

# ZLG500B 读卡模块使用指南

版本号 1.4

2004年8月10日

## 目 录

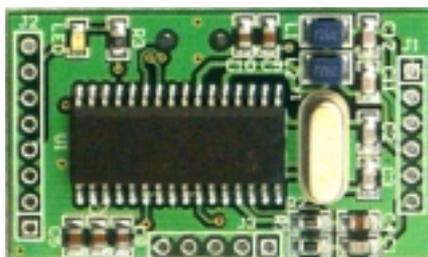
<b>§1 概述</b> .....	<b>3</b>
1.1 照片.....	3
1.1.1 独立模块.....	3
1.1.2 天线一体化模块.....	3
1.2 特征.....	4
1.3 电气特性.....	4
<b>§2 硬件描述</b> .....	<b>5</b>
2.1 引脚描述.....	5
2.2 串行接口.....	6
<b>§3 ZLG500B 读卡模块数据传输协议</b> .....	<b>7</b>
3.1 介绍.....	7
3.2.1 自动波特率探测.....	7
3.2.2 控制字符定义.....	7
3.2.3 协议描述.....	8
3.2.4 数据块格式.....	8
3.3 ZLG500B 和 MCU 命令 C51 函数(版本号 1.4).....	10
3.3.1 底层函数和高级函数.....	10
3.3.2 状态值列表.....	11
3.3.3 版本说明.....	12
3.4 函数描述.....	12
3.4.1 请求—Request.....	13
3.4.2 防碰撞—Anticoll.....	14
3.4.3 选择—Select.....	15
3.4.4 证实—Authentication.....	16
3.4.5 暂停—Halt.....	17
3.4.6 读—Read.....	18
3.4.7 写—Write.....	19
3.4.8 加—Increment.....	20
3.4.9 减—Decrement.....	21
3.4.10 恢复—Restore.....	22
3.4.11 传送—Transfer.....	23
3.4.12 装载密钥—Load Key.....	24
3.4.13 复位—Reset.....	25
3.4.14 获取信息—Get Info.....	26
3.4.15 置位控制位—Set Control Bit.....	27
3.4.16 清除控制位—Clr Control Bit.....	28
3.4.17 配置—Config.....	29
3.4.18 检查写—Check Write.....	30
3.4.19 输出蜂鸣器信号—Buzzer.....	31
3.4.20 读 EEPROM.....	32
3.4.21 写 EEPROM.....	33

3.4.22	关闭 RC500—Close .....	34
3.4.23	值操作.....	35
3.4.24	防碰撞 2—Anticoll2.....	36
3.4.25	证实 2—Authentication2.....	37
3.4.26	直接密码证实—AuthKey .....	38
3.4.27	多层防碰撞—CascAnticoll.....	39
3.4.28	多层选择—Select.....	40
3.4.29	写 UltraLight—ULWrite.....	41
3.4.30	带内部自动传送的值操作.....	42
3.4.31	写寄存器.....	43
3.4.32	读寄存器.....	44
3.5	全局变量.....	45
3.6	串行口看门狗定时器.....	45
3.7	应用程序举例.....	46

## § 1 概述

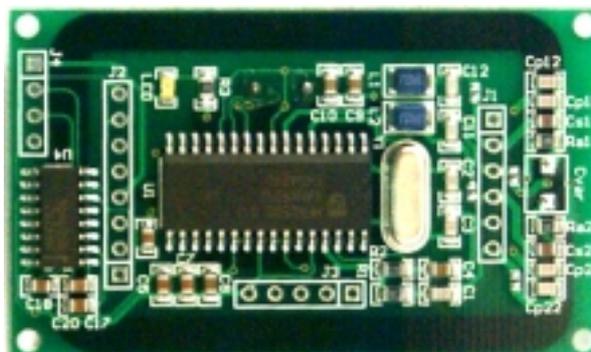
### 1.1 照片

#### 1.1.1 独立模块



需外加天线才可使用，实际尺寸：41.5mm×25.3mm。

#### 1.1.2 天线一体化模块



实际尺寸：58mm×34.4mm。

## 1.2 特征

- 四层电路板设计，双面表贴，EMC 性能优良
- 采用最新 PHILIPS 高集成 IS014443A 读卡芯片—MF RC500
- UART 串行接口，能外接 RS232 或 RS485 芯片
- 自动波特率探测
- 控制线输出口
- 蜂鸣器信号输出口，能用软件控制输出频率及持续时间
- 能读写 RC500 内 EEPROM
- 发光二极管指示模块当前状态
- 可提供 C51 函数库

## 1.3 电气特性

符号	参数	最小	典型	最大	单位
T <sub>STR</sub>	环境或存储温度范围	-40		+150	°C
T <sub>OP</sub>	工作温度范围	-25	+25	+85	°C
V <sub>CC</sub>	工作电压范围	4.5	5	5.5	V
I <sub>CC1</sub>	电流消耗, config 成功后		75		MA
I <sub>CC2</sub>	电流消耗, close 成功后		7		MA

## § 2 硬件描述

### 2.1 引脚描述

如照片所示，J1 为与天线的接口，J2 为与 MCU 的接口。如下二表所示。

天线接口表

管脚	符号	描述
J1-1	GND	地
J1-2	TX1	天线发送 1
J1-3	GND	地
J1-4	TX2	天线发送 2
J1-5	GND	地
J1-6	RX	天线接收

CMOS/TTL 电平串行接口表

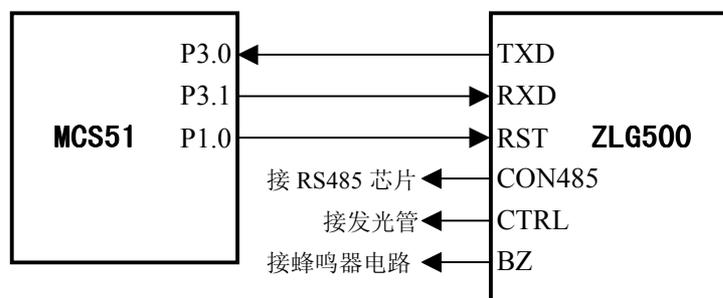
管脚	符号	类型	描述
J2-1	CTRL	输出	控制线输出
J2-2	BZ	输出	蜂鸣器信号输出，平时为高，输出方波或低电平有效
J2-3	CON485	输出	RS485 控制，平时为低，TXD 发送时为高
J2-4	VCC	PWR	电源正端
J2-5	RST	复位	内部 MCU 复位端，高电平有效
J2-6	GND	PWR	电源负端
J2-7	RXD	输入	UART 接收端
J2-8	TXD	输出	UART 发送端

RS232 电平串行接口表

管脚	符号	描述
J4-1	VCC	加+5V 电源
J4-2	RXD	232 接口接收端
J4-3	TXD	232 接口发送端
J4-4	GND	地

## 2.2 串行接口

ZLG500B 模块可方便地与 MCU 进行接口，如下图所示与 MCS51 单片机的典型接口。



也可以与 RS232 或 RS485 芯片直接连接，与外部控制器或 PC 接口。

ZLG500B 的 RST 有两种接法，一是接硬件复位电路，如阻容复位等，这样系统上电后必须要等待 ZLG500B 复位结束；二是接外部 MCU 的一个 I/O 口，由 MCU 控制复位。推荐使用第二种方法，这样在 ZLG500B 出现异常时可由 MCU 控制复位。

## § 3 ZLG500B 读卡模块数据传输协议

### 3.1 介绍

本文档描述了 MIFARE 串行读卡模块 ZLG500B 与主机（微处理器）之间的串行通信软件的通信协议和命令。

ZLG500B 是一个简单的串行读写模块，它可以读写 MIFARE 无线智能卡。在这个器件中包括了一个 PCB 天线，提供了一个 UART 接口（CMOS 电平），可受控于主机微处理器。

### 3.2 协议

#### 3.2.1 自动波特率探测

UART 接口一帧的数据格式为 1 个起始位，8 个数据位、无奇偶校验位、1 个停止位，自动波特率探测。ZLG500B 上电后，将 RC500 内 EEPROM 中地址为 0x7F 单元的内容读出，取其低 3 位来初始化自己的波特率。在接收主机发出的 STX 的同时，测量波特率，若一致，则继续通信；若不一致，则修改自己的波特率与主机一致，存入 0x7F 单元，准备接收下一个 STX。

ZLG500B 支持的波特率有：2400、4800、9600、19200、28800 和 57600。波特率与 RC500 内 EEPROM 中 0x7F 单元低 3 位值的对应关系为：

波特率 (BPS)	0x7F
2400	0x00
4800	0x01
9600	0x02
19200	0x03
28800	0x04
57600	0x05

#### 3.2.2 控制字符定义

描述	定义	值
开始符	STX	0x20
终止符	ETX	0x03
应答	ACK	0x06
无应答	NAK	0x15

### 3.2.3 协议描述

通信必须先由主机发送命令和数据给 ZLG500B，ZLG500B 执行命令完毕后，将命令执行的状态和响应数据发回主机。

开始通信前，收发双方必须处于就绪状态。首先主机发出 STX，然后等待 ZLG500B 的响应 ACK。若在 10ms 内未检测到此响应，或收到 NAK，则再次发送 STX，如此反复 3 次，若 3 次均未收到正确应答 ACK，则退出本次传输，将错误代码返回给主程序，由主程序进行错误处理。若主机收到 ZLG500B 正确响应 ACK，则可将数据块发送出去，最后发送 1 个终止符 ETX 结束本次发送。主机发送格式：

主机	数据传送方向	ZLG500B	说明
STX	→		主机在 10ms 内未收到 ACK 或收到 NAK，应至少再重发 STX 一次；主机收到 ACK 后，必须在 50ms 内发送数据，且发送两个字节的间隔时间必须少于 10ms。
	←	ACK	
DATA+ETX	→		

然后主机等待 ZLG500B 发回的状态和响应数据。若在 300ms 内未检测到响应，则退出本次传输，且向主程序报告错误代码。ZLG500B 发送格式：

ZLG500B	数据传送方向	主机	说明
STX	→		ZLG500B 在 50ms 内未收到 ACK，不重发 STX
	←	ACK	
DATA+ETX	→		

### 3.2.4 数据块格式

#### 3.2.4.1 主机→ZLG500B（命令模式）

<i>SeqNr</i>	<i>Command</i>	<i>Len</i>	<i>Data[0..N]</i>	<i>BCC</i>
INFO[0]	.	.	.	INFO[n]

*SeqNr*: 1 Byte 数据交换包的序号  
*Command*: 1 Byte 命令字符  
*Len*: 1 Byte 数据的长度  
*Data[...]*: *Len* Byte 数据字节  
*BCC*: 1Byte 的 BCC 校验

#### 3.2.4.2 ZLG500B→主机（响应模式）

<i>SeqNr</i>	<i>status</i>	<i>Len</i>	<i>Data[0..N]</i>	<i>BCC</i>
INFO[0]	.	.	.	INFO[n]

*SeqNr*: 1 Byte 数据交换包的序号  
*status*: 1 Byte 状态字符  
*Len*: 1 Byte 数据的长度  
*Data[...]*: *Len* Byte 数据字节  
*BCC*: 1Byte 的 BCC 校验

### 3.2.4.3 数据块格式描述

- 数据交换包的序号由主机发送数据块时产生，取值范围为 0-255。在经过一次正确的数据交换后，主机在发送下一个命令时，将数据包的序号加 1。MSR 返回最近接收的包序号。通常主机应用程序最好检查命令/响应包交换时的数据包的序号。
- 不管在执行命令时出现了任何错误，响应包中的数据长度为 0 ( $Len = 0$ )。
- BCC 校验码计算数据块中所有的 INFO 字节。然后将结果传送到数据块的最后一个字节，如下式所示：

$$INFO[n] = BCC = \sim (INFO[0] \oplus INFO[1] \oplus \dots \oplus INFO[n-1]) \quad (\oplus \dots \text{XOR}, \sim \dots \text{NOT})$$

### 3.3 ZLG500B 和 MCU 命令 C51 函数(版本号 1.4)

#### 3.3.1 底层函数和高级函数

命令		参数		补充
名称	数值	发送	接收	
Request	0x41	<i>_Mode</i>	<i>_TagType</i>	发出询问命令, 检查在有效范围内是否有卡存在
Anticoll	0x42	<i>_Bcnt</i>	<i>_SNR</i>	开始防冲突操作, 返回卡的序号
Anticoll2	0x71	<i>_Encoll, _Bcnt</i>	<i>_SNR</i>	可禁止或允许许多张卡进入
CascAnticoll*	0x74	<i>_Bcnt, _Select_Code</i>	<i>_SNR</i>	可实现三层防碰撞协议
Select	0x43	<i>_SNR</i>	<i>_Size</i>	选择卡, 返回卡的存储容量
CascSelect*	0x75	<i>_Select_Code, _SNR</i>	<i>_Sak</i>	可实现三层选择
Authentication	0x44	<i>_Mode, _SecNr</i>	--	开始验证操作
Authentication2	0x72	<i>_Mode, _SecNr, _KeyNr</i>	--	可选择密匙区验证
AuthKey	0x73	<i>_Mode, _SecNr, *_Key</i>	--	直接密码验证
Halt	0x45	--	--	将卡置于挂起模式
Read	0x46	<i>_Adr</i>	<i>_Data</i>	从卡中相应地址中读出一个 16 字节的块
Write	0x47	<i>_Adr, _Data</i>	--	向卡中相应地址写入一 16 字节的数据块
ULWrite*	0x76	<i>_Adr, _Data</i>	--	向 mifare UltraLight 卡中相应地址页写入 4 字节数据
Increment	0x48	<i>_Adr, _Value</i>	--	增加访问单元块的字节数, 并将结果保存在卡的内部寄存器
Decrement	0x49	<i>_Adr, _Value</i>	--	减少访问单元块的字节数, 并将结果保存在卡的内部寄存器
Resore	0x4A	<i>_Adr</i>	--	将所访问单元块的字节数保存在卡的内部寄存器中
Transfer	0x4B	<i>_Adr</i>	--	将卡内部寄存器的内容转输到访问快的字节数
Value	0x70	<i>_Mode, _Adr, _Value, _Trans_Adr</i>	--	包含加、减、恢复函数, 并带自动传送
ValueDebit*	0x77	<i>_Mode, _Adr, _Value</i>	--	带内部自动传送的值操作, 支持 Mifare Light
LoadKey	0x4C	<i>_Mode, _SecNr, _Nkey</i>	--	改变存储在 EEPROM 中的密匙
Reset	0x4E	<i>_Msec</i>	--	关闭天线输出数 ms, 使卡复位
Get Info	0x4F	--	<i>_Info</i>	读取固件信息 RC500 序列号
Set Control Bit	0x50	--	--	将控制位置为高电平
Clr Control Bit	0x51	--	--	将控制位置为低电平
Config	0x52	--	--	复位且配置 RC500

Write_Reg	0x3D	<i>_Reg, _Value</i>		写 RC500 内寄存器
Read_Reg	0x3E	<i>_Reg</i>	<i>*_Value</i>	读 RC500 内寄存器
Close	0x3F	--	--	关闭 RC500
Check Write	0x53	<i>_SNR, _Authmode, _Adr, _Data</i>	--	将所传送的数据和上一次所写的数据进行比较
Buzzer	0x60	<i>_Freguence, _Opentm, _Closetm, _Repcnt</i>	--	输出驱动蜂鸣器信号, 能控制动作时间、间隙时间和重复次数
Read E2	0x61	<i>_Adr, _Length</i>	<i>_Data</i>	读 RC500 内 EEPROM 的内容
Write E2	0x62	<i>_Adr, _Length, _Data</i>	--	写数据到 RC500 内 EEPROM

### 3.3.2 状态值列表

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_KEYERR	9	证实密码错
MI_NOTAUTHERR	10	卡没有验证
MI_BITCOUNTEERR	11	从卡中接收到了错误数量的位
MI_BYTECOUNTEERR	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_ACCESSTIMEOUT	27	访问超时
MI_QUIT	30	上一次的送命令时被打断
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

### 3.3.3 版本说明

一、版本 1.1 对应于 1.0 增加了两个函数：

- 防碰撞 **Anticoll2**：可允许或禁止多张卡同时进入天线区。
- 证实 **Authentication2**：可选择不同的密匙区对一个扇区进行密码验证。

二、版本 1.2 对应于 1.1 增加了一个函数：

- 直接密码证实 **Authkey**：密码证实时，6 字节密码直接由主控制器传入模块。

三、版本 1.3 对应于 1.2 增加了四个函数：

- 多层防碰撞 **CascAnticoll**：支持多字节序列号
- 多层选择 **CascSelect**：支持多字节序列号
- UL 写函数 **ULWrite**：支持 Mifare Ultralight 四字节写
- 自动传送值操作 **ValueDebit**：支持 Mifare Light 和 pro

四、版本 1.4 相对于 1.3 增加了二个函数：

- 写寄存器 **Write\_Reg**：可直接控制 RC500 内的寄存器
- 读寄存器 **Read\_Reg**：可读出 RC500 内的寄存器

### 3.4 函数描述

下面是 C51 函数声明，包含在头文件“ZLG500B.h”中，写应用程序时，将其包含在应用函数中即可。

### 3.4.1 请求—Request

声明:

```
uchar mifs_request(uchar _Mode,uchar idata *_TagType);
```

MCU ⇒ ZLG500:

命令符: 0x41  
长度: 1  
Data[0]: \_Mode

ZLG500 ⇒ MCU :

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_BITCOUNTERR, COMM\_ERR  
长度: 2  
Data[0]: tagtype (低字节)  
Data[1]: tagtype (高字节)

参数:

\_Mode: 

							ALL
--	--	--	--	--	--	--	-----

ALL=0: 请求天线范围内 IDLE 状态的卡 (HALT 状态的除外)

ALL=1: 请求天线范围内的所有卡

\_tagtype: 当发生错误时, 不返回任何内容 (Len=0)

描述:

此函数发送 Request 命令, 检查在有效范围内是否有卡存在。这个函数在选择一个新的卡, 是必须调用的。

该函数的典型执行时间:

波特率 9600: 20.4ms

波特率 57600: 6ms

## 3.4.2 防碰撞—Anticoll

声明:

```
uchar mifs_anticoll(uchar _Bcnt,uchar idata *_SNR);
```

MCU ⇒ ZLG500:

命令符: 0x42

长度: 1

Data[0]: \_Bcnt

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_BITCOUNTERR, COMM\_ERR

长度: 4

Data[0]: snr(LL)

Data[1]: snr(LH)

Data[2]: snr(HL)

Data[3]: snr(HH)

参数:

\_Bcnt: 为预选卡所分配的位的个数, 通常Bcnt=0

\_SNR: 卡的序列号。以小端模式存贮在一个无符号的四字节数组中, 低字节放在低地址处。

描述:

此函数开始防冲突操作。必须在调用了Request命令后立即调用。当知道了所要选择卡的序列号后, 就没有必要调用AntiColl。此时, 调用了Request后, 直接调用Select函数即可。

该函数的典型执行时间:

波特率 9600: 23.6ms

波特率57600: 7.6ms

### 3.4.3 选择—Select

声明:

```
uchar mifs_select(uchar idata * _SNR,uchar idata * _Size);
```

MCU ⇒ ZLG500:

命令符: 0x43  
长度: 4  
Data[0]: snr(LL)  
Data[1]: snr(LH)  
Data[2]: snr(HL)  
Data[3]: snr(HH)

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_CRCERR, MI\_PARITYERR,  
MI\_BITCOUNTERR, COMM\_ERR  
长度: 1  
Data[0]: \_Size

参数:

\_SNR: 卡的序号。存贮在一个无符号4字节字符数组中，低字节放在低地址处。  
\_Size: 当Select命令返回值为MI\_OK时，ATS (answer to select)将返回主机。

描述:

这个函数选择某一个序列号的卡，返回ATS字节给主机。

该函数的典型执行时间:

波特率 9600: 23.9ms

波特率57600: 7.9ms

### 3.4.4 证实—Authentication

声明:

```
uchar mifs_authentication(uchar _Mode,uchar _SecNr);
```

MCU ⇒ ZLG500:

命令符: 0x44  
长度: 2  
Data[0]: \_Mode  
Data[1]: \_SecNr

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_AUTHERR, MI\_BITCOUNTERR,  
MI\_PARITYERR, COMM\_ERR  
长度: 0

参数:

\_Mode: 

					AB		
--	--	--	--	--	----	--	--

AB = 0: 利用密钥A进行验证

AB = 1: 利用密钥B进行验证

\_SecNr: 所访问卡的扇区号, 必须小于16。

描述:

在对卡进行读、写、加、减等操作前, 必须对卡进行验证。若卡中一扇区的密钥与RC500中相应密码区存储的密码相匹配。则证实成功, 函数将返回MI\_OK。

该函数的典型执行时间:

波特率 9600: 22.7ms

波特率57600: 9.2ms

### 3.4.5 暂停—Halt

声明:

```
uchar mifs_halt(void);
```

MCU ⇒ ZLG500:

命令符: 0x45

长度: 0

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_CODE, COMM\_ERR

长度: 0

参数:

无

描述:

此函数将所选择卡置为挂起状态。如果要进行重新选择,则应用ALL模式调用Request命令。或将卡复位,如将卡离开天线操作区再进入,或执行复位函数mifs\_reset();

该函数的典型执行时间:

波特率 9600: 18ms

波特率57600: 6.2ms

### 3.4.6 读—Read

描述:

```
uchar mifs_read(uchar _Adr,uchar idata *_Data);
```

MCU ⇒ ZLG500:

命令符: 0x46

长度: 1

Data[0]: \_Adr

ZLG500 ⇒ MCU:

状态值: MI\_OK , MI\_QUIT , MI\_NOTAGERR , MI\_CRCERR , MI\_NOTAUTHERR ,  
MI\_PAROTUERR, MI\_BITCOUNTERR, COMM\_ERR

长度: 16

Data[0]: 所访问块的第一个字节

:

Data[15]: 所访问块的最后一个字节

参数:

\_Adr: 所读数据地址

描述:

此函数在所选的卡通过验证后，读取一个16字节的块。

该函数的典型执行时间:

波特率 9600: 38.1ms

波特率57600: 12ms

### 3.4.7 写—Write

描述:

```
uchar mifs_write(uchar _Adr, uchar idata *_Data);
```

MCU ⇒ ZLG500:

命令符:	0x47
长度:	17
Data[0]:	address
Data[1]:	所访问块的第一个字节
:	
Data[16]:	所访问块的最后一个字节

ZLG500 ⇒ MCU:

状态值:	MI_OK, MI_QUIT, MI_NOTAGERR, MI_NOTAUTHERR, MI_WRITEERR, MI_BITCOUNTERR, COMM_ERR
长度:	0

参数:

_Adr:	所写数据块地址 (0—63)
_Data:	16 字节数据指针

描述:

此函数在所选的卡通过验证后，写入一个16字节的块。

该函数的典型执行时间:

波特率 9600: 42.5ms

波特率57600: 16.2ms

### 3.4.8 加一Increment

声明:

```
uchar mifs_increment(uchar _Adr,ulong idata *_Value);
```

MCU ⇒ ZLG500:

命令符: 0x48  
长度: 5  
Data[0]: \_Adr  
Data[1]: \_Value(LL)  
Data[2]: \_Value(LH)  
Data[3]: \_Value(HL)  
Data[4]: \_Value(HH)

ZLG500 ⇒ MCU:

状态值: MI\_OK , MI\_QUIT , MI\_NOTAGERR , MI\_NOTAUTHERR , MI\_INCRERR ,  
MI\_BITCOUNTERR, COMM\_ERR, MI\_INCRERR  
长度: 0

参数:

\_Adr: 所加数据块的地址  
\_Value: 增加值。贮在一个无符号的长整型变量（4Byte）中，低地址存放高字节。

描述:

该函数读被访问的值块，检查数据的结构，用传输的值减值块的值，并将结果贮存在卡的内部寄存器中。值块有标准的格式。不能自动进行对卡中 EEPROM 的写操作。

该函数的典型执行时间:

波特率 9600: 16.7ms

波特率 57600: 4.1ms

### 3.4.9 减—Decrement

声明

```
uchar mifs_decrement(uchar _Adr,ulong idata *_Value);
```

MCU ⇒ ZLG500:

命令符: 0x49  
长度: 5  
Data[0]: \_Adr  
Data[1]: \_Value(LL)  
Data[2]: \_Value(LH)  
Data[3]: \_Value(HL)  
Data[4]: \_Value(HH)

MSR ⇒ 主机:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_NOTAUTHERR, MI\_DECRERR,  
MI\_BITCOUNTERR, COMM\_ERR  
长度: 0

参数:

\_Adr: 所减数据块的地址  
\_Value: 减少值。贮在一个无符号的长整型变量（4Byte）中，低地址存放高字节。

描述:

该函数读被访问的值块，检查数据的结构，用传输的值减值块的值，并将结果贮存在卡的内部寄存器中，值块有标准的格式。不能自动进行对卡中 EEPROM 的写操作。

该函数的典型执行时间:

波特率 9600: 16.7ms

波特率 57600: 4.1ms

### 3.4.10 恢复—Restore

声明:

```
uchar mifs_restore(uchar _Adr);
```

MCU ⇒ZLG500:

命令符: 0x4A

长度: 1

Data[0]: \_Adr

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_NOTAUTHERR, MI\_DECRERR,  
MI\_BITCOUNTERR, COMM\_ERR, MIS\_EMPTY

长度: 0

参数:

\_Adr: 所读块的地址

描述:

该函数读被访问的值块，且检查数据的结构，并将结果贮存在卡的内部寄存器中。不能自动进行对卡中 EEPROM 的写操作。

该函数的典型执行时间:

波特率 9600: 16.7ms

波特率 57600: 4.1ms

### 3.4.11 传送—Transfer

声明:

```
uchar mifs_transfer(uchar _Adr);
```

MCU ⇒ZLG500:

命令符: 0x4B

长度: 1

Data[0]: \_Adr

ZLG500⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_CODE, MI\_BITCOUNTEERR, MI\_TRANSERR,  
MI\_CODEERR, COMM\_ERR

长度: 0

参数:

\_Adr: 卡中欲传输的块地址

描述:

此函数将卡的内部寄存器内容转送给所选块地址。在此操作前必须通过验证。这个函数只能在 Increment,Decrement,或 Restore 操作后。

该函数的典型执行时间:

波特率 9600: 31.8ms

波特率 57600: 19.2ms

### 3.4.12 装载密钥—Load Key

声明:

```
uchar mifs_load_key(uchar _Mode,uchar _SecNr,uchar *_Nkey);
```

MCU ⇒ZLG500:

```
命令符:    0x4C
长度:      8
Data[0]:   _Mode
Data[1]:   _SecNr
Data[2]:   _Nkey[0]
::
Data[7]:   _Nkey[5]
```

ZLG500⇒主机:

```
状态值:    MI_OK, MI_QUIT, MI_AUTHERR, COMM_ERR
长度:      0
```

参数:

\_Mode:

					AB		
--	--	--	--	--	----	--	--

AB = 0: 利用密钥A进行验证

AB = 1: 利用密钥B进行验证

\_SecNr: 密钥扇区号, 必须要小于 16。

\_Nkey: 6 字节密钥首址

描述:

这个函数将一个新的密钥写入到 RC500 的只写 EEPROM 存储器的相应区中。

该函数的典型执行时间:

波特率 9600: 33.1ms

波特率 57600: 14.4ms

### 3.4.13 复位—Reset

声明:

```
uchar mifs_reset(uchar _Msec);
```

MCU ⇒ZLG500:

命令符: 0x4E

长度: 1

Data[0]: \_Msec

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, COMM\_ERR

长度: 0

参数:

\_Msec: 射频电路关闭时间（以毫秒为单位）

描述:

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。

举例:

_Msec = 0	⇒	∞	射频电路关闭
_Msec = 0x01	⇒	1 ms	射频电路关闭 1ms
_Msec= 0xFF	⇒	255 ms	射频电路关闭 255ms

该函数的典型执行时间: (\_Msec=1)

波特率 9600: 17.9ms

波特率 57600: 5.2ms

### 3.4.14 获取信息—Get Info

声明:

```
uchar mifs_get_info(uchar idata *_Info);
```

MCU ⇒ZLG500:

命令符: 0x4F

长度: 0

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, COMM\_ERR

长度: 10

Data[0]: 产品类型标识 0

::

Data[4]: 产品类型标识 4

Data[5]: RC500 序列号 0

::

Data[8]: RC500 序列号 3

Data[9]: 软件版本号

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  
\_Info[9]为固件版本号, 高四位为版本号的整数, 取值从 1 到 15, 低四位为版本号的小数, 取值从 0 到 9。

描述:

此函数返回一个包含有 RC500 的产品类型标识、序列号和软件版本信息的数组。

该函数的典型执行时间:

波特率 9600: 28.9ms

波特率 57600: 8.7ms

### 3.4.15 置位控制位—Set Control Bit

声明:

```
uchar mifs_set_control_bit();
```

MCU ⇒ZLG500:

命令符: 0x50

长度: 0

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, COMM\_ERR

长度: 0

描述:

此函数设置 MIFARE 读卡器中的控制位为高电平。

该函数的典型执行时间:

波特率 9600: 15.8ms

波特率 57600: 3.9ms

### 3.4.16 清除控制位—Clr Control Bit

声明:

```
uchar mifs_clr_control_bit();
```

MCU ⇒ZLG500:

命令符: 0x51

长度: 0

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, COMM\_ERR

长度: 0

描述:

此函数清除 MIFARE 读卡器中的控制位

该函数的典型执行时间:

波特率 9600: 15.8ms

波特率 57600: 3.9ms

### 3.4.17 配置—Config

声明:

```
uchar mifs_config(void);
```

MCU ⇒ZLG500:

命令符: 0x52

长度: 0

ZLG500⇒MCU:

状态值: MI\_OK, MI\_QUIT, COMM\_ERR

长度: 0

参数:

说明:

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。

该函数的典型执行时间:

波特率 9600: 44.9ms

波特率 57600: 33ms

### 3.4.18 检查写—Check Write

声明:

```
uchar mifs_check_write(uchar idata * _SNR, uchar _Authmode, uchar _Adr, uchar idata * _Data);
```

MCU ⇒ZLG500:

命令符:	0x53
长度:	22
Data[0]:	_SNR(LL)
Data[1]:	_SNR(LH)
Data[2]:	_SNR(HL)
Data[3]:	_SNR(HH)
Data[4]:	_Authmode
Data[5]:	_Adr
Data[6]:	块的第一个字节
::	
Data[21]:	块的最后一个字节

ZLG500⇒MCU:

状态值:	MI_QUIT, MIS_CHK_OK, MIS_CHK_FAILED, MIS_CHK_COMPERR, COMM_ERR
长度:	0

参数:

_SNR:	所要检查的卡的序号
_Authmode:	上一次写命令时的验证模式
_Adr:	所要检查的数据块的地址
_Data:	所检查的数据

描述:

此函数在数据写入卡的数据进行检查。将重新进行 Request/Select/Authenticated 操作。此函数进行将所给出的数据与相应地址的数据进行比较。如果正确，则返回 MIS\_CHK\_OK 信息。如果两者间数据不相符，则返回 MIS\_CHK\_COMPERR 信息。发生其它任何错误时，返回 MIS\_CHK\_FAILED 信息。

注：验证密钥时所用的密钥区与块\_Adr 所在的扇区号相同。

### 3.4.19 输出蜂鸣器信号—Buzzer

声明:

```
mifs_buzzer(uchar _Frquence, uchar _Opentm, uchar _Closetm, uchar _Repcnt);
```

MCU ⇒ZLG500:

```
命令符:    0x60
长度:      2
Data[0]:   _Frquence
Data[1]:   _Opentm
Data[2]:   _Closetm
Data[3]:   _Repcnt
```

ZLG500⇒MCU:

```
状态值:    MI_OK,  COMM_ERR
长度:      0
```

参数:

```
_Frquence:  输出方波频率, 取值(0—255), 对应频率(0.73—4K), 0为直流, 198对应2K
_Opentm:    方波输出持续时间, 取值(0—255), 10ms的分辨率
_Closetm:   间隙时间, 取值(0—255), 10ms的分辨率
_Repcnt:    重复次数
```

描述:

此函数输出一方波或低电平可驱动蜂鸣器, 频率、持续时间、间隙时间和重复次数可调。下面是频率对照表:

_Frquence	输出频率 (KHz)
0	输出低电平 (直流)
1	0.735
20	0.787
40	0.847
60	0.913
80	0.990
100	1.081
120	1.198
140	1.333
160	1.515
180	1.74
200	2.04
220	2.50
240	3.17
255	4.00

## 3.4.20 读 EEPROM

声明:

```
uchar mifs_read_E2(uchar _Adr,uchar _Length,uchar idata *_Data);
```

MCU ⇒ZLG500:

```
命令符:    0x61  
长度:      2  
Data[0]:   _Adr  
Data[1]:   _Length
```

ZLG500⇒MCU:

```
状态值:    MI_OK, MI_QUIT, MI_CRCERR, MI_BITCOUNTERR, COMM_ERR  
长度:      _Length  
Data[0]:   byte  
...  
Data[_Length]: byte
```

参数:

```
_Adr:      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
```

描述:

此函数将 RC500 内 EEPROM 的数据读出。

该函数的典型执行时间:

波特率 9600: 36.8ms (16 字节)

波特率 57600: 9.9ms (16 字节)

### 3.4.21 写 EEPROM

声明:

```
uchar mifs_write_E2(uchar _Adr,uchar _Length,uchar idata *_Data);
```

MCU ⇒ZLG500:

```
命令符:      0x62
长度:        _Length+2
Data[0]:     _Adr
Data[1]:     _Length
Data[2]:     _Data[0]
...
Data[_Length+1]: _Data[_Length-1]
```

ZLG500⇒MCU:

```
状态值:      MI_OK, MI_QUIT, MI_CRCERR, MI_BITCOUNTERR, COMM_ERR
长度:        0
```

参数:

```
_Adr:        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
```

描述:

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。

该函数的典型执行时间:

波特率 9600: 42.9ms (16 字节)

波特率 57600: 15.8ms (16 字节)

### 3.4.22 关闭 RC500—Close

声明:

```
uchar mifs_close(void);
```

MCU ⇒ZLG500:

命令符: 0x3F

长度: 0

ZLG500⇒MCU:

状态值: MI\_OK, COMM\_ERR

长度: 0

参数:

描述:

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()。

该函数的典型执行时间:

波特率 9600: 15.5ms

波特率 57600: 3.6ms

### 3.4.23 值操作

声明:

```
uchar mifs_value(uchar _Mode, uchar _Adr, ulong idata *_Value, uchar _Trans_Adr);
```

MCU ⇒ ZLG500:

命令符: 0x70  
长度: 7  
Data[0]: \_Mode  
Data[1]: \_Adr  
Data[2]: \_Value(LL)  
Data[3]: \_Value(LH)  
Data[4]: \_Value(HL)  
Data[5]: \_Value(HH)  
Data[6]: \_Trans\_Adr

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_CODE, MI\_BITCOUNTERR, MI\_TRANSERR,  
MI\_CODEERR, COMM\_RERR  
长度: 0

参数:

\_Mode: 0xC0—减  
0xC1—加  
0xC2—恢复  
\_Adr: 卡内块地址, 对该块进行值操作, 取值范围 0—63。  
\_Value: 当进行加或减操作时, 为加数或减数; 当进行恢复操作时, 该值为空值。  
\_Trans\_Adr: 传输块地址, 取值范围 0—63。

描述:

此函数对卡内的某一块进行加、减或数据备份, 该块必须为值块格式, 并支持自动传送。

该函数的典型执行时间:

波特率 9600: 38.3ms

波特率 57600: 20.6ms

## 3.4.24 防碰撞 2—Anticoll2

声明:

```
uchar mifs_anticoll2(uchar _Encoll, uchar _Bcnt, uchar idata * _SNR);
```

MCU ⇒ ZLG500:

```
命令符:    0x71
长度:      2
Data[0]:   _Encoll
Data[1]:   _Bcnt
```

ZLG500 ⇒ MCU:

```
状态值:    MI_OK, MI_NOTAGERR, MI_BITCOUNTERR, MI_COLLERR, COMM_ERR
长度:      4
Data[0]:   snr(LL)
Data[1]:   snr(LH)
Data[2]:   snr(HL)
Data[3]:   snr(HH)
```

参数:

```
_Encoll:    若为1, 则使能多张卡进入天线区; 若为0, 则不多张卡进入, 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:      卡的序列号。存储在一个无符号的四字节数组中, 低字节放在地址处。
```

描述:

此函数开始防冲突操作。必须在调用了Request命令后立即调用。当知道了所要选择卡的序列号后, 就没有必要调用AntiColl。此时, 调用了Request后, 直接调用Select函数即可。

该函数的典型执行时间:

波特率 9600: 25.1ms

波特率 57600: 8.3ms

### 3.4.25 证实 2—Authentication2

声明:

```
uchar mifs_authentication2(uchar _Mode, uchar _SecNr, uchar _KeyNr);
```

MCU ⇒ ZLG500:

命令符: 0x72  
长度: 3  
Data[0]: \_Mode  
Data[1]: \_SecNr  
Data[2]: \_KeyNr

ZLG500 ⇒ MCU:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_PAROTUERR, MI\_BITCOUNTERR,  
COMM\_ERR  
长度: 0

参数:

\_Mode: 

					AB		
--	--	--	--	--	----	--	--

AB = 0: 利用密钥A进行验证

AB = 1: 利用密钥B进行验证

\_SecNr: 所访问卡的扇区号

\_KeyNr: 用于证实的密匙区号, 必须小于16。

描述:

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RC500中所选择的密码相匹配。则证实成功, 函数将返回MI\_OK。

该函数的典型执行时间:

波特率 9600: 24.3ms

波特率 57600: 9.8ms

### 3.4.26 直接密码证实—AuthKey

声明:

```
uchar mifs_authentication2(uchar _Mode, uchar _SecNr, uchar *_Key);
```

MCU ⇒ ZLG500:

```
命令符:    0x73
长度:      8
Data[0]:   _Mode
Data[1]:   _SecNr
Data[2]:   _Key[0]
...
Data[7]:   _Key[5]
```

ZLG500 ⇒ MCU:

```
状态值:    MI_OK, MI_QUIT, MI_NOTAGERR, MI_PAROTUERR, MI_BITCOUNTErr,
           COMM_ERR
长度:      0
```

参数:

```
_Mode:     

|  |  |  |  |  |    |  |  |
|--|--|--|--|--|----|--|--|
|  |  |  |  |  | AB |  |  |
|--|--|--|--|--|----|--|--|


```

AB = 0: 利用密钥A进行验证

AB = 1: 利用密钥B进行验证

```
_SecNr:    所访问卡的扇区号
```

```
_Key:      用于证实的密码首址
```

描述:

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。

该函数的典型执行时间:

波特率 9600: 31.2ms

波特率 57600: 12.6ms

### 3.4.27 多层防碰撞—CascAnticoll

声明:

```
uchar mifs_Cascanticoll(uchar _Select_Code,uchar _Bcnt,uchar *_SNR);
```

主机 ⇒ 读卡器:

```
命令符:    0x74
长度:      2
Data[0]:   _Select_Code
Data[1]:   _Bcnt
```

读卡器 ⇒ 主机:

```
状态值:    MI_OK, MI_QUIT, MI_NOTAGERR, MI_BITCOUNTERR, COMM_ERR
长度:      4
Data[0]:   snr(LL)
Data[1]:   snr(LH)
Data[2]:   snr(HL)
Data[3]:   snr(HH)
```

参数:

```
_Select_Code: 防碰撞层级编码: 一层——0x93; 二层——0x95; 三层——0x97。
_Bcnt:        为预选卡所分配的位的个数, 通常Bcnt=0。
_SNR:         卡的序列号。存贮在一个无符号的四字节数组中, 低字节放在地址处。
```

描述:

此函数开始防冲突操作。必须在调用了Request命令后立即调用。当知道了所要选择卡的序列号后, 就没有必要调用AntiColl。此时, 调用了Request后, 直接调用Select函数即可。

### 3.4.28 多层选择—Select

声明:

```
uchar mifs_CascSelect(uchar _Select_Code, uchar *_SNR,uchar *_Sak);
```

主机 ⇒ 读卡器:

命令符: 0x75  
长度: 5  
Data[0]: \_Select\_Code  
Data[1]: snr(LL)  
Data[2]: snr(LH)  
Data[3]: snr(HL)  
Data[4]: snr(HH)

读卡器 ⇒ 主机:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_CRCERR, MI\_PARITYERR,  
MI\_BITCOUNTERR, COMM\_ERR  
长度: 1  
Data[0]: \_Sak

参数:

\_Select\_Code: 防碰撞层级编码: 一层——0x93; 二层——0x95; 三层——0x97。  
\_SNR: 卡的序号。存贮在一个无符号4字节字符数组中, 低字节放在地址处。  
\_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。

描述:

这个函数选择某一个序列号的卡, 返回ATS字节给主机。

### 3.4.29 写 UltraLight—ULWrite

描述:

```
uchar mifs_ULWrite(uchar _Adr, uchar *_Data);
```

主机 ⇒ 读卡器:

命令符:	0x76
长度:	5
Data[0]:	页地址
Data[1]:	所访问块的第一个字节
:	
Data[4]:	所访问块的最后一个字节

读卡器 ⇒ 主机:

状态值:	MI_OK, MI_QUIT, MI_NOTAGERR, MI_NOTAUTHERR, MI_WRITEERR, MI_BITCOUNTERR, COMM_ERR
长度:	0

参数:

_Adr:	所写数据块地址 (0—15)
_Data:	4 字节数据指针

描述:

写入一个4字节的数据。

### 3.4.30 带内部自动传送的值操作

声明:

```
uchar mifs_ValueDebit(uchar _Mode, uchar _Adr, ulong *_Value);
```

主机 ⇒读卡器:

命令符: 0x77  
长度: 6  
Data[0]: \_Mode  
Data[1]: \_Adr  
Data[2]: \_Value(LL)  
Data[3]: \_Value(LH)  
Data[4]: \_Value(HL)  
Data[5]: \_Value(HH)

读卡器⇒主机:

状态值: MI\_OK, MI\_QUIT, MI\_NOTAGERR, MI\_CODE, MI\_BITCOUNTEERR, MI\_TRANSERR,  
MI\_CODEERR, COMM\_RERR  
长度: 0

参数:

\_Mode: 0xC0—减 (Mifare Light 只支持减)  
0xC1—加  
0xC2—恢复  
\_Adr: 卡内块地址, 对该块进行值操作, 取值范围 0—63。  
\_Value: 当进行加或减操作时, 为加数或减数; 当进行恢复操作时, 该值为空值。

描述:

此函数对卡内的某一块进行加、减或数据备份, 该块必须为值块格式, 并支持内部自动传送。支持 Mifare Light。

### 3.4.31 写寄存器

声明:

```
uchar mifs_write_reg(uchar _Reg,uchar _Value);
```

主机 ⇒读卡器:

命令符: 0x3d

长度: 2

Data[0]: \_Reg

Data[1]: \_Value

读卡器⇒主机:

状态值: MI\_OK, COMM\_RERR

长度: 0

参数:

\_Reg: 寄存器地址

\_Value: 寄存器值

描述:

此函数可对 RC500 内的寄存器直接进行控制,在某些应用中是比较方便的。

例如:

- 1、向地址 0x11 写入值 0x59 或 0x5a 可关闭一个发送管脚,从而节省功耗;
- 2、向地址0x12写入0x01至0x3f之间的一个值,可以控制天线发送管脚的电导率,值越大电导率越高,功耗越大,读卡距离越远。

### 3.4.32 读寄存器

声明:

```
uchar mifs_read_reg(uchar _Reg,uchar *_Value);
```

主机 ⇒读卡器:

命令符: 0x3e

长度: 1

Data[0]: \_Reg

读卡器⇒主机:

状态值: MI\_OK, COMM\_RERR

长度: 1

Data[0]: 寄存器值

参数:

\_Reg: 寄存器地址

\*\_Value: 寄存器值地址

描述:

此函数可读 RC500 内的寄存器的值。

## 3.5 全局变量

在“ZLG500.C”中使用了二个全局变量。

一是通信数据缓冲区“ser\_buffer[26]”，所有的要通过串行接口发送和接收的数据都放在这里，包括序号、命令/状态、长度和BCC等，为了方便地访问这些变量，定义了如索引常量：

```
#define SEQNR      0
#define COMMAND   1
#define STATUS    1
#define LENGTH    2
#define DATA     3
```

二是标志位“newdata”，当外部中断1中断，接收到新数据时该位置位。

## 3.6 串行口看门狗定时器

定时器2作为串行口看门狗定时器。

发送STX后，启动定时器2为10毫秒溢出，若定时器2溢出，则本次数据链建立失败。

发送命令完毕后，启动定时器2为2毫秒溢出，若溢出计数达150次，共300毫秒内，ZLG500B未返回响应，则本次命令执行失败。

收到ZLG500B的STX后，必须在50ms内回应ACK，以后启动定时器2为10毫秒溢出，若10毫秒内，未接收到下一个字节，则本次接收数据失败。

## 3.7 应用程序举例

例子：读出RC500和MIFARE卡的序列号，然后选择一个卡，将该卡的20块初始化成值块，且备份到21块中，最后使该卡进入HALT状态，且驱动2KHz蜂鸣器响200毫秒。

将” ZLG500B.c”、及主程序文件” main.c” 放于同一项目中，然后在主程序中输入以下代码。

```
#define __SRC
#include "main.h"
#undef __SRC

#include "zlg500B.h"

sbit zlg500B_RST=P1^4;
uchar baud_num;
uchar card_snr[4];
uchar code Nkey_a[6] = {0xA0, 0xA1, 0xA2, 0xA3, 0xA4, 0xA5};
uchar code Nkey_b[6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

main()
{
    uchar idata tt[2];
    uchar idata size;
    uchar idata bankdata[16];
    ulong idata value=1;
    uchar i,j;

    zlg500_RST=1;                //使ZLG500复位
    for(i=255;i>0;i--)
        for(j=255;j>0;j--);
    zlg500_RST=0;
    for(i=255;i>0;i--)
        for(j=255;j>0;j--);
    baud_num=3;                  //波特率为57600
    serial_init();
    EA=1;
    i=mifs_config();             //ZLG500配置
    i=mifs_get_info(bankdata);    //读信息
    memcpy(RC500_snr,&bankdata[8],4); //存RC500序列号
    mifs_load_key(KEYA,5,Nkey_b); //装载密钥
    while(1)
    {
        while(mifs_request(IDLE,tt)!=0); //请求
        if(mifs_anticoll(0,card_snr)!=0) continue; //防碰撞
        if(mifs_select(card_snr,&size)!=0) continue; //选择
        if(mifs_authentication(KEYA,5)!=0) continue; //证实
    }
}
```

```
bankdata[0]=0x10;
bankdata[4]=~0x10;
bankdata[8]=0x10;
for(i=1;i<4;i++)
{
    bankdata[i]=0x00;
    bankdata[4+i]=0xff;
    bankdata[8+i]=0x00;
}
bankdata[12]=0x14;
bankdata[13]=~0x14;
bankdata[14]=0x14;
bankdata[15]=~0x14;
if(mifs_write(20,bankdata)!=0) continue; //写一个值块
if(mifs_check_write(card_snr,KEYA,20,bankdata)!=0) continue; //检查写
if(mifs_restore(20)!=0) continue; //恢复20块的数据
if(mifs_transfer(21)!=0) continue; //传送到21块
if(mifs_read(21,bankdata)!=0) continue; //读出
mifs_halt(); //使卡进入HALT状态
mifs_buzzer(198,20,0,1); //蜂鸣器口输出2KHz方波，持续200毫秒
}
}
```