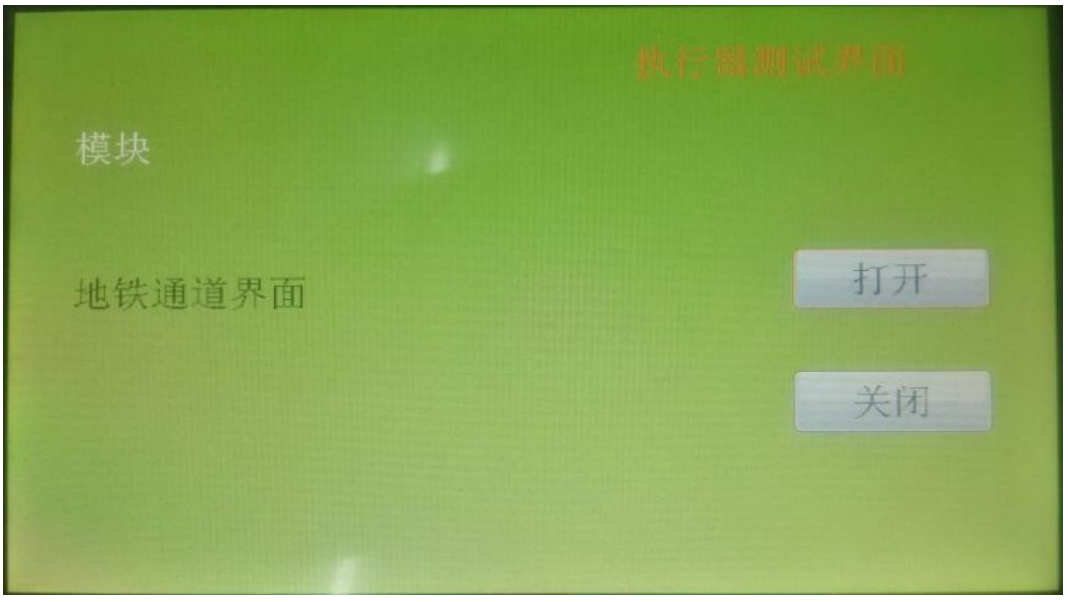

图 8.1 执行器操作界面

- 开发工具选择

  我们选择在 Windows 7 下用 Keil μVision5 IDE 进行开发，迎合当下潮流，开发方便快捷。

- 创建工程

  直接使用光盘中的 STemWin 模板工程。

❖ UI 设计

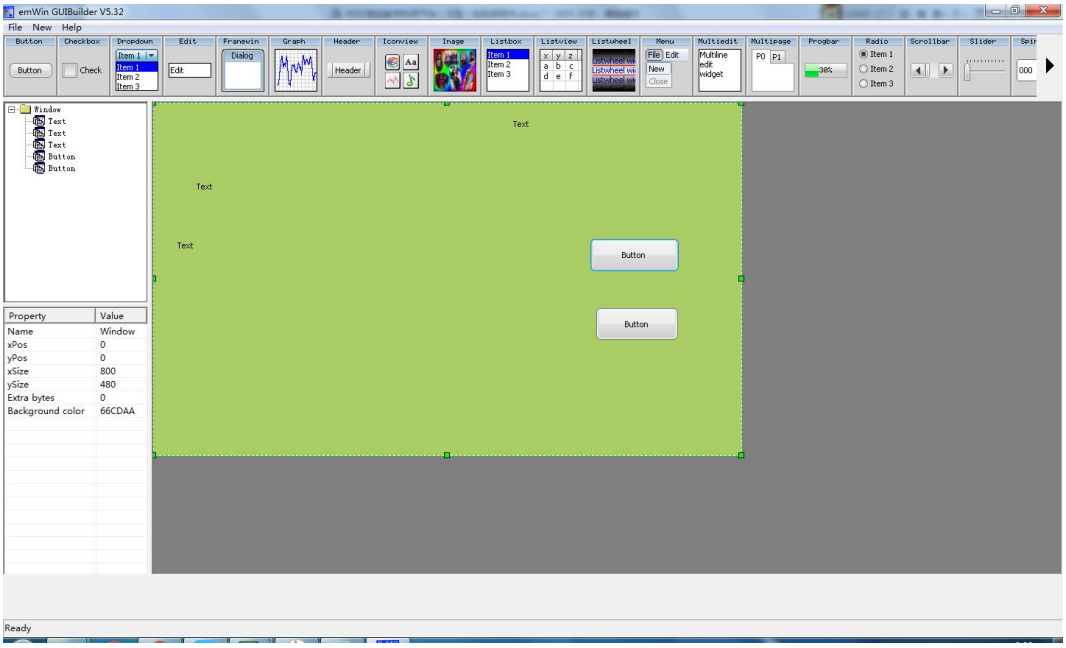

图 8.2 UI 界面

在 GUIBuilder 上完成设计，到 keil 中只需要给控件设置一下中文字体。

❖ 编码

执行部件 具体命令请阅读“执行模块串口通讯协议.pdf” 文档。

程序方面非常简单，触发按键按下消息则发送对应指令，指令由 执行部件 自动应答。

```
        case GUI_ID_BUTTON0:
            switch(NCode)
            {
            case WM_NOTIFICATION_CLICKED:
                for(i  = 0; i < 7; i++)
                {
                    BSP_ST16C554_CS2B_Write(dakai[i]);
                    BSP_Delay_ms(1);
                }
                break;
            case WM_NOTIFICATION_RELEASED:
                break;
            case WM_NOTIFICATION_MOVED_OUT:
                break;
            }
            break;
        case GUI_ID_BUTTON1:
            switch(NCode)
            {
```

```
            case WM_NOTIFICATION_CLICKED:
               for(i  = 0; i < 7; i++)
               {
                  BSP_ST16C554_CS2B_Write(guanbi[i]);
                  BSP_Delay_ms(1);
               }
               break;
            case WM_NOTIFICATION_RELEASED:
               break;
            case WM_NOTIFICATION_MOVED_OUT:
               break;
            }
            break;
```