

MUR-100 上位机库函数及演示软件使用说明

一. USB 驱动程序的安装

USB 驱动安装文件在“USB 驱动”目录下，里面分别有 WIN98 和 WIN2000_xp 的驱动安装文件。

在安装驱动程序之前，首先把读卡器连接到主机的 USB 口上，这时系统会自动提示监测到新硬件，要求安装这种硬件的驱动程序，下面具体解释不同系统的安装步骤：

1. 对于 WIN98

第一次连接到 USB 口时出现的提示框如下：



(1) 点击下一步，出现下图：



(2) 选择第一个选项, 然后点击下一步, 出现下图:



(3) 选择指定位置选项, 点击浏览按钮, 从中选择“USB 驱动安装\win98\drivers”目录, 点击下一步, 出现下图:



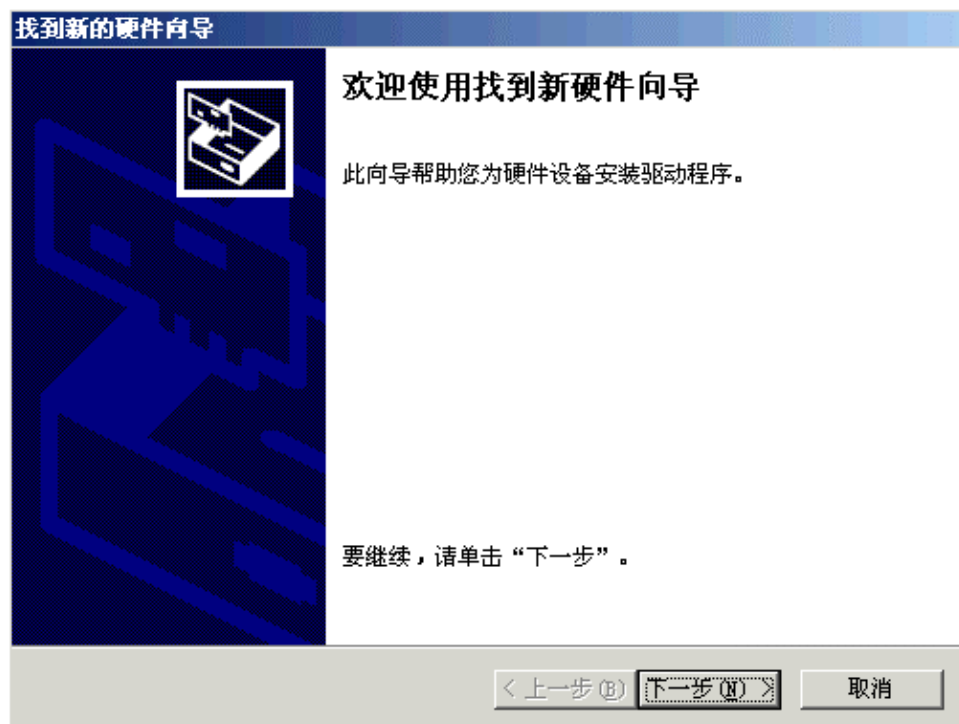
(4) 点击下一步，最后出现完成对话框：



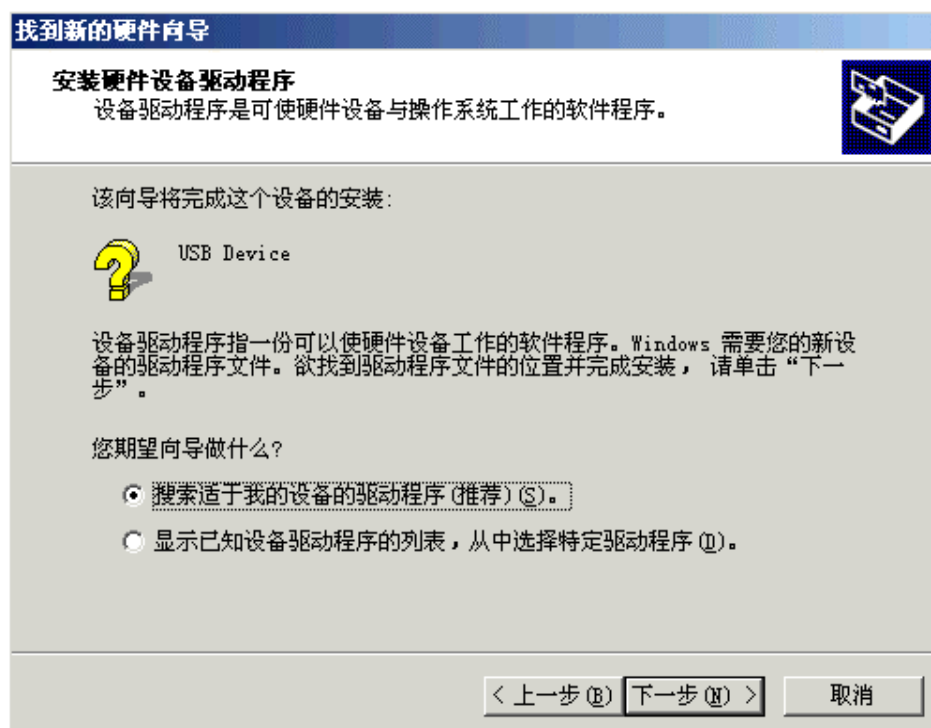
(5) 点击完成按钮就完成了对 USB 驱动的安装。

2. 对于 WIN2000/XP

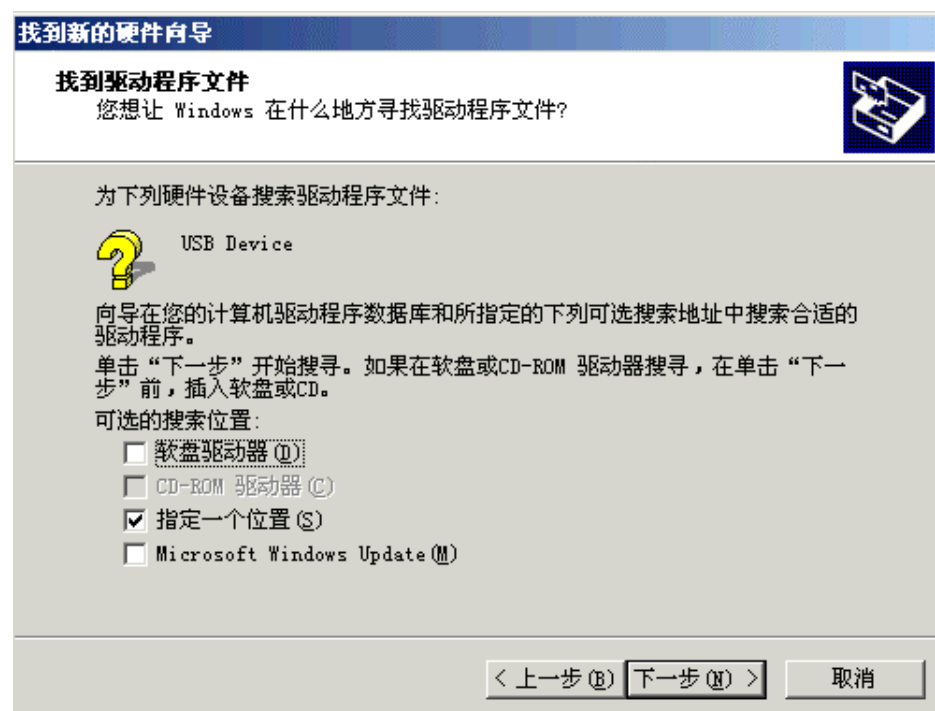
第一次连接到 USB 口时出现的提示框如下：



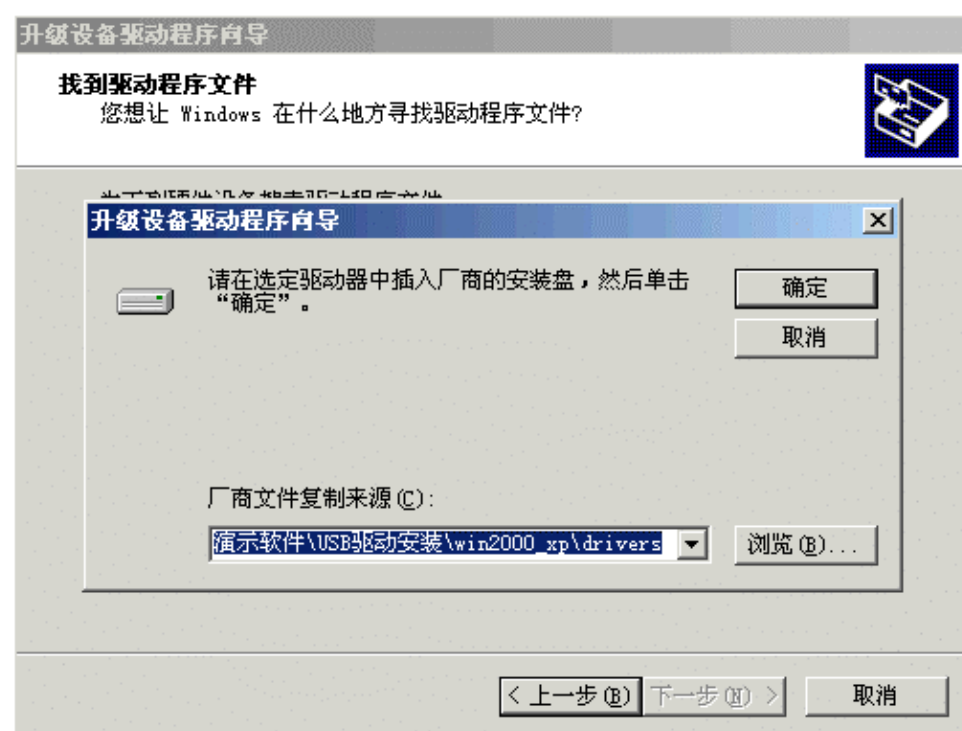
(1) 点击下一步，出现下图：



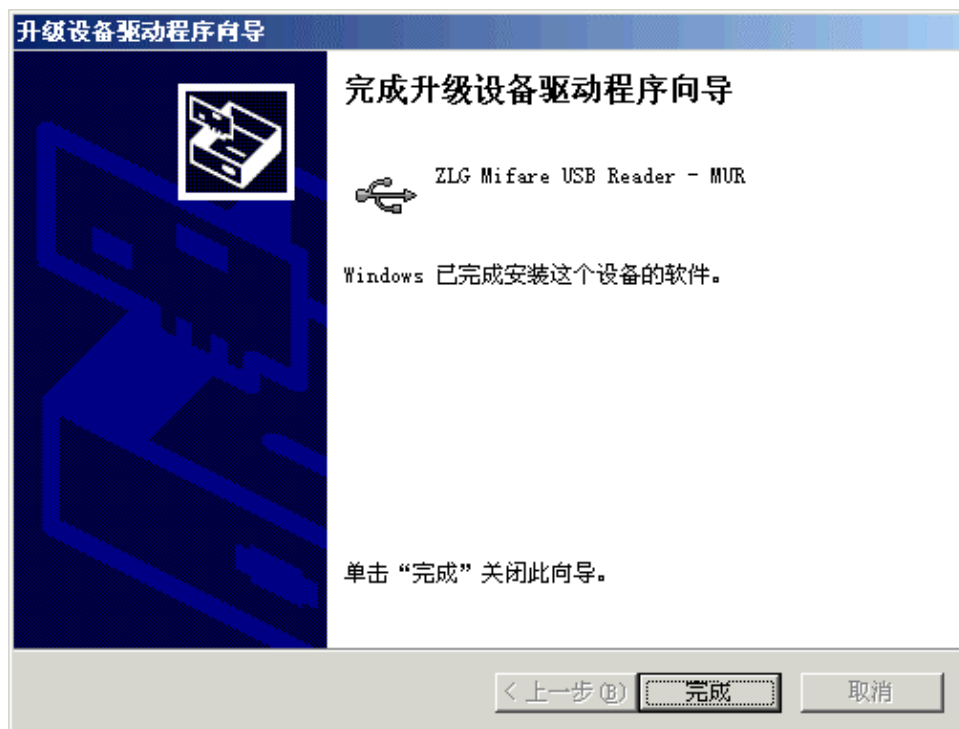
(2) 选择第一个选项，然后点击下一步，出现下图：



(3) 选择指定一个位置选项，点击下一步，出现下图：



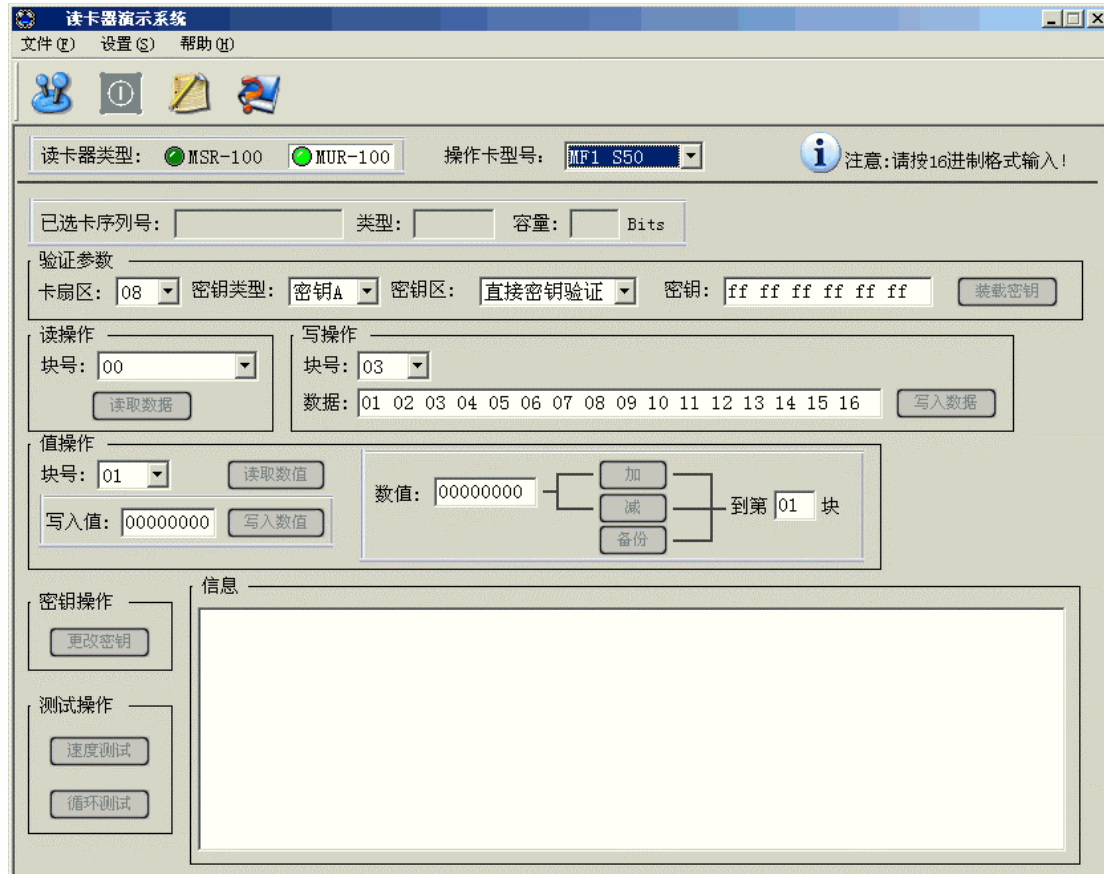
(4) 点击浏览按钮，从中选择“USB 驱动安装\win2000_xp\drivers”目录，然后点确定按钮，最后出现完成对话框：



(5) 点击完成按钮就完成了对 USB 驱动的安装。

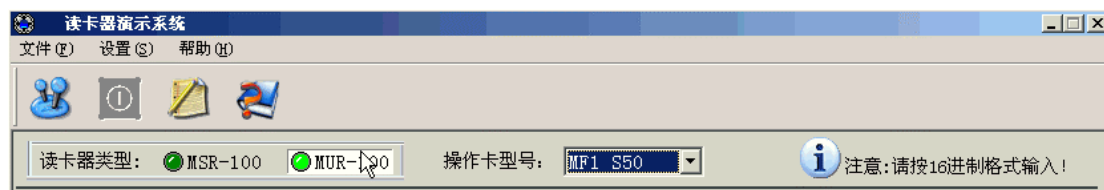
二. 演示软件的使用

开始进入系统时界面如下图：



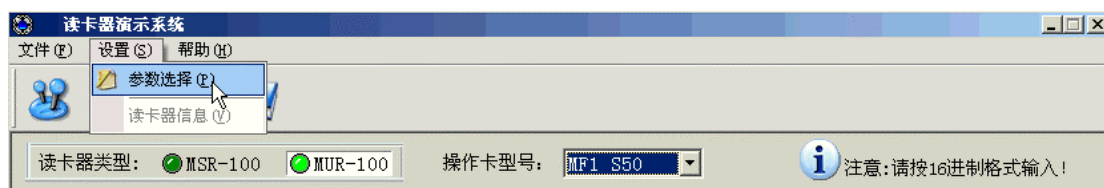
1. 选择要进行操作的读卡器类型

开始进行操作前，首先选择所要操作的读卡器类型（这里选择 MUR-100）：

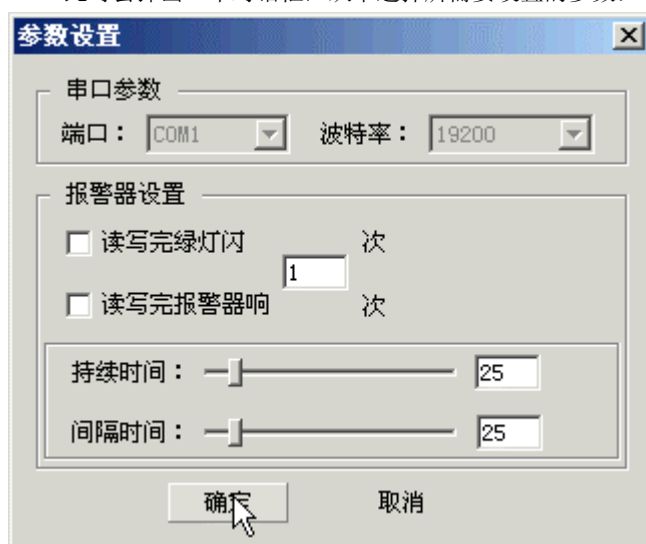


2. 设置连接参数

接下来从设置菜单中选择参数选择选项：

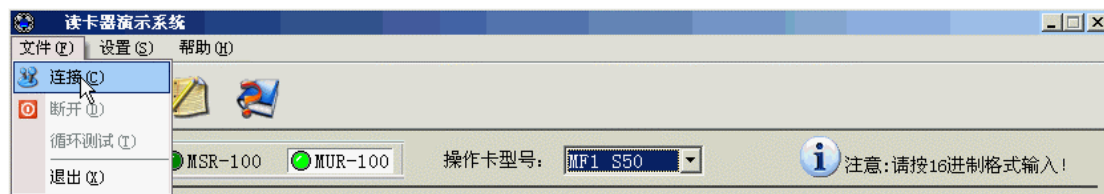


此时会弹出一个对话框，从中选择所需要设置的参数：

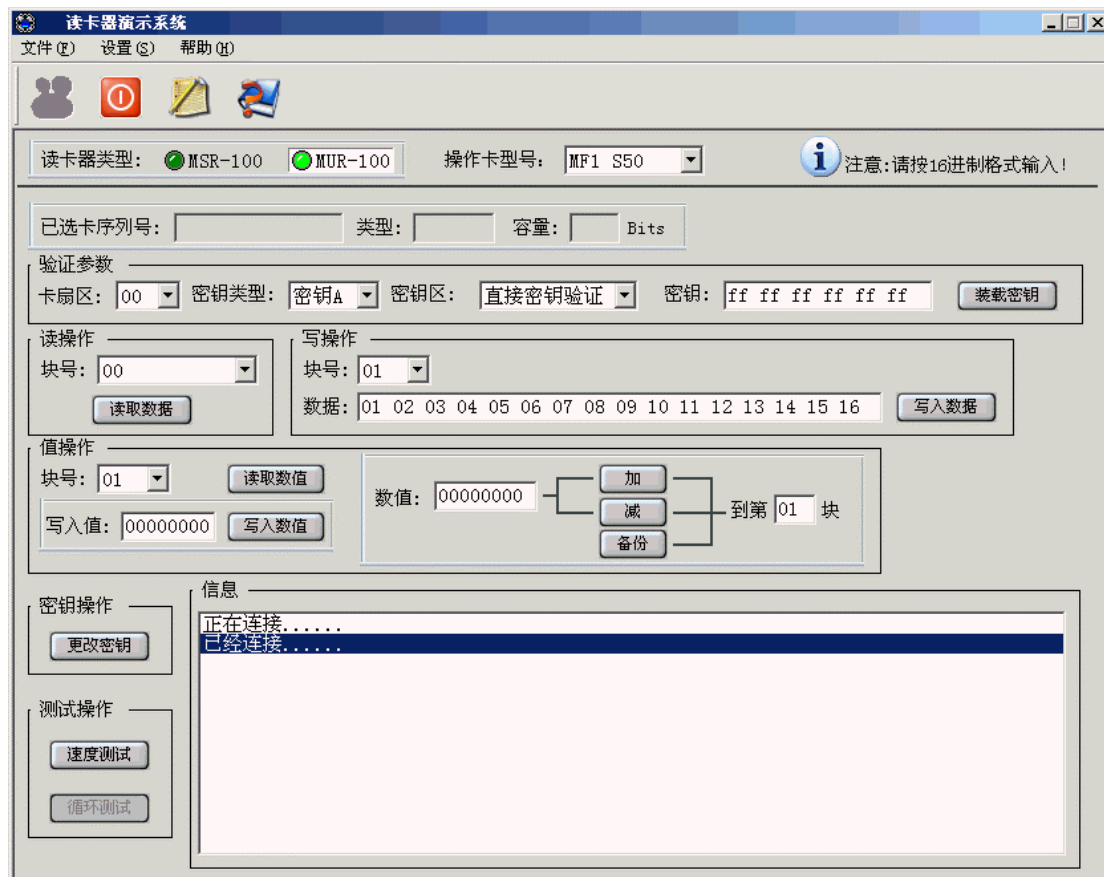


3. 连接

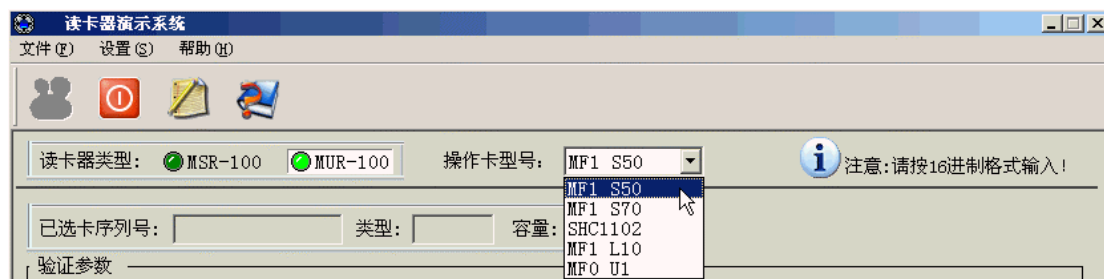
接着从文件菜单中选择连接选项进行连接：



连接上后系统界面如下图：



此时你要选择你想操作的卡型号:



3.1 选择 MF1 S50/70 卡

接下来的操作有两个分支:

3.1.1 手工操作

此时的操作界面如上图, 只要你选择好要进行操作的各项参数, 就可以点击读取数据, 写入数据, 读取数值, 写入数值和加减还有装载密钥以及更改密钥按钮进行操作了 (注: 你可以直接双击左边树列表框中的某个数据块来直接选择此数据块)。这里有必要详细说明一下装载密钥以及更改密钥的功能:

3.1.1.1 装载密钥

点击此按钮后会弹出一个装载密钥对话框:

密钥区	密钥A	密钥B
0	ff ff ff ff ff ff	ff ff ff ff ff ff
1	ff ff ff ff ff ff	ff ff ff ff ff ff
2	ff ff ff ff ff ff	ff ff ff ff ff ff
3	ff ff ff ff ff ff	ff ff ff ff ff ff
4	ff ff ff ff ff ff	ff ff ff ff ff ff
5	ff ff ff ff ff ff	ff ff ff ff ff ff
6	ff ff ff ff ff ff	ff ff ff ff ff ff
7	ff ff ff ff ff ff	ff ff ff ff ff ff
8	ff ff ff ff ff ff	ff ff ff ff ff ff
9	ff ff ff ff ff ff	ff ff ff ff ff ff
0A	ff ff ff ff ff ff	ff ff ff ff ff ff
0B	ff ff ff ff ff ff	ff ff ff ff ff ff
0C	ff ff ff ff ff ff	ff ff ff ff ff ff
0D	ff ff ff ff ff ff	ff ff ff ff ff ff
0E	ff ff ff ff ff ff	ff ff ff ff ff ff
0F	ff ff ff ff ff ff	ff ff ff ff ff ff

全部装载密钥A 全部装载密钥B

在其中你可以进行装载密钥操作（装载密钥就是预先把密钥装载到读卡器里面，在以后要验证卡密钥的时候就可以调用 RC500USB_authentication2 这个函数来进行验证密钥操作）。

3.1.1.2 更改密钥

点击此按钮会弹出一个更改密钥对话框：

更改密钥

验证参数
卡扇区: 00 密钥类型: 密钥A 密钥区: 直接密码验证 密钥: ff ff ff ff ff ff

说明: 每个扇区的最后一块为密钥块, 该块的前6个字节为密钥A, 接下来的4个字节为访问条件, 最后6个字节为密钥B。

原始参数
原密钥A: 旧访问条件: 原密钥B: 读取原始参数

更改密钥
新密钥A: ff ff ff ff ff ff 新访问条件: ff 07 80 69 新密钥B: ff ff ff ff ff ff 更改

数据块 (即在本扇区中除最后一块密钥块以外的其它块) 的访问条件

第0块	第1块	第2块
选定	C1 C2 C3	读 (Read) 写 (Write) 增加值 (Increment) 减少值 (Decrement), 传输 (Transfer), 恢复 (Restore)
<input checked="" type="radio"/>	0 0 0	密钥A B 密钥A B 密钥A B 密钥A B
<input type="radio"/>	0 1 0	密钥A B 无 无 无
<input type="radio"/>	1 0 0	密钥A B 密钥B 无 无
<input type="radio"/>	1 1 0	密钥A B 密钥B 密钥B 密钥A B
<input type="radio"/>	0 0 1	密钥A B 无 无 密钥A B
<input type="radio"/>	0 1 1	密钥B 密钥B 无 无
<input type="radio"/>	1 0 1	密钥B 无 无 无
<input type="radio"/>	1 1 1	无 无 无 无

(注: 在数据块的访问条件表格中的“密钥A|B”表示用密钥A或密钥B验证此扇区后, 可对此块进行该列对应的操作, 比如选择C1 C2 C3=0 0 0的话, 由于在列“读 (Read)”中对应的值为“密钥A|B”, 则表示无论用密钥A还是B验证后都可以对此块进行读取数据操作, 同样如果值为“密钥A”或“密钥B”的话, 则表示只有用密钥A或密钥B验证后才能对此块进行读取数据操作, 而如果值为“无”的话, 则表示无论用密钥A还是B验证都不能从此块中读出数据; 在密钥块的访问条件表格中的格式和数据块的访问条件表格一样, 比如选择C1 C2 C3=0 0 0的话, 由于在列“密钥A-写”中的值为“密钥A”, 则表示用密钥A进行验证后可更改密钥A, 而列“密钥A-读”中的值为“无”表示无论用密钥A还是密钥B进行验证后都不能把密钥A从卡中读取出来。)

密钥块 (即本扇区的最后一块) 的访问条件

注意: 当密钥B能被读出来时, 用密钥B验证后不能进行数据操作, 即不能对数据块进行读写。

选定	C1 C2 C3	密钥A		存取控制位		密钥B	
		读	写	读	写	读	写
<input type="radio"/>	0 0 0	无	密钥A	密钥A	无	密钥A	密钥A
<input type="radio"/>	0 1 0	无	无	密钥A	无	密钥A	无
<input type="radio"/>	1 0 0	无	密钥B	密钥A B	无	无	密钥B
<input type="radio"/>	1 1 0	无	无	密钥A B	无	无	无
<input checked="" type="radio"/>	0 0 1	无	密钥A	密钥A	密钥A	密钥A	密钥A
<input type="radio"/>	0 1 1	无	密钥B	密钥A B	密钥B	无	密钥B
<input type="radio"/>	1 0 1	无	无	密钥A B	密钥B	无	无
<input type="radio"/>	1 1 1	无	无	密钥A B	无	无	无

当你设置好验证参数后, 点击“读取原始参数”按钮就会把卡密钥区的所有信息读取出来, 此时界面如下:

更改密钥

验证参数
卡扇区: 05 密钥类型: 密钥A 密钥区: 直接密码验证 密钥: ff ff ff ff ff ff

说明: 每个扇区的最后一块为密钥块, 该块的前6个字节为密钥A, 接下来的4个字节为访问条件, 最后6个字节为密钥B。

原始参数
原密钥A: 00 00 00 00 00 00 旧访问条件: fe 1e 10 69 原密钥B: ff ff ff ff ff ff 读取原始参数

更改密钥
新密钥A: ff ff ff ff ff ff 新访问条件: ff 07 80 69 新密钥B: ff ff ff ff ff ff 更改

数据块 (即在本扇区中除最后一块密钥块以外的其它块) 的访问条件

第0块	第1块	第2块
选定	C1 C2 C3	读 (Read) 写 (Write) 增加值 (Increment) 减少值 (Decrement), 传输 (Transfer), 恢复 (Restore)
<input checked="" type="radio"/>	0 0 0	密钥A B 密钥A B 密钥A B 密钥A B
<input type="radio"/>	0 1 0	密钥A B 无 无 无
<input type="radio"/>	1 0 0	密钥A B 密钥B 无 无
<input type="radio"/>	1 1 0	密钥A B 密钥B 密钥B 密钥A B
<input type="radio"/>	0 0 1	密钥A B 无 无 密钥A B
<input type="radio"/>	0 1 1	密钥B 密钥B 无 无
<input checked="" type="radio"/>	1 0 1	密钥B 无 无 无
<input type="radio"/>	1 1 1	无 无 无 无

(注: 在数据块的访问条件表格中的“密钥A|B”表示用密钥A或密钥B验证此扇区后, 可对此块进行该列对应的操作, 比如选择C1 C2 C3=0 0 0的话, 由于在列“读 (Read)”中对应的值为“密钥A|B”, 则表示无论用密钥A还是B验证后都可以对此块进行读取数据操作, 同样如果值为“密钥A”或“密钥B”的话, 则表示只有用密钥A或密钥B验证后才能对此块进行读取数据操作, 而如果值为“无”的话, 则表示无论用密钥A还是B验证都不能从此块中读出数据; 在密钥块的访问条件表格中的格式和数据块的访问条件表格一样, 比如选择C1 C2 C3=0 0 0的话, 由于在列“密钥A-写”中的值为“密钥A”, 则表示用密钥A进行验证后可更改密钥A, 而列“密钥A-读”中的值为“无”表示无论用密钥A还是密钥B进行验证后都不能把密钥A从卡中读取出来。)

密钥块 (即本扇区的最后一块) 的访问条件

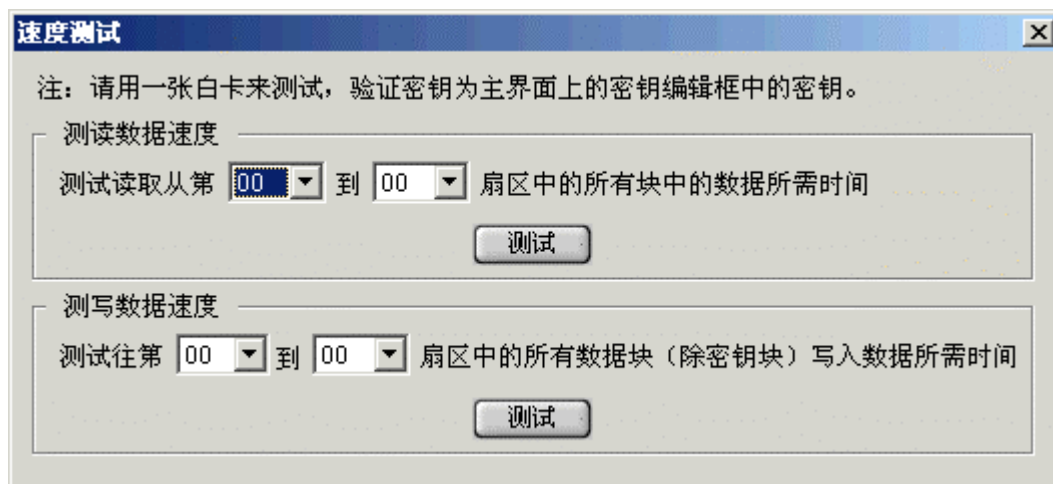
注意: 当密钥B能被读出来时, 用密钥B验证后不能进行数据操作, 即不能对数据块进行读写。

选定	C1 C2 C3	密钥A		存取控制位		密钥B	
		读	写	读	写	读	写
<input checked="" type="radio"/>	0 0 0	无	密钥A	密钥A	无	密钥A	密钥A
<input type="radio"/>	0 1 0	无	无	密钥A	无	密钥A	无
<input type="radio"/>	1 0 0	无	密钥B	密钥A B	无	无	密钥B
<input type="radio"/>	1 1 0	无	无	密钥A B	无	无	无
<input checked="" type="radio"/>	0 0 1	无	密钥A	密钥A	密钥A	密钥A	密钥A
<input type="radio"/>	0 1 1	无	密钥B	密钥A B	密钥B	无	密钥B
<input type="radio"/>	1 0 1	无	无	密钥A B	密钥B	无	无
<input type="radio"/>	1 1 1	无	无	密钥A B	无	无	无

在对话框中的两个表格中颜色变为反白的一行表示为对应的卡的原始访问条件, 而被选中但是颜色正常的一行表示为卡的新的访问条件。在设置好新密钥 A 和 B 以及新访问条件后就可以点击“更改”按钮进行修改。

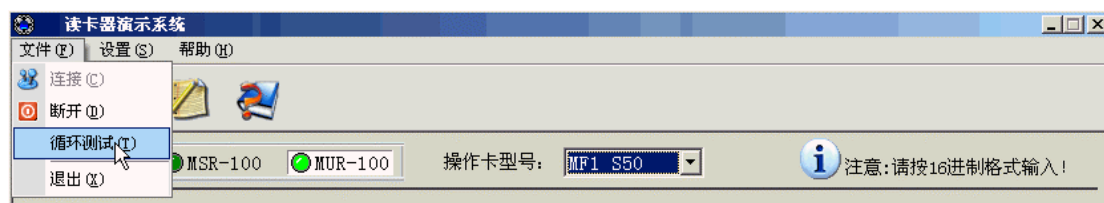
3.1.1.3 速度测试

此时点击速度测试按钮可以进行读写卡速度测试:

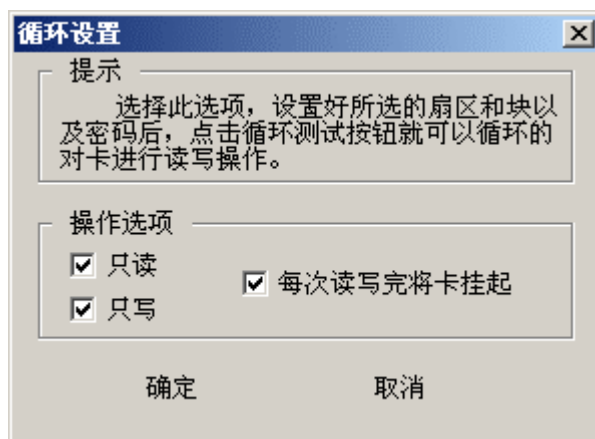


3.1.2 循环测试

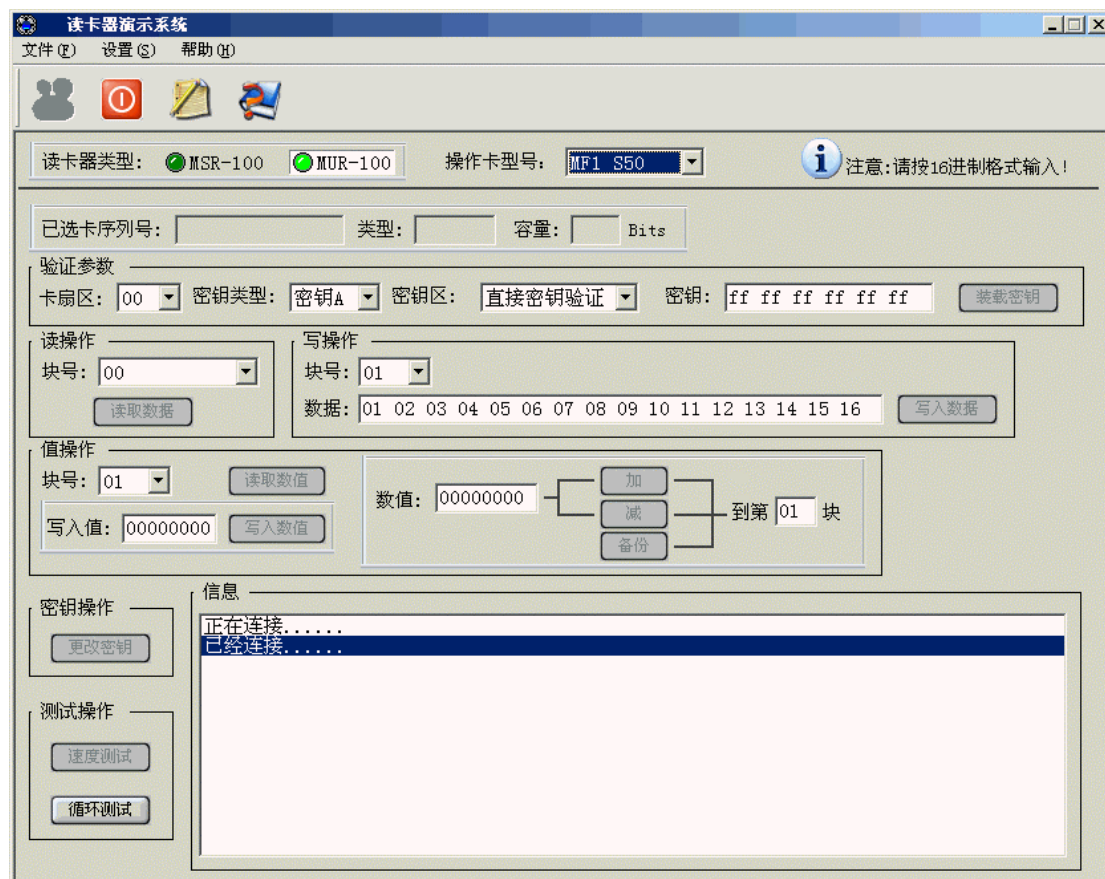
要进行循环操作时，首先从文件菜单中选择循环测试按钮：



此时弹出一个对话框：



从中你可以选择你循环测试时所要进行的操作（“每次读写完将卡挂起”指的是当有卡进入射频感应区的时候，就开始对卡进行读写操作。每次只对同一张卡进行一次读写操作，要进行第二次操作的话，得把卡从感应区内拿开，然后再放进去才行。），点确定后界面如下图：



此时你可以点击循环测试按钮进行测试（注意：在这之前首先得确保你已经选择好要进行操作的各项参数，在循环操作中已经设置好的各项参数都不能改变，要改变只能先中止循环测试再进行更改），你只要把卡放到感应区就能自动进行指定的操作。要中止循环测试则只需再次选择文件菜单中的循环测试选项即可（循环测试选项前打勾表示已经选择，未打勾表示没有选择）。

3.2 选择 SHC1102 卡

此时会弹出一个专门操作 SHC1102 卡的对话框：

SHC1102

已选卡序列号: 类型: 容量: Bits

密钥:

第0块: <input type="text" value="ff ff ff ff"/>		第8块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第1块: <input type="text" value="ff ff ff ff"/>		第9块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第2块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第10块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第3块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第11块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第4块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第12块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第5块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第13块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第6块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第14块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>
第7块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>	第15块: <input type="text" value="ff ff ff ff"/>	<input type="button" value="写入"/>

在密钥编辑框中输入卡的密钥后，就可以对卡进行数据的读取和写入操作，点击“更改密钥”按钮则会弹出一个更改密钥对话框来进行更改密钥操作：

更改密钥

原密钥:

更改为:

注：卡内第8块为密钥块

点击速度测试按钮可以进行读写卡速度测试：

速度测试

注：请用一张白卡来测试，验证密钥为主界面上的密钥编辑框中的密钥。

测读数据速度

测试读取从第 到 块中的数据所需时间

测写数据速度

测试往第 到 块写入数据所需时间

3.3 选择 MF1 L10 卡

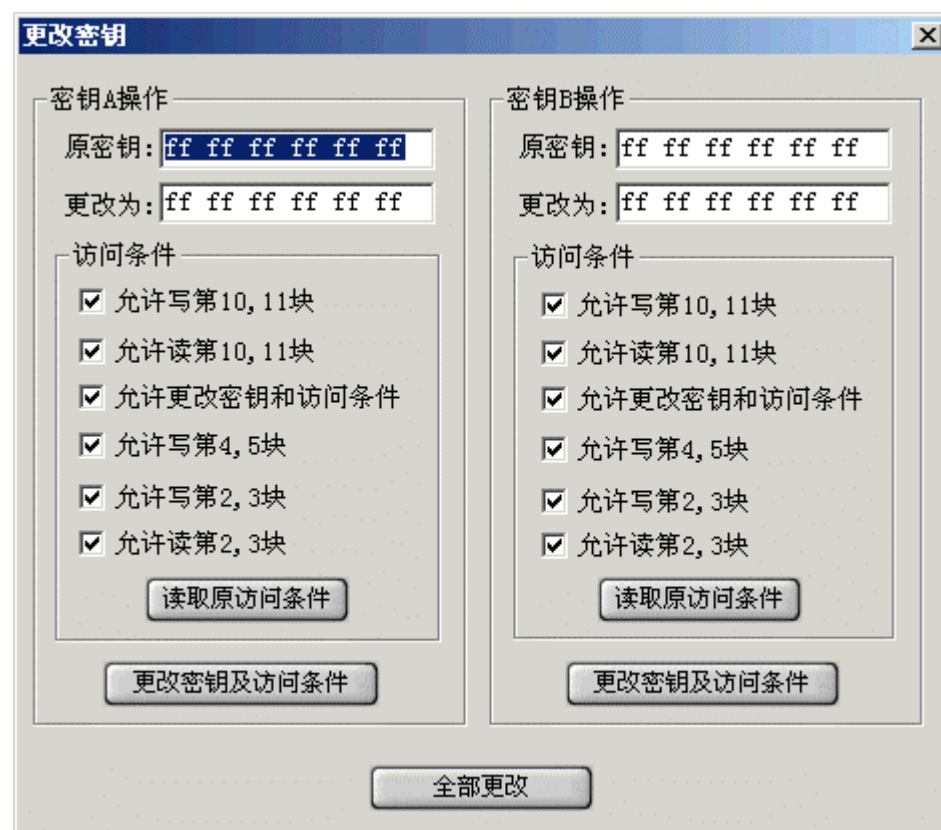
此时会弹出一个专门操作 MF1 L10 卡的对话框：



The dialog box is titled "MF1 L10". It contains the following fields and controls:

- 已选卡序列号: [Text Field]
- 类型: [Text Field]
- 容量: [Text Field] Bits
- 密钥类型: 密钥A (Dropdown)
- 密钥: ff ff ff ff ff ff (Text Field)
- A table with 12 rows (第0块 to 第11块) for data entry. Each row has a data field (e.g., ff ff ff ff) and a "写入" (Write) button.
- 数值操作 section:
 - 第4块 减少 数值:0x 0001
 - 第5块 减少 数值:0x 0001
- Buttons at the bottom: 读取所有块数据, 往所有块写入数据, 更改密钥.

在设置好用来验证的密钥类型和密钥后，就可以对卡进行数据的读写操作了。点击其中的“更改密钥”按钮弹出更改密钥对话框来进行更改密钥操作：



The dialog box is titled "更改密钥" (Change Key). It is divided into two main sections: 密钥A操作 and 密钥B操作.

- 密钥A操作:**
 - 原密钥: ff ff ff ff ff ff
 - 更改为: ff ff ff ff ff ff
 - 访问条件 (Access Conditions):
 - ☒ 允许写第10, 11块
 - ☒ 允许读第10, 11块
 - ☒ 允许更改密钥和访问条件
 - ☒ 允许写第4, 5块
 - ☒ 允许写第2, 3块
 - ☒ 允许读第2, 3块
 - Buttons: 读取原访问条件, 更改密钥及访问条件.
- 密钥B操作:**
 - 原密钥: ff ff ff ff ff ff
 - 更改为: ff ff ff ff ff ff
 - 访问条件 (Access Conditions):
 - ☒ 允许写第10, 11块
 - ☒ 允许读第10, 11块
 - ☒ 允许更改密钥和访问条件
 - ☒ 允许写第4, 5块
 - ☒ 允许写第2, 3块
 - ☒ 允许读第2, 3块
 - Buttons: 读取原访问条件, 更改密钥及访问条件.
- Bottom button: 全部更改 (All Change).

在更改之前最好先把原访问条件读出来。

3.4 选择 MF0 U1 卡

此时会弹出一个专门操作 MF0 U1 卡的对话框：

The MF0U1 dialog box is titled "MF0U1" and contains fields for "已选卡序列号:" (Selected Card Serial Number), "类型:" (Type), and "容量:" (Capacity) in Bits. Below these fields is a table with 16 rows, each representing a memory block (第0块 to 第15块). Each row has a text input field for data (initially showing "ff ff ff ff") and a "写入" (Write) button. At the bottom of the dialog are three buttons: "读取所有块数据" (Read all block data), "往所有块写入数据" (Write data to all blocks), and "更改访问条件" (Change access conditions).

在其中可以直接对卡进行数据的读写操作。点击“更改访问条件”按钮弹出更改访问条件对话框来进行更改卡的访问条件操作：

The "更改访问条件" (Change Access Conditions) dialog box contains three main sections, each with a checkbox for locking access conditions and a list of checkboxes for OTP write permissions for specific blocks:

- ☐ 锁定块3访问条件不可更改
 - ☐ 块3 (OTP) 可写
- ☐ 锁定块4到块9访问条件不可更改
 - ☐ 块4 (OTP) 可写
 - ☐ 块5 (OTP) 可写
 - ☐ 块6 (OTP) 可写
 - ☐ 块7 (OTP) 可写
 - ☐ 块8 (OTP) 可写
 - ☐ 块9 (OTP) 可写
- ☐ 锁定块10到块15访问条件不可更改
 - ☐ 块10 (OTP) 可写
 - ☐ 块11 (OTP) 可写
 - ☐ 块12 (OTP) 可写
 - ☐ 块13 (OTP) 可写
 - ☐ 块14 (OTP) 可写
 - ☐ 块15 (OTP) 可写

At the bottom are two buttons: "读取原访问条件" (Read original access conditions) and "更改访问条件" (Change access conditions).

在更改访问条件之前最好先读出原访问条件。

三. 库函数说明及其使用

1. 库函数说明

1.1 unsigned char __stdcall RC500USB_init()

入口参数：

无

函数功能：此函数的功能是打开 USB。

1.2 void __stdcall RC500USB_exit()

入口参数:

无

函数功能: 此函数的功能是关闭 USB。

1.3 unsigned char __stdcall RC500USB_request(unsigned char mode, unsigned short &tagtype)

入口参数:

mode:请求类型

mode=0:请求天线范围内 IDLE 状态的卡 (HALT 状态的除外)

mode=1: 请求天线范围内的所有卡

tagtype:返回的目标类型

函数功能: 此函数发送 Request 命令, 检查在有效范围内是否有卡的存在。

返回值: 0 表示成功, 否则返回错误码。

1.4 unsigned char __stdcall RC500USB_anticoll(unsigned char bcnt,unsigned long &snr)

入口参数:

bcnt:为预选卡所分配的位的个数, 通常 Bcnt=0

snr:返回卡的序号

函数功能: 此函数开始防冲突操作。

返回值: 0 表示成功, 否则返回错误码。

1.5 unsigned char __stdcall RC500USB_anticoll2(unsigned char encoll,unsigned char bcnt,unsigned long &snr)

入口参数:

encoll:若为 1, 则使能多张卡进入天线区; 若为 0, 则不允许多张卡进入, 此时返回错误 MI_COLLERR

bcnt:为预选卡所分配的位的个数, 通常 Bcnt=0

snr:返回卡的序号

函数功能: 此函数开始防冲突操作。

返回值: 0 表示成功, 否则返回错误码。

1.6 unsigned char __stdcall RC500USB_select(unsigned long snr,unsigned char &size)

入口参数:

snr:卡的序号

size:返回卡的容量

函数功能: 此函数选择某一个序号的卡。

返回值: 0 表示成功, 否则返回错误码。

1.7 unsigned char __stdcall RC500USB_authentication(unsigned char mode,unsigned char secnr)

入口参数:

mode:密钥类型,密钥 A:0x00,密钥 B:0x04

secnr:要验证的卡扇区号

函数功能：此函数将 RC500 中的密钥与卡中的密钥进行验证。在进行对卡片的读写操作之前，必须成功执行此指令。在系统初始化时，主控设备将把每个扇区的密码加载到读卡器中。

返回值：0 表示成功，否则返回错误码。

1.8 unsigned char __stdcall RC500USB_authentication2(unsigned char mode,unsigned char secnr,unsigned char keynr)

入口参数：

mode:密钥类型,密钥 A:0x00,密钥 B:0x04

secnr:要验证的卡扇区号

keynr:用于证实的密钥区号

函数功能：此函数将 RC500 中的密钥与卡中的密钥进行验证。在进行对卡片的读写操作之前，必须成功执行此指令。在系统初始化时，主控设备将把每个扇区的密码加载到读卡器中。

返回值：0 表示成功，否则返回错误码。

1.9 unsigned char __stdcall RC500USB_authkey(unsigned char mode,unsigned char *key,unsigned char secnr)

入口参数：

mode:密钥类型,密钥 A:0x00,密钥 B:0x04

key:存储密钥缓冲区

secnr:要验证的卡扇区号

函数功能：在对卡进行读、写、加、减等操作前，必须对卡进行验证。若卡中的密钥与所传输的密码相匹配。则证实成功，函数将返回 MI_OK

返回值：0 表示成功，否则返回错误码。

1.10 unsigned char __stdcall RC500USB_halt(void)

入口参数：

无

函数功能：此函数使被选择的卡产生一个暂停，即使之处于 Halt 模式。处于 Halt 模式的卡，只能用 ALL 方式进行选择，即 RC500USB_request 函数中的 mode 参数值设为 1。

返回值：0 表示成功，否则返回错误码。

1.11 unsigned char __stdcall RC500USB_read(unsigned char addr,unsigned char *data)

入口参数：

addr:块号(0x00--0x3f)

data:块数据，长度 16 个字节

函数功能：此函数功能是读取卡的一个块的数据。（注：对于 Mifare Ultra Light 卡，由于其每个块的大小只有 4 个字节，则此时将会读出相邻 4 个块的数据，比如 addr=0，则读出块 0，1，2，3 的数据，addr=14,则读出块 14，15，0，1 的数据）

返回值：0 表示成功，否则返回错误码。

1.12 unsigned char __stdcall RC500USB_write(unsigned char addr,unsigned char *data)

入口参数：

addr:块号(0x00--0x3f)

data:块数据，长度 16 个字节

函数功能：此函数功能是对指定的块进行整块写操作。

返回值：0 表示成功，否则返回错误码。

1.13 unsigned char __stdcall RC500USB_writeval(unsigned char addr,long value)

入口参数：

addr: 块号(0x00--0x3f)

value: 要写入的值

函数功能：此函数往一个块中写入数值。

返回值：0 表示成功，否则返回错误码。

1.14 unsigned char __stdcall RC500USB_readval(unsigned char addr,long &value)

入口参数：

addr: 块号(0x00--0x3f)

value: 存储要读出的值

函数功能：此函数读出一个块中的数值。

返回值：0 表示成功，否则返回错误码。

1.15 unsigned char __stdcall RC500USB_value(unsigned char mode,unsigned char addr,long &value,unsigned char trans_addr)

入口参数：

mode:模式（1 字节，加 C1H，减 C0H，复制 C2H）

addr:块号(0x00--0x3f)

value:要操作的值

trans_addr:传输块地址

函数功能：此函数功能是对一个块进行值操作，并把结果存到 trans_addr 中。

返回值：0 表示成功，否则返回错误码。

1.16 unsigned char __stdcall RC500USB_load_key(unsigned char mode,unsigned char secnr,unsigned char *key)

入口参数：

mode:密钥类型,密钥 A:0x00,密钥 B:0x04

secnr:要载入的读卡器中的密钥区号

key:密码，长度 6 个字节

函数功能：这条指令的功能是将 RC500 中的密钥与卡中的密钥进行验证。在进行对卡片的读写操作之前，必须成功执行此指令。在系统初始化时，主控设备将把每个扇区的密码加载到读卡器中。

返回值：0 表示成功，否则返回错误码。

1.17 unsigned char __stdcall RC500USB_reset(unsigned char msec)

入口参数：

msec:射频电路关闭时间（以毫秒为单位）

函数功能：该函数使射频电路关闭所规定的时间，若 msec=0，射频电路部分将一直处于关闭状态，一直到下一个 Request 命令到来。关闭射频能使天线内的所有卡复位。

返回值：0 表示成功，否则返回错误码。

1.18 unsigned char __stdcall RC500USB_close(void)

入口参数: 无

函数功能: 此函数将 RC500 的复位管脚置为高电平, 关闭 RC500, 使之电流最小。若要重新启动则需调用 Config()。

返回值: 0 表示成功, 否则返回错误码。

1.19 unsigned char __stdcall RC500USB_config(void)

入口参数: 无

函数功能: 模块每次上电复位之后, 都必须首先调用此函数对模块进行初始化, 才能进行进一步的操作。

返回值: 0 表示成功, 否则返回错误码。

1.20 unsigned char __stdcall RC500USB_get_info(unsigned char *info)

入口参数:

info: info[0]-info[4]为 RC500 的产品类型标识, 依次为 0x30,0x88,0xf8,0x00,0xXX

info[5]-info[8]为 RC500 的序列号

函数功能: 此函数返回一个包含有 RC500 的产品类型标识和序列号的数组。

返回值: 0 表示成功, 否则返回错误码。

1.21 unsigned char __stdcall RC500USB_set_control_bit()

入口参数: 无

函数功能: 此函数设置 MIFARE 读卡器中的控制位为高电平。

返回值: 0 表示成功, 否则返回错误码。

1.22 unsigned char __stdcall RC500USB_clr_control_bit()

入口参数: 无

函数功能: 此函数清除 MIFARE 读卡器中的控制位

返回值: 0 表示成功, 否则返回错误码。

1.23 unsigned char __stdcall RC500USB_buzzer(unsigned char contrl, unsigned char opentm, unsigned char closetm, unsigned char repcnt)

入口参数:

contrl: 控制字, 如下表相应位为 1 时该器件动作。

contrl^1 为绿灯, contrl^0 为蜂鸣器

opentm: 低电平持续时间, 取值 (0-255), 10ms 的分辨率

closetm: 高电平间隙时间, 取值 (0-255), 10ms 的分辨率

repent: 重复次数

函数功能: 此函数输出一驱动信号可驱动蜂鸣器和绿色发光管, 持续时间、间隙时间和重复次数可调。

返回值: 0 表示成功, 否则返回错误码。

1.24 unsigned char __stdcall RC500USB_read_E2(unsigned char addr, unsigned char length, unsigned char *data)

入口参数:

addr: 被读 RC500 内 EEPROM 首址, 必须小于 0x80

length: 被读数据长度

data: 读出数据缓冲区首址

函数功能: 此函数将 RC500 内 EEPROM 的数据读出。

返回值: 0 表示成功, 否则返回错误码。

1.25 unsigned char __stdcall RC500USB_write_E2(unsigned char addr,unsigned char length,unsigned char *data)

入口参数:

addr: RC500 内 EEPROM 的写入首址, 取值范围 (0x30-0x7E)

length: 被写数据长度

data: 写入数据缓冲区首址

函数功能: 此函数将数据写入 RC500 内 EEPROM 中。RC500 内 EEPROM 的 0x00-0x0F 为只读产品信息区, 0x10-0x2F 为启动寄存器初始化文件区, 最好不要改写, 0x80-0x1FF 为只读密钥区, 可用 LoadKey 写入。0x7F 为模块波特率参数存贮区, 最好不要改写。

返回值: 0 表示成功, 否则返回错误码。

1.26 unsigned char __stdcall RC500USB_authshc1102(unsigned char keyblock,unsigned char* key)

入口参数:

keyblock: 密码块地址,对于 SHC1102 为 8

key: 4 字节密码首址

函数功能: 此函数将参数中的密码与卡中密码块的密码进行比较若密码匹配则函数返回 0。

返回值: 0 表示成功, 否则返回错误码。

1.27 unsigned char __stdcall RC500USB_readshc1102(unsigned char block,unsigned char* data)

入口参数:

block: 所读块地址

data: 4 字节数据首址

函数功能: 此函数将卡中的数据块读出。

返回值: 0 表示成功, 否则返回错误码。

1.28 unsigned char __stdcall RC500USB_writeshc1102(unsigned char block,unsigned char* data)

入口参数:

block: 所写块地址

data: 4 字节数据首址

函数功能: 此函数将数据写入卡中。

返回值: 0 表示成功, 否则返回错误码。

1.29 void __stdcall RC500USB_GetDllVer(char *ver)

入口参数:

ver: 存储动态库版本号, 版本 1.4 表示为 “V1.4”

函数功能: 此函数读取动态库版本号。

返回值: 无。

1.30 unsigned char __stdcall RC500USB_CaseAnticoll(unsigned char bcnt,unsigned char code,unsigned long &snr)

入口参数:

bcnt: 为预选卡所分配的位的个数, 通常 Bcnt=0
code: 防碰撞层级编码一层 0x93, 二层 0x95, 三层 0x97
snr: 返回卡的序号

函数功能: 此函数开始防冲突操作。

返回值: 0 表示成功, 否则返回错误码。

1.31 unsigned char __stdcall RC500USB_CaseSelect(unsigned char code,unsigned long snr,unsigned char &sak)

入口参数:

code: 防碰撞层级编码一层 0x93, 二层 0x95, 三层 0x97
snr: 返回卡的序号
sak: 当 Select 命令返回值为 MI_OK 时,Sak 将返回主机,若 _Sak.2 为 1,则表示还有序列号的一部分未读出应进行下一层的选择否则选择结束

函数功能: 这个函数选择某一个序列号的卡。

返回值: 0 表示成功, 否则返回错误码。

1.32 unsigned char __stdcall RC500USB_ULwrite(unsigned char addr,unsigned char* data)

入口参数:

addr: 块号(0x02--0x0f)
data: 块数据, 长度 4 个字节

函数功能: 此函数功能是对 MF0 U1 卡中指定的块进行整块写操作。

返回值: 0 表示成功, 否则返回错误码。

1.33 unsigned char __stdcall RC500USB_ValueDebit(unsigned char mode,unsigned char addr,long val)

入口参数:

mode: 模式 (加 C1H, 减 C0H (Mifare Light 只支持减), 复制 C2H)
addr: 卡内块地址,对该块进行值操作
val: 要操作的值

函数功能: 此函数功能是对一个块进行值操作。

返回值: 0 表示成功, 否则返回错误码。

1.34 unsigned char __stdcall RC500USB_Lread(unsigned char addr,unsigned char *data)

入口参数:

addr: 卡内块地址
data: 存储读出的数据, 长度为 8

函数功能: 此函数读取 Mifare Light 卡中两个块的数据, 比如当 addr=0 时, 读取的是块 0 和块 1 的数据, 当 addr=1, 读取的还是块 0 和块 1 的数据; 而当 addr=2 时, 读取的是块 2 和块 3 的数据, 依此类推。

返回值: 0 表示成功, 否则返回错误码。

1.35 unsigned char __stdcall RC500USB_Lwrite(unsigned char addr,unsigned char *data)

入口参数:

addr: 卡内块地址
data: 要写入的块数据, 长度为 4

函数功能: 此函数往 Mifare Light 卡中指定块写入数据。

返回值: 0 表示成功, 否则返回错误码。

2. 库函数使用方法

2.1 调用动态库方法

首先, 把库函数文件都放到工作目录中。总共有四个文件: RC500USB.h, RC500USB.lib, RC500USB.dll, EasyD12_500.dll。

2.1.1 VC 隐式调用动态库的方法

- (1) 在.CPP 中包含 RC500USB.h 头文件;
- (2) 在工程文件中加入 RC500USB.lib 文件。

2.1.2 VC 显式调用动态库的方法

要显示调用动态库, 首先得调用 LoadLibrary 函数来加载 RC500USB.dll, 用完后调用 FreeLibrary 函数来卸载之。比如你要调用库中的 unsigned char __stdcall RC500USB_read(unsigned char addr, unsigned char *data) 这个函数, 则可以按照以下步骤来调用:

```
typedef unsigned char (CALLBACK* LPRC500USB_READ)(unsigned char, unsigned char*);  
.  
.  
.  
HANDLE m_hDLL = LoadLibrary("RC500USB");  
LPRC500USB_READ RC500USB_read;  
RC500USB_read = (LPRC500USB_READ)GetProcAddress(m_hDLL, "RC500USB_read");  
unsigned char addr=0;  
unsigned char recbuff[16];  
RC500USB_read(1, recbuff); //调用 RC500USB_read 函数  
.  
.  
.  
FreeLibrary(m_hDLL);
```

2.1.3 VB 调用动态库的方法

通过以下方法进行声明后就可以调用了。

语法:

```
[Public | Private] Declare Function name Lib "libname" [Alias "aliasname"] [(arglist)] [As type]
```

Declare 语句的语法包含下面部分:

Public (可选) 用于声明在所有模块中的所有过程都可以使用的函数。

Private (可选), 用于声明只能在包含该声明的模块中使用的函数。

Name (必选), 任何合法的函数名。动态链接库的入口处 (entry points) 区分大小写。

Libname (必选), 包含所声明的函数动态链接库名或代码资源名。

Alias (可选), 表示将被调用的函数在动态链接库 (DLL) 中还有另外的名称。当外部函数名与某个函数重名时, 就可以使用这个参数。当动态链接库的函数与同一范围内的公用变量、常数或任何其它过程的名称相同时, 也可以使用 Alias。如果该动态链接库函数中的某个字符不符合动态链接库的命名约定时, 也可以使用 Alias。

Aliasname (可选) 动态链接库。如果首字符不是数字符号 (#), 则 aliasname 是动态链接库中该函数入口处的名称。如果首字符是 (#), 则随后的字符必须指定该函数入口处的顺序号。

Arglist (可选), 代表调用该函数时需要传递参数的变量表。

Type (可选), Function 返回值的数据类型; 可以是 Byte、Boolean、Integer、Long、Currency、Single、Double、Decimal (目前尚不支持)、Date、String (只支持变长) 或 Variant, 用户定义类型, 或对象类型。

arglist 参数的语法如下:

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
```

部分描述:

Optional (可选), 表示参数不是必需的。如果使用该选项, 则 arglist 中的后续参数都必需是可选的, 而且必须都使用 Optional 关键字声明。如果使用了 ParamArray, 则任何参数都不能使用 Optional。

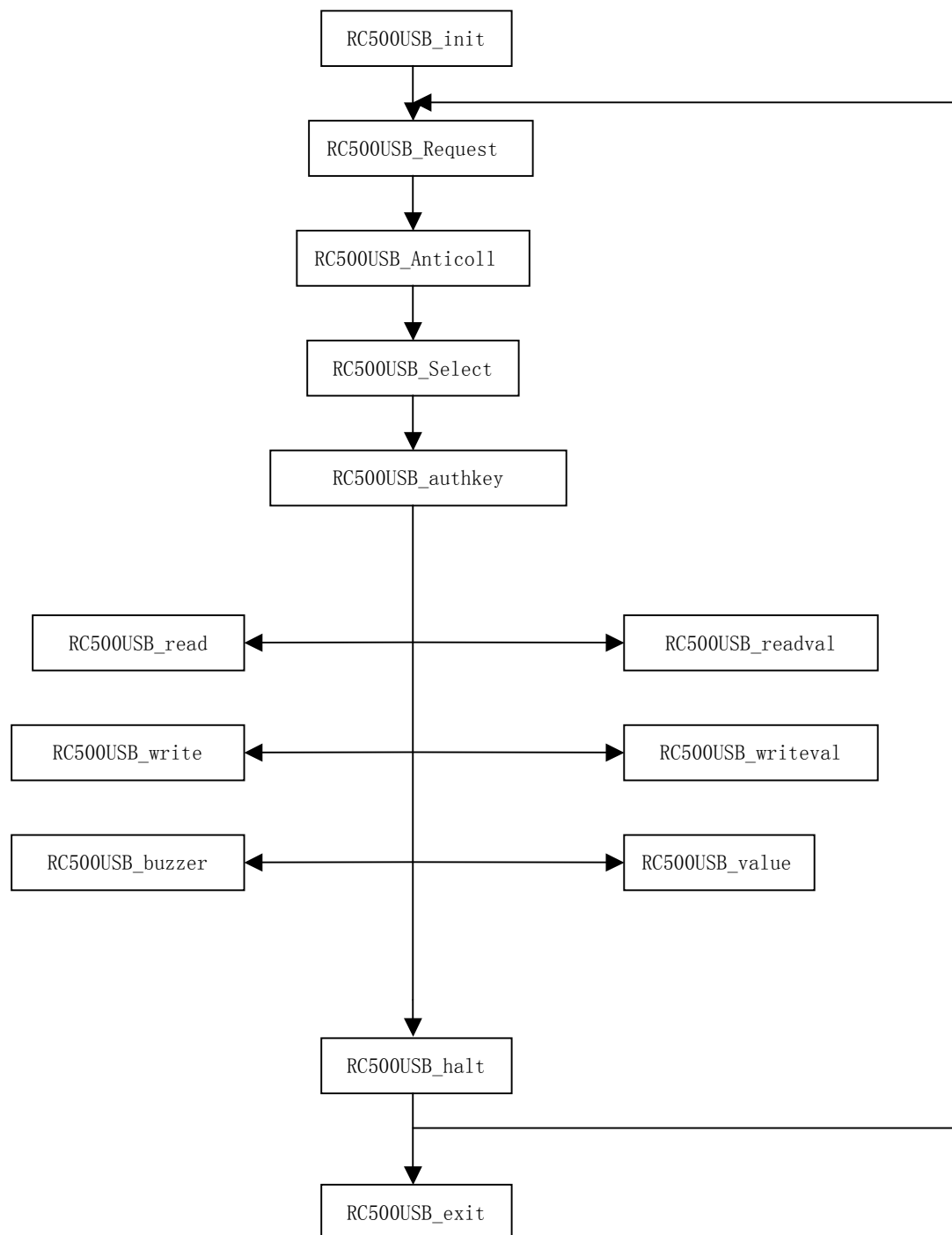
ByVal (可选), 表示该参数按值传递。

ByRef (可选), 表示该参数按地址传递。

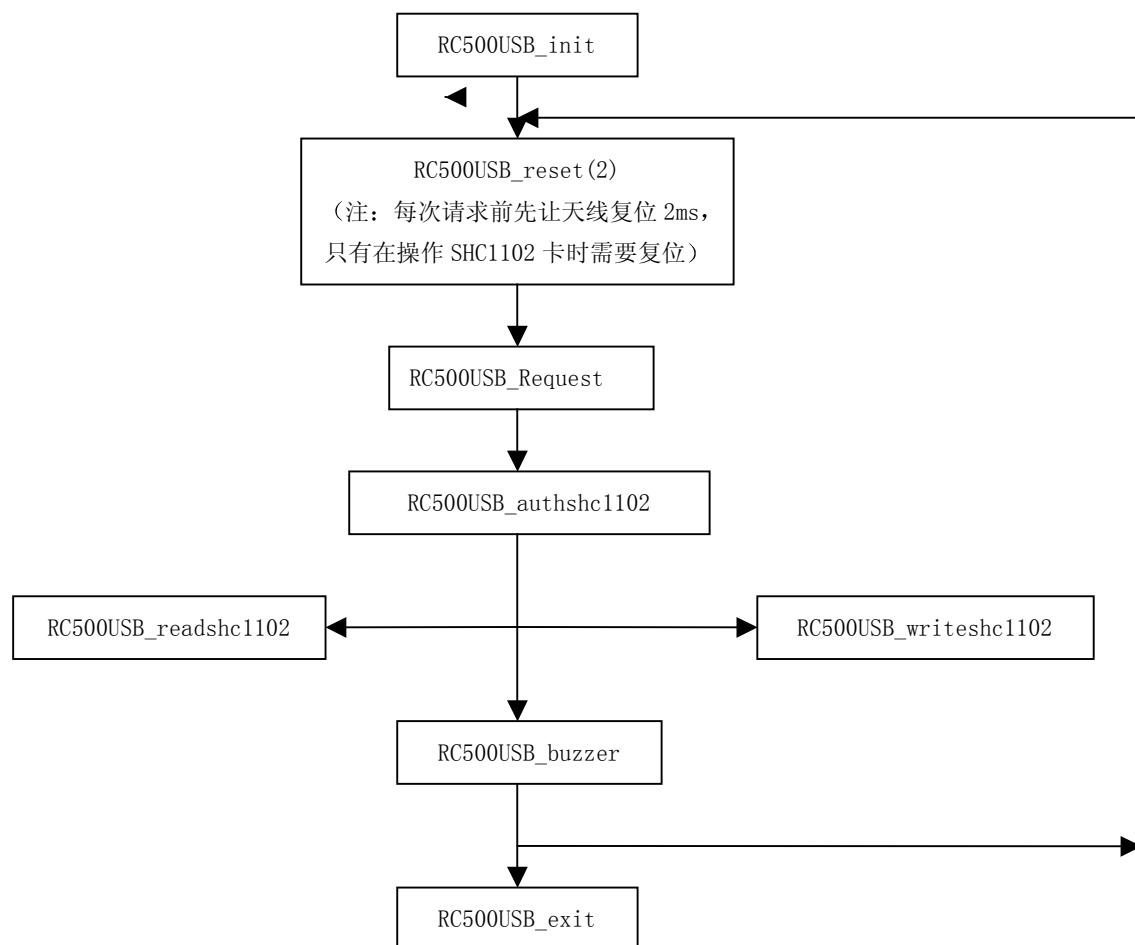
例如: `Public Declare Function RC500USB_read Lib "RC500USB" (ByVal blocknr As Byte, ByRef data As Byte) As Byte`

2.2 库函数使用流程

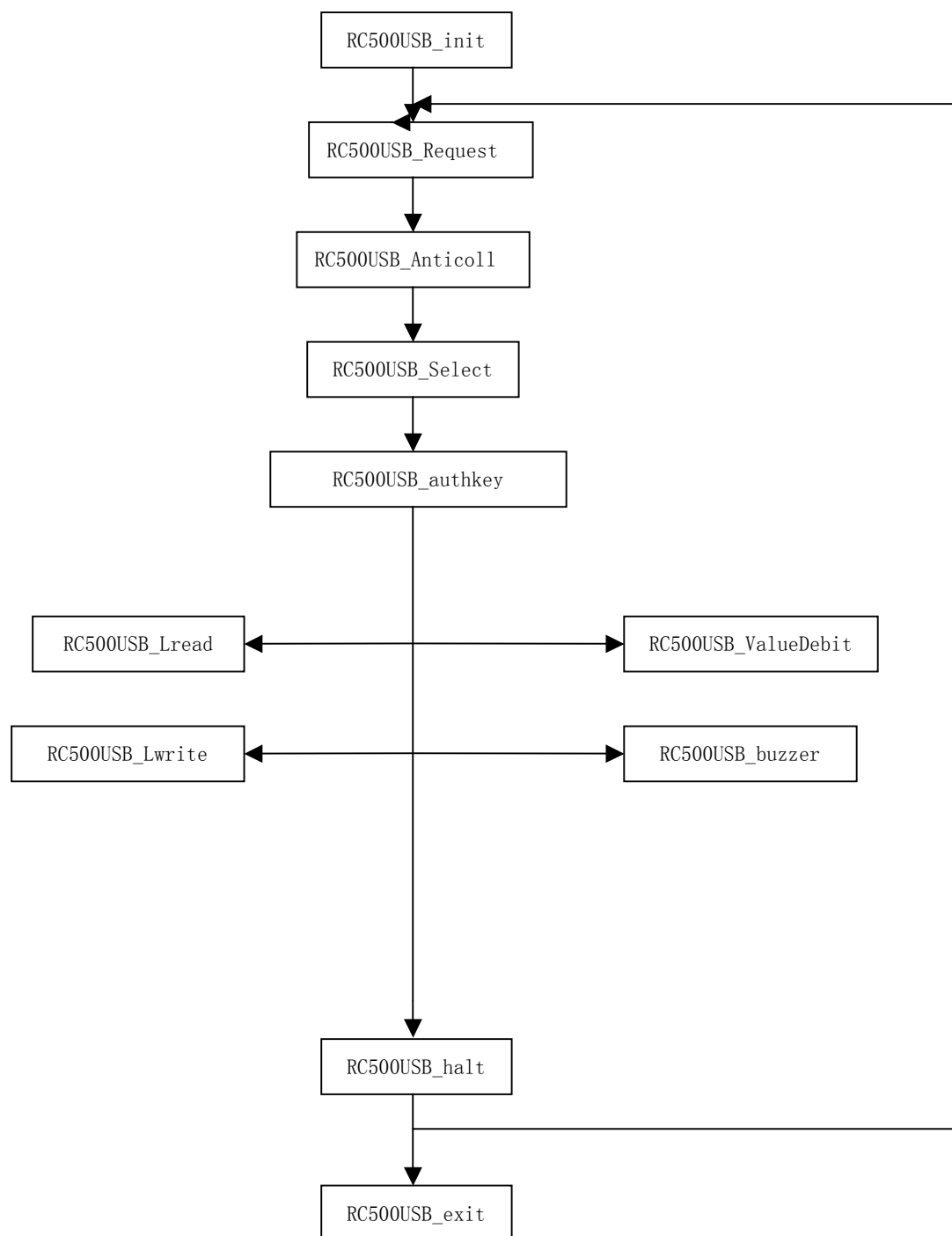
2.2.1 操作 MIFARE S70/S50 卡时的操作流程:



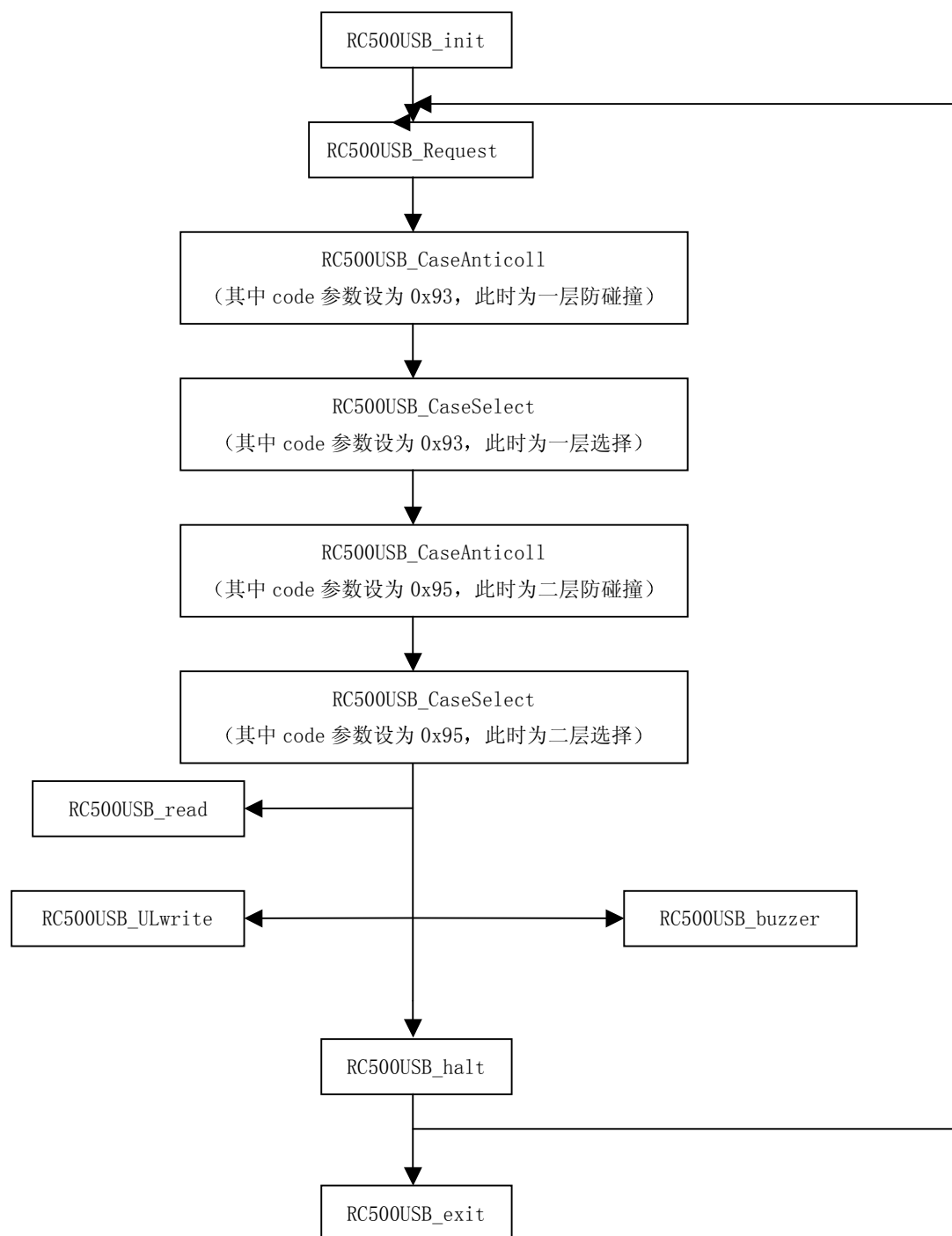
2.2.2 操作 SHC1102 卡时的操作流程:



2.2.3 操作 MF1 L10 卡时的操作流程:



2.2.4 操作 MF0 U1 卡时的操作流程:



2.3 使用举例

2.3.1 操作 MIFARE S70/S50 例子

```
#include "RC500USB.h"
```

```
unsigned long snr;  
unsigned short type;  
unsigned char size;  
unsigned char data[16];  
unsigned char szpwd[6]={0xff,0xff,0xff,0xff,0xff,0xff};  
for(int i=0;i<16;i++)  
{  
    data[i]=i;  
}  
long value=0x500;
```

```
RC500USB_init();                //打开 USB  
while(1)  
{  
    if(RC500USB_request(0,type)!=0)    //对第一个读卡器进行操作  
        continue;  
    if(RC500USB_anticoll(0,snr)!=0)    //进行防碰撞选择，成功则返回卡号  
        continue;  
    if(RC500USB_select(snr,size)!=0)    //选择卡  
        continue;  
    if(RC500USB_authkey(0,szpwd,0)!=0)    //验证第 0 扇区的密钥 A  
        continue;  
    if(RC500USB_wirte(2,data)!=0)    //往第二块写入数据  
        continue;  
    if(RC500USB_read(2,data)!=0)    //读取第二块的数据  
        continue;  
    for(i=0;i<16;i++)  
    {  
        printf("%x ",data[i]);  
    }  
    printf("\n");  
    if(RC500USB_writeval(1,value)!=0)    //往第一块写入数值 value  
        continue;  
    if(RC500USB_value(0xc1,1,value,1)!=0)    //把第一块的数值加上 value  
        continue;  
    if(RC500USB_readval(1,value)!=0)    //读取第一块的数值  
        continue;
```

```
printf("%x\n",value);
RC500USB_halt();           //使卡进入 halt 状态
RC500USB_buzzer(1,25,25,1); //让蜂鸣器响一次
sleep(50);
}
```

2.3.2 操作 SHC1102 例子

```
unsigned short type;
unsigned char data[4];
unsigned char szpwd[4]={0xff,0xff,0xff,0xff};
for(int i=0;i<4;i++)
{
    data[i]=i;
}

RC500USB_init();           //打开端口和设定波特率
while(1)
{
    RC500USB_reset(2);      //让天线复位 2ms
    if(RC500USB_request(0,type)!=0) //发送请求命令
        continue;
    if(RC500USB_authshc1102(8,szpwd)!=0) //验证密钥
        continue;
    if(RC500USB_wirteshc1102(2,data)!=0) //往第二块写入数据
        continue;
    if(RC500USB_readshc1102(2,data)!=0) //读取第二块的数据
        continue;
    RC500USB_buzzer(1,25,25,1); //让蜂鸣器响一次
    sleep(50);
}
```

2.3.3 操作 MF1 L10 例子

```
#include "RC500USB.h"

unsigned long snr;
unsigned short type;
unsigned char size;
unsigned char data[8];
unsigned char szpwd[6]={0xff,0xff,0xff,0xff,0xff,0xff};
for(int i=0;i<4;i++)
{
    data[i]=i;
}
```

```
}

RC500USB_init();                //打开 USB
while(1)
{
    if(RC500USB_request(0,type)!=0)    //对第一个读卡器进行操作
        continue;
    if(RC500USB_anticoll(0,snr)!=0)    //进行防碰撞选择，成功则返回卡号
        continue;
    if(RC500USB_select(snr,size)!=0)    //选择卡
        continue;
    if(RC500USB_authkey(0,szpwd,0)!=0)    //验证密钥 A
        continue;
    if(RC500USB_wirte(10,data)!=0)    //往第 10 块写入数据
        continue;
    if(RC500USB_read(10,data)!=0)    //读取第 10,11 块的数据
        continue;
    RC500USB_halt();                //使卡进入 halt 状态
    RC500USB_buzzer(1,25,25,1);    //让蜂鸣器响一次
    sleep(50);
}
```

2.3.4 操作 MF0 U1 例子

```
#include "RC500USB.h"
```

```
unsigned long snr[2];
unsigned short type;
unsigned char sak;
unsigned char data[16];
for(int i=0;i<4;i++)
{
    data[i]=i;
}

RC500USB_init();                //打开 USB
while(1)
{
    if(RC500USB_request(0,type)!=0)    //对第一个读卡器进行操作
        continue;
    if(RC500USB_CaseAnticoll(0,0x93,snr[0])!=0)    //进行第一层防碰撞选择，成功则返回卡号的一部分
        continue;
    snr[0]=(snr[0]>>8);                //此为卡号的低三个字节部分
    if(RC500USB_CaseSelect(0x93,snr[0],sak)!=0)    //进行第一层选择
```

```
        continue;
    if(RC500USB_CaseAnticoll(0,0x95,snr[1])!=0)    //进行第二层防碰撞选择，成功则返回卡号其余部分
                                                    //snr[1]中存储卡号的高四个字节
        continue;
    if(RC500USB_CaseSelect(0x95,snr[1],sak)!=0)    //进行第二层选择
        continue;
    if(RC500USB_wirte(8,data)!=0)                  //往第 8 块写入数据
        continue;
    if(RC500USB_read(8,data)!=0)                  //读取第 8,9,10,11 块的数据
        continue;
    RC500USB_halt();                              //使卡进入 halt 状态
    RC500USB_buzzer(1,25,25,1);                  //让蜂鸣器响一次
    sleep(50);
}
```

附录:

函数执行时间:

函数名称	执行时间 (ms)
RC500USB_request	3.295
RC500USB_anticoll	3.911
RC500USB_anticoll2	3.930
RC500USB_select	4.024
RC500USB_authentication	4.946
RC500USB_authentication2	5.970
RC500USB_authkey	6.922
RC500USB_halt	4.020
RC500USB_read	5.000
RC500USB_write	10.285
RC500USB_writeval	10.400
RC500USB_readval	5.101
RC500USB_value	15.644
RC500USB_load_key	9.967
RC500USB_reset	2.914
RC500USB_close	0.911
RC500USB_config	30.979
RC500USB_set_control_bit	0.941
RC500USB_clr_control_bit	1.847
RC500USB_buzzer	1.965
RC500USB_read_E2	3.551 (16 bytes)
RC500USB_write_E2	9.914 (16 bytes)
RC500USB_authshc1102	4.040
RC500USB_readshc1102	3.848
RC500USB_writeshc1102	21.036

错误码列表:

名称	值	描述
MI_OK, COMM_OK	0	函数调用成功
MI_NOTAGERR	1	在有效区域内没有卡
MI_CRCERR	2	从卡中接收到了错误的 CRC 校验和
MI_EMPTY	3	值溢出
MI_AUTHERR	4	不能验证
MI_PARITYERR	5	从卡中接收到了错误的校验位
MI_CODEERR	6	通信错误
MI_SENDRERR	8	在防冲突时读到了错误的串行码
MI_NOTAUTHERR	10	卡没有验证
MI_BITCOUNTERR	11	从卡中接收到了错误数量的位
MI_BYTECOUNTERR	12	从卡中接收到了错误数量的字节
MI_TRANSERR	14	调用 Transfer 函数出错
MI_WRITEERR	15	调用 Write 函数出错
MI_INCRERR	16	调用 Increment 函数出错
MI_DECRERR	17	调用 Decrment 函数出错
MI_READERR	18	调用 Read 函数出错
MI_COLLERR	24	冲突错
MI_QUIT	30	上一次了送命令时被打断
MIS_CHK_OK	0	Check Write 正确
MIS_CHK_FAILED	1	Check Write 出错
MIS_CHK_COMPERR	2	Check Write:写出错（比较出错）
COMM_ERR	255	串行通信错误