FM1715 编程指南

Ver 1.0

2004 年 4 月

上海复旦微电子股份有限公司

一 概述

在此文档中用户可找到关于 FM1715 底层函数库的描述,使用这些函数库,可方便地使用 FM1715 模块访问 MIFARE 卡和上海标准卡.

此编程指南中提供的底层函数库是基于 MCS-51 的应用环境，采用 C51 编制，并在 KEIL C51 V6.0 编译环境下通过。

二、ISO14443A 基础

2.1 卡片返回的代码说明

- 0x00：对指定地址的访问被拒绝
- 0x01：CRC 或奇偶校验错误
- 0x04：
  - 交易：溢出错误
  - 其它命令:对指定地址的访问被拒绝
- 0x05：CRC 或奇偶校验错误
- 0x0A：确认

2.2 基本命令

2.2.1 REQUEST

- **控制单元 ⇒ 射频卡**
- Command： 0x26 or 0x52
  0x26： IDLE 模式，只选择天线范围内 IDLE 模式的卡片
  0x52： ALL 模式，选择天线范围内所有卡片
- Len： 0
- **射频卡 ⇒ 控制单元**
- *Len： 2*
- *Data[0]： _TagType（低字节）0x04*
- *Data[1]： _TagType（高字节）0x00*
- 在重新选择卡片时必须执行 request 操作。

2.2.2 ANTICOLL

- **控制单元 ⇒ 射频卡**
- Command： 0x93
- Len： 1
- Data[0]： 0x20 NVB
- **射频卡 ⇒ 控制单元**
- *Len: 5*
- *Data[0]: _Snr(LL)*
- *Data[1]: _Snr(LH)*

- *Data[2]: _Snr(HL) 卡片系列号*
- *Data[3]: _Snr(HH)*
- *Data[4]: BCC*
- 此操作必须紧随在 request 操作后执行.如果被选的卡片的系列号已知,可以不用执行此操作

2.2.3 SELECT

- **控制单元 ⇒ 射频卡**
- Command: 0x93
- Len: 6
- Data[0]: 0x70
- Data[1]: _Snr(LL)
- Data[2]: _Snr(LH)
- Data[3]: _Snn(HL)
- Data[4]: _Snr(HH) 卡片系列号(UID)
- Data[5]: BCC
- **射频卡 ⇒ 控制单元**
- *Len: 1*
- *Data[0]: _Size (卡片容量值：0x08 或 0x88)*

2.2.4 AUTHENTICATION

- **控制单元 ⇒ 射频卡**
- Command: 0x60 or 0x61
- Len: 2
- Data[0]: 0x60 or 0x61 (0x60 使用 KEYA 作验证，0x61 使 KEYB 作验证)
- Data[1]: _SecNr (扇区号) *4(即每个扇区的块 0 的块地址)
- **射频卡 ⇒ 控制单元**
- *Len: 0*
- 如果读写模块中的密码与卡片中的密码相匹配，则可以进行读、写等操作。

2.2.5 HALT

- **控制单元 ⇒ 射频卡**
- Command: 0x50
- Len: 0
- **射频卡 ⇒ 控制单元**
- *Len: 0*
- 将操作后的卡片置于 halt 模式。如果又要对卡片操作，必须重新执行 request 操作。

2.2.6 READ

- **控制单元 ⇒ 射频卡**
- Command:　0x30
- Len:　1
- Data[0]:　_Adr　块地址（0～63）
- **射频卡 ⇒ 控制单元**
- *Len: 16*
- *Data[0]: 数据块的第一字节*

  :
- *Data[15]：数据块的最后一个字节*

2.2.7 WRITE

- **控制单元 ⇒ 射频卡**
- Command:　0xA0
- Len:　17
- Data[0]:　_Adr　要写入数据的块地址（1～63）
- **射频卡 ⇒ 控制单元**
- *Len：4Bit*
- *DATA[0]: 0x0A(ACK)*
- Data[1]:　要写入卡片中的第一个数据
- :
- Data[16]:　要写入卡片中的最后一个数据
- **射频卡 ⇒ 控制单元**
- *Len: 4Bit*
- *DATA[0]: 0x0A(ACK)*

2.2.8 INCREMENT

- **控制单元 ⇒ 射频卡**
- Command:　0xC1
- Len:　5
- Data[0]:　_Adr　数值块的地址
- **射频卡 ⇒ 控制单元**
- *Len: 4Bit*
- *DATA[0]: 0x0A(ACK)*
- Data[1]:　_Value(LL)
- Data[2]:　_Value(LH)
- Data[3]:　_Value(HL)

- Data[4]: _Value(HH) 要增加的数值
- **射频卡 ⇒ 控制单元**
- *Len: 0*

2.2.9 DECREMENT

- **控制单元 ⇒ 射频卡**
- Command: 0xC0
- Len: 5
- Data[0]: _Adr 数值块的地址
- **射频卡 ⇒ 控制单元**
- *Len: 4Bit*
- *DATA[0]: 0x0A(ACK)*
- Data[1]: _Value(LL)
- Data[2]: _Value(LH)
- Data[3]: _Value(HL)
- Data[4]: _Value(HH) 要减少的数值
- **射频卡 ⇒ 控制单元**
- *Len: 0*

2.2.10 RESTORE

- **控制单元 ⇒ 射频卡**
- Command: 0xC2
- Len: 6
- Data[0]: _Adr 数值块的地址
- **射频卡 ⇒ 控制单元**
- *Len: 4Bit*
- *DATA[0]: 0x0A(ACK)*
- Data[1]: 0x00
- Data[2]: 0x00
- Data[3]: 0x00
- Data[4]: 0x00
- **射频卡 ⇒ 控制单元**
- *Len: 0*
- 此操作相当于执行 decrement(0)。

2.2.11 TRANSFER

- **控制单元 ⇒ 射频卡**

- Command: 0xB0
- Len: 1
- Data[0]: Adr 要传输数据的卡片块地址
- **射频卡 ⇒ 控制单元**
- *Len: 4Bit*
- *DATA[0]: 0x0A(ACK)*

三、FM1715 简介

3.1 自动侦测微处理器接口类型

在每一次上电或硬件复位后，FM1712/1714/1715 会复位并行微处理器接口模式，并且通过检测控制管脚上的电平来侦测当前的微处理器接口模式。

3.2 不同类型微处理器接口连接关系

FM1715和不同微处理器接口的连接见下表：

FM1712 **FM1714** **FM1715**	并行接口类型				
	独立的读/写选通模式		通用的读/写选通模式		
	独立的地址/数据总线	复用的地址/数据总线	独立的地址/数据总线	复用的地址/数据总线	握手联络方式下复用地址/数据总线
ALE	HIGH	ALE	HIGH	AS	nAStrb
A2	A2	LOW	A2	LOW	HIGH
A1	A1	HIGH	A1	HIGH	HIGH
A0	A0	HIGH	A0	LOW	nWait
NRD	NRD	NRD	NDS	NDS	nDStrb
NWR	NWR	NWR	R/NW	R/NW	nWrite
NCS	NCS	NCS	NCS	NCS	LOW
D7…D0	D7…D0	AD7…AD0	D7…D0	AD7…AD0	AD7…AD0

3.3 FM1715 寄存器

FM1715 的内部寄存器按功能不同分成 8 组，每组为一页，包含 8 个寄存器：

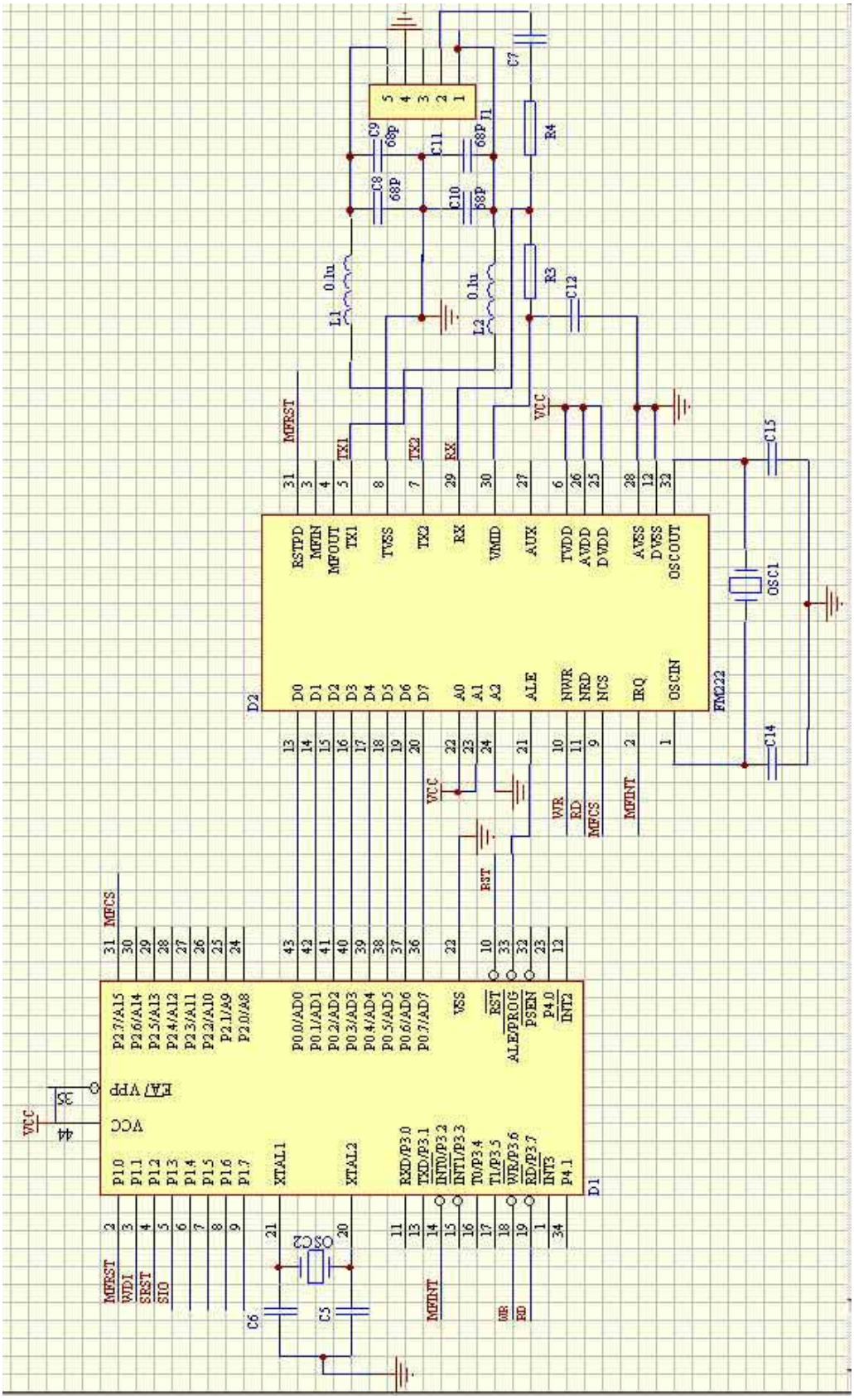
Page0：指令和状态寄存器组
Page1：控制和状态寄存器组
Page2：发射及编码控制寄存器组
Page3：接收及解码控制寄存器组
Page4：时间及校验控制寄存器组
Page5：FIFO，Timer 及 IRQ 控制寄存器组
Page6：预留寄存器组
Page7：预留寄存器组

Page	地址 (hex)	寄存器名	功能
0	0	Page	选择寄存器组
	1	Command	指令寄存器
	2	FIFOData	64byte FIFO 的输入输出寄存器
	3	PrimaryStatus	发射器，接收器及 FIFO 的标识位寄存器
	4	FIFOLength	当前 FIFO 内 byte 数
	5	SecondaryStatus	各种状态标识寄存器
	6	InterruptEn	中断使能/禁止控制寄存器
	7	InterruptRq	中断请求标识寄存器
1	8	Page	选择寄存器组
	9	Control	各种控制标识寄存器
	A	ErrorFlag	上一条指令结束后错误标识
	B	CollPos	侦测到的第一个冲突位的位置
	C	TimerValue	当前 Timer 值
	D	CRCResultLSB	CRC 协处理器低 8 位
	E	CRCResultMSB	CRC 协处理器高 8 位
	F	BitFraming	调整面向 bit 的帧格式
2	10	Page	选择寄存器组
	11	TxControl	发射器控制寄存器
	12	CWConductance	选择发射脚 TX1 和 TX2 发射天线的阻抗
	13	ModConductance	定义输出驱动阻抗
	14	CoderControl	定义编码模式和时钟频率
	15	ModWidth	选择载波调制宽度
	16	PreSet16	预设寄存器，不要改变内容
	17	TypeBFraming	定义 ISO14443-B 帧格式
3	18	Page	选择寄存器组
	19	RXControl1	接收器控制寄存器
	1A	DecoderControl	解码控制寄存器

	1B	BitPhase	调整发射器和接收器时钟相差
	1C	Rxthreshold	选择 bit 解码的阈值
	1D	BPSKDemControl	BPSK 接收器控制寄存器
	1E	RxControl2	解码控制及选择接收源
	1F	ClockQControl	时钟产生控制寄存器
4	20	Page	选择寄存器组
	21	RxWait	选择发射和接收之间的时间间隔
	22	ChannelRedundancy	选择数据校验种类和模式
	23	CRCPresetLSB	CRC 预置寄存器低 8 位
	24	CRCPresetMSB	CRC 预置寄存器高 8 位
	25	PreSet25	预设寄存器，不要改变内容
	26	MFOUTSelect	选择 MFOUT 信号源
	27	PreSet27	预设寄存器，不要改变内容
5	28	Page	选择寄存器组
	29	FIFOLevel	定义 FIFO 溢出级别
	2A	TimerClock	选择 Timer 时钟的分频
	2B	TimerControl	选择 Timer 启动/停止条件
	2C	TimerReload Timer	预置值
	2D	IRQPinConfig IRQ	IRQ 输出配置
	2E	PreSet2E	预设寄存器，不要改变内容
	2F	PreSet2F	预设寄存器，不要改变内容
6	30	Page	选择寄存器组
	31	CryptoSelect	认证模式选择
	32	RFU	预留寄存器
	33	RFU	预留寄存器
	34	RFU	预留寄存器
	35	RFU	预留寄存器
	36	RFU	预留寄存器
	37	RFU	预留寄存器
7	38	Page	选择寄存器组
	39	RFU	预留寄存器
	3A	RFU	预留寄存器
	3B	RFU	预留寄存器
	3C	RFU	预留寄存器
	3D	RFU	预留寄存器
	3E	RFU	预留寄存器
	3F	RFU	预留寄存器

关于 FM1715 内部寄存器的详细描述请参阅《FM1712_1714_1715 中文说明书.pdf》文档。

四、典型应用电路


D1
D2
J1
L1
L2
C5
C6
C7
C8
C9
C10
C11
C12
C14
C15
R3
R4
OSC1
OSC2
VCC
RST
MFRST
MFCS
MFINT
RSTPD
MFIN
MFOUT
TX1
TVSS
TX2
RX
VMID
AUX
TVDD
AVDD
DVDD
AVSS
DVSS
OSCOUT
OSCIN
IRQ
ALE
NWR
NRD
NCS
WR
RD
MFCS
MFINT
FM22
EA/VPP
XTAL1
XTAL2
ALE/PROG
PSEN
WDI
SRST
SIO

五、底层函数库

5.1 头文件

```
/*****************************************************************/
/*    main 程序头文件                                            */
/*    主要功能:常量定义                                          */
/*    编制:                                                      */
/*    时间:                                                      */
/*    修改:wangwenqing                                           */
/*    时间:2003 年 9 月                                          */
/*****************************************************************/

#ifndef _MAIN_INCLUDED_
#define _MAIN_INCLUDED_

//常量定义
#define FALSE                0
#define    TRUE              1

//FM1715 命令码
#define    Transceive        0x1E      //发送接收命令
#define Transmit             0x1a      //发送命令
#define ReadE2               0x03      //读 FM1715 EEPROM 命令
#define WriteE2              0x01      //写 FM1715 EEPROM 命令
#define Authent1             0x0c      //验证命令认证过程第 1 步
#define Authent2             0x14      //验证命令认证过程第 2 步
#define LoadKeyE2            0x0b      //将密钥从 EEPROM 复制到 KEY 缓存
#define LoadKey              0x19      //将密钥从 FIFO 缓存复制到 KEY 缓存

#define    RF_TimeOut        0xfff     //发送命令延时时间
#define Req                  0x01
#define Sel                  0x02

//数据类型定义
#define uchar unsigned char
#define uint unsigned int

//卡片类型定义定义
#define TYPEA_MODE           0         //TypeA 模式
#define TYPEB_MODE           1         //TypeB 模式
#define SHANGHAI_MODE        2         //上海模式

#define TM0_HIGH             0xf0      //定时器 0 高位,4MS 定时
#define TM0_LOW              0x60      //定时器 0 低位
#define TIMEOUT              100       //超时计数器 4MS×100=0.4 秒

//射频卡通信命令码定义
```

```
#define   RF_CMD_REQUEST_STD  0x26
#define   RF_CMD_REQUEST_ALL  0x52
#define   RF_CMD_ANTICOL      0x93
#define   RF_CMD_SELECT       0x93
#define   RF_CMD_AUTH_LA      0x60
#define   RF_CMD_AUTH_LB      0x61
#define   RF_CMD_READ         0x30
#define   RF_CMD_WRITE        0xa0
#define   RF_CMD_INC          0xc1
#define   RF_CMD_DEC          0xc0
#define   RF_CMD_RESTORE      0xc2
#define   RF_CMD_TRANSFER     0xb0
#define   RF_CMD_HALT         0x50

//Status Values
#define   ALL                 0x01
#define   KEYB                0x04
#define   KEYA                0x00
#define   _AB                 0x40
#define   CRC_A               1
#define   CRC_B               2

#define   CRC_OK              0
#define   CRC_ERR             1

#define   BCC_OK              0
#define   BCC_ERR             1

//******************** 卡类型定义**********************
#define   MIFARE_8K           0         //MIFARE 系列 8KB 卡片
#define   MIFARE_TOKEN        1         //MIFARE 系列 1KB TOKEN 卡片
#define   SHANGHAI_8K         2         //上海标准系列 8KB 卡片
#define   SHANGHAI_TOKEN      3         //上海标准系列 1KB TOKEN 卡片

//******************** 函数错误代码定义**********************
#define   FM1715_OK           0         //正确
#define   FM1715_NOTAGERR     1         //无卡
#define   FM1715_CRCERR       2         //卡片 CRC 校验错误
#define   FM1715_EMPTY        3         //数值溢出错误
#define   FM1715_AUTHERR      4         //验证不成功
#define   FM1715_PARITYERR    5         //卡片奇偶校验错误
#define   FM1715_CODEERR      6         //通讯错误(BCC 校验错)
#define   FM1715_SERNRERR     8         //卡片序列号错误(anti-collision  错误)
#define   FM1715_SELECTERR    9         //卡片数据长度字节错误(SELECT 错误)
#define   FM1715_NOTAUTHERR   10        //卡片没有通过验证
#define   FM1715_BITCOUNTERR  11        //从卡片接收到的位数错误
#define   FM1715_BYTECOUNTERR 12        //从卡片接收到的字节数错误（仅读函数
有效）
#define   FM1715_RESTERR      13        //调用 restore 函数出错
#define   FM1715_TRANSERR     14        //调用 transfer 函数出错
#define   FM1715_WRITEERR     15        //调用 write 函数出错
#define   FM1715_INCRERR      16        //调用 increment 函数出错
#define   FM1715_DECRERR      17        //调用 decrement 函数出错
```

```
#define   FM1715_READERR          18          //调用 read 函数出错
#define   FM1715_LOADKEYERR       19          //调用 LOADKEY 函数出错
#define   FM1715_FRAMINGERR       20          //FM1715 帧错误
#define   FM1715_REQERR           21          //调用 req 函数出错
#define   FM1715_SELERR           22          //调用 sel 函数出错
#define   FM1715_ANTICOLLERR      23          //调用 anticoll 函数出错
#define   FM1715_INTIVALERR       24          //调用初始化函数出错
#define   FM1715_READVALERR       25          //调用高级读块值函数出错
#define   FM1715_DESELECTERR      26
#define   FM1715_CMD_ERR          42          //命令错误

//******************** FM1715 地址定义*********************
#define   Page_Sel                XBYTE[0x7000]   //页写寄存器
#define   Command                 XBYTE[0x7001]   //命令寄存器
#define   FIFO                    XBYTE[0x7002]   //64 字节 FIFO 缓冲的输入输出寄存器
#define   PrimaryStatus           XBYTE[0x7003]   //发射器，接收器及 FIFO 的状态寄存器 1
#define   FIFO_Length             XBYTE[0x7004]   //当前 FIFO 内字节数寄存器
#define   SecondaryStatus         XBYTE[0x7005]   //各种状态寄存器 2
#define   InterruptEn             XBYTE[0x7006]   //中断使能/禁止寄存器
#define   Int_Req                 XBYTE[0x7007]   //中断请求标识寄存器

#define   Control                 XBYTE[0x7009]   //控制寄存器
#define   ErrorFlag               XBYTE[0x700A]   //错误状态寄存器
#define   CollPos                 XBYTE[0x700B]   //冲突检测寄存器
#define   TimerValue              XBYTE[0x700c]   //定时器当前值
#define   Bit_Frame               XBYTE[0x700F]   //位帧调整寄存器

#define   TxControl               XBYTE[0x7011]   //发送控制寄存器
#define   CWConductance           XBYTE[0x7012]   //选择发射脚 TX1 和 TX2 发射天线的阻抗
#define   ModConductance          XBYTE[0x7013]   //定义输出驱动阻抗
#define   CoderControl            XBYTE[0x7014]   //定义编码模式和时钟频率
#define   TypeBFraming            XBYTE[0x7017]   //定义 ISO14443B 帧格式

#define   DecoderControl          XBYTE[0x701a]   //解码控制寄存器
#define   Rxcontrol2              XBYTE[0x701e]   //解码控制及选择接收源

#define   RxWait                  XBYTE[0x7021]   //选择发射和接收之间的时间间隔
#define   ChannelRedundancy       XBYTE[0x7022]   //RF 通道检验模式设置寄存器
#define   CRCPresetLSB            XBYTE[0x7023]
#define   CRCPresetMSB            XBYTE[0x7024]
#define   MFOUTSelect             XBYTE[0x7026]   //mf OUT  选择配置寄存器

#define   TimerClock              XBYTE[0x702a]   //定时器周期设置寄存器
#define   TimerControl            XBYTE[0x702b]   //定时器控制寄存器
#define   TimerReload             XBYTE[0x702c]   //定时器初值寄存器

#define   TypeSH                  XBYTE[0x7031]   //上海标准选择寄存器
#define   TestDigiSelect          XBYTE[0x703d]   //测试管脚配置寄存器
#endif

sbit  MFRST = P1 ^ 0;                             //FM1715 复位管脚定义
```

```
uchar idata readdata[16]        _at_ 0x0040;    //读写数据缓冲区
uchar idata value[4]            _at_ 0x0050;    //增减的数值
uchar idata KeySet;                             //密码类型
uchar idata tagtype[2]          _at_ 0x0096;    //卡片标识字符

//******************** FM1715 变量定义*********************
uchar idata PRO_SendBuf[16]     _at_ 0x0080;    //发送处理缓冲区  16 BYTE
uchar idata PRO_RecvBuf[16]     _at_ 0x0080;    //接收处理缓冲区  16 BYTE
uchar idata buffer[24]          _at_ 0x0060;    //FM1715 命令发送接收缓冲区
uchar idata      UID[5]         _at_ 0x007a;    //序列号
uchar idata Secnr               _at_ 0x0090;    //扇区号
```

5.2 常用函数

5.2.1 卡片复位应答信号的判断

```
/*****************************************************************/
/*名称: Judge_Req                                                */
/*功能: 该函数实现对卡片复位应答信号的判断                        */
/*输入: *buff, 指向应答数据的指针                                 */
/*输出:   TRUE, 卡片应答信号正确                                  */
/*          FALSE, 卡片应答信号错误                               */
/*****************************************************************/
uchar Judge_Req(uchar idata *buff)
{
   uchar temp1,temp2;

   temp1 = *buff;
   temp2 = *(buff + 1);
   if(((temp1 == 0x03)||(temp1 == 0x04)||(temp1 == 0x05)||(temp1 == 0x53))&&(temp2 == 0x00))
   {
      return TRUE;
   }
   return FALSE;
}
```

注：temp1 = 03 ; 上海标准 TOKEN 卡
temp1 = 04 ; MIFARE 标准 8K
temp1 = 05 ; MIFARE 标准 TOKEN 卡
temp1 = 53 ; 上海标准 8K 卡
此返回值为目前市场使用的各类卡片的返回结果，将来可能有更多的返回类别。

5.2.2 接收到的卡片 UID 号的判别

```
/*****************************************************************/
/*名称: Check_UID                                                */
/*功能: 该函数实现对收到的卡片的序列号的判断                      */
/*输入: N/A                                                      */
/*输出:   TRUE: 序列号正确                                       */
/*          FALSE: 序列号错误                                    */
/*****************************************************************/
uchar Check_UID(void)
{
```

```
    uchar temp;
    uchar i;

    temp = 0x00;
    for(i = 0;i < 5; i++)
    {
        temp = temp ^ UID[i];
    }
    if(temp == 0)
    {
        return TRUE;
    }
    return FALSE;
}
```

5.2.3 保存卡片的 UID 号

```
/*****************************************************************/
/*名称: Save_UID                                                 */
/*功能: 该函数实现保存卡片收到的序列号                           */
/*输入:   row: 产生冲突的行                                      */
/*          col: 产生冲突的列                                    */
/*          length: 接收到的 UID 数据长度                        */
/*输出:    N/A                                                   */
/*****************************************************************/
void Save_UID(uchar row,uchar col,uchar length)
{
    uchar i;
    uchar temp;
    uchar temp1;

    if ((row == 0x00) && (col == 0x00))
    {
        for(i = 0; i < length; i++)
        {
            UID[i] = buffer[i];
        }
    }
    else
    {
        temp = buffer[0];
        temp1 = UID[row - 1];
        switch (col)
```

```
        {
            case 0:
                temp1 = 0x00;
                row = row + 1;
                break;
            case 1:
                temp = temp & 0xFE;
                temp1 = temp1 & 0x01;
                break;
            case 2:
                temp = temp & 0xFC;
                temp1 = temp1 & 0x03;
                break;
            case 3:
                temp = temp & 0xF8;
                temp1 = temp1 & 0x07;
                break;
            case 4:
                temp = temp & 0xF0;
                temp1 = temp1 & 0x0F;
                break;
            case 5:
                temp = temp & 0xE0;
                temp1 = temp1 & 0x1F;
                break;
            case 6:
                temp = temp & 0xC0;
                temp1 = temp1 & 0x3F;
                break;
            case 7:
                temp = temp & 0x80;
                temp1 = temp1 & 0x7F;
                break;
            default:
                break;
        }
        buffer[0] = temp;
        UID[row - 1] = temp1 | temp;
        for(i = 1; i < length; i++)
        {
            UID[row - 1 + i] = buffer[i];
        }
    }
}
```

5.2.4 设置待发送数据的字节数

```
/*****************************************************************************/
/*名称: Set_BitFraming                                                       */
/*功能: 该函数设置待发送数据的字节数                                         */
/*输入:   row: 产生冲突的行                                                  */
/*        col: 产生冲突的列                                                  */
/*输出:   N/A                                                                */
/*****************************************************************************/
void Set_BitFraming(uchar row,uchar col)
{
    switch (row)
    {
        case 0:
            buffer[1] = 0x20;
            break;
        case 1:
            buffer[1] = 0x30;
            break;
        case 2:
            buffer[1] = 0x40;
            break;
        case 3:
            buffer[1] = 0x50;
            break;
        case 4:
            buffer[1] = 0x60;
            break;
        default:
            break;
    }

    switch(col)
    {
        case 0:
            Bit_Frame = 0x00;
            break;
        case 1:
            Bit_Frame = 0x11;
            buffer[1] = (buffer[1] | 0x01);
            break;
        case 2:
            Bit_Frame = 0x22;
```

```
                buffer[1] = (buffer[1] | 0x02);
                break;
            case 3:
                Bit_Frame = 0x33;
                buffer[1] = (buffer[1] | 0x03);
                break;
            case 4:
                Bit_Frame = 0x44;
                buffer[1] = (buffer[1] | 0x04);
                break;
            case 5:
                Bit_Frame = 0x55;
                buffer[1] = (buffer[1] | 0x05);
                break;
            case 6:
                Bit_Frame = 0x66;
                buffer[1] = (buffer[1] | 0x06);
                break;
            case 7:
                Bit_Frame = 0x77;
                buffer[1] = (buffer[1] | 0x07);
                break;
            default:
                break;
        }
}
```

5.3 FM1715 基本函数

5.3.1 总线选择

```
/*****************************************************************/
/*名称: FM1715_Bus_Sel                                           */
/*功能: 该函数实现对 FM1715 操作的总线方式(并行总线,SPI)选择     */
/*输入: N/A                                                      */
/*输出:   TRUE,   总线选择成功                                   */
/*        FALSE,  总线选择失败                                   */
/*****************************************************************/
uchar FM1715_Bus_Sel(void)
{
   uchar i;
   Page_Sel = 0x80;                    //表示PageSelect的值做为寄存器地址A5，A4 和A3，低
                                       //三位寄存器地址A2－A0 由//外部地址线A2－A0 决定
   for(i = 0; i < RF_TimeOut; i++)     //延时
   {
      if(Command == 0x00)              //读命令执行结果, bit7为0表示接口检测结束
      {
         Page_Sel = 0x00;
         return TRUE;
      }
   }
   return FALSE;
}
```

5.3.2 FM1715 初始化

```
/*****************************************************************/
/*名称: Init_FM1715                                              */
/*功能: 该函数实现对 FM1715 初始化操作                           */
/*输入: mode:工作模式,      0:TYPEA 模式                         */
/*                          1:TYPEB 模式                         */
/*                          2:上海模式                           */
/*输出:   N/A                                                    */
/*****************************************************************/
void Init_FM1715(uchar mode)
{
   uchar   idata    temp;
   uint     i;

   MFRST = 1;                                  //FM1715 复位
   for (i = 0; i < 0x1fff; i++)
   {
      _nop_();
   }
   MFRST = 0;
   for (i = 0; i < 0x1fff; i++)
```

```
    {
        _nop_();
    }

    while(Command != 0)                    //等待 Command = 0,FM1715 复位成功
    {
        _nop_();
    }

    FM1715_Bus_Sel();                      //FM1715 总线选择
    TimerClock = 0x0b;                     //151us/per
    TimerControl = 0x02;                   //发送结束开定时器，接收开始关定时器
    TimerReload = 0x42;                    //10ms 定时
    InterruptEn = 0x7f;                    //关所有中断
    temp = InterruptEn;
    Int_Req = 0x7f;
    MFOUTSelect = 0x02;                    //调试用
    TxControl = 0x5b;                      //开启 TX1、TX2
    if (mode == SHANGHAI_MODE)             //上海模式
    {
        TypeSH = 0x01;
    }
    else
    {
        TypeSH = 0x00;
    }

//Rxcontrol2=0x01;
    if(mode == TYPEB_MODE)                 //TYPEB 模式
    {
        CoderControl = 0x20;
        TypeBFraming = 0x05;
        DecoderControl = 0x19;
        ChannelRedundancy = 0x24;
        TxControl = 0x4b;
        CWConductance = 0x3f;
        ModConductance = af;
    }
}
```

5.3.3 命令传输

```
/*****************************************************************/
/*名称: Command_Send                                             */
/*功能: 该函数实现向 FM1715 发送命令集的功能                     */
/*输入: count, 待发送命令集的长度                                */
/*       buff, 指向待发送数据的指针                              */
/*       Comm_Set, 命令码                                        */
/*输出: TRUE, 命令被正确执行                                     */
/*       FALSE, 命令执行错误                                     */
/*****************************************************************/
uchar Command_Send(uchar count, uchar idata * buff, uchar Comm_Set)
```

```
{
   uint    j;
   uchar idata temp,temp1;
//Int_Req=0x7f;
   Command = 0x00;
   Clear_FIFO();
   Write_FIFO(count, buff);
//Rxcontrol2=0x01;
   temp = MFOUTSelect;
   Command = Comm_Set;                                  //命令执行
   for(j = 0; j< RF_TimeOut; j++)                       //检查命令执行否
   {
      temp = MFOUTSelect;
      temp = Command;
      temp1 = Int_Req & 0x80;
      if((temp == 0x00) || (temp1 == 0x80))
      {
         //Rxcontrol2=0x41;
         return TRUE;
      }
   }
//Rxcontrol2=0x41;
   return FALSE;
}
```

5.3.4 读 FM1715 中的 EEPROM 数据

```
/*********************************************************************/
/*名称: Read_E2                                                      */
/*功能: 该函数实现从 FM1715 的 EE 中读出数据                         */
/*输入:   lsb, EE 地址(低字节)                                       */
/*          msb, EE 地址(高字节)                                     */
/*           count, 待读出数据 EE 的字节个数                         */
/*           buff, 指向待读出数据的指针                              */
/*输出:   TRUE, EE 数据正确读出                                      */
/*          FALSE, EE 数据读出有误                                   */
/*********************************************************************/
uchar      Read_E2(uchar lsb,uchar msb,uchar count,uchar idata *buff)
{
      uchar temp;

      *buff=lsb;
      *(buff+1)=msb;
      *(buff+2)=count;
      temp=Command_Send(3,buff,ReadE2);
      Read_FIFO(buff);
      if (temp==FALSE)
            return(TRUE);
      return(FALSE);
}
```

5.3.5 向 FM1715 的 EEPROM 中写入数据

```
/*****************************************************************/
/*名称: Write_E2                                                 */
/*功能: 该函数实现向 FM1715 的 EE 中写入数据                     */
/*输入:    lsb, EE 地址(低字节)                                  */
/*          msb, EE 地址(高字节)                                 */
/*          count, 待写入数据 EE 的字节个数                      */
/*          buff, 指向待写入数据的指针                           */
/*输出:    TRUE, EE 数据正确写入                                 */
/*          FALSE, EE 数据写入有误                               */
/*****************************************************************/
uchar      Write_E2(uchar lsb,uchar msb,uchar count,uchar idata *buff)
{
      uchar idata temp,i;

      for(i = 0;i < count; i++)
      {
            *(buff + count - i + 2) = *(buff - i + count);
      }
      *buff = lsb;
      *(buff + 1) = msb;
      temp = Command_Send(count + 2, buff, WriteE2);
      temp = SecondaryStatus;
      temp = temp & 0x40;
      if (temp == 0x40)
      {
            return TRUE;
      }
      return FALSE;
}
```

5.3.6 清除 FM1715 的 FIFO 中的数据

```
/*****************************************************************/
/*名称: Clear_FIFO                                               */
/*功能: 该函数实现清空 FM1715 中 FIFO 的数据                     */
/*输入: N/A                                                      */
/*输出: TRUE, FIFO 被清空                                        */
/*          FALSE, FIFO 未被清空                                 */
/*****************************************************************/
uchar Clear_FIFO(void)
{
      uchar temp;
      uint    i;

      temp = Control;                                 //清空 FIFO
      temp = (temp | 0x01);
      Control = temp;
      for(i = 0; i < RF_TimeOut; i++)                 //检查 FIFO 是否被清空
      {
```

```
            temp = FIFO_Length;
            if(temp == 0)
            {
                return TRUE;
            }
        }
        return FALSE;
}
```

5.3.7 向 FM1715 的 FIFO 中写入 x 字节数据

```
/*****************************************************************/
/*名称: Write_FIFO                                                */
/*功能: 该函数实现向 FM1715 的 FIFO 中写入 x bytes 数据           */
/*输入: count, 待写入字节的长度                                   */
/*        buff, 指向待写入数据的指针                              */
/*输出: N/A                                                       */
/*****************************************************************/
void Write_FIFO(uchar count,uchar idata *buff)
{
    uchar i;

    for(i = 0; i < count; i++)
    {
        FIFO = *(buff + i);
    }
}
```

5.3.8 从 FM1715 的 FIFO 中读出 x 字节数据

```
/*****************************************************************/
/*名称: Read_FIFO                                                 */
/*功能: 该函数实现从 FM1715 的 FIFO 中读出 x bytes 数据           */
/*输入:   buff, 指向读出数据的指针                                */
/*输出:   N/A                                                     */
/*****************************************************************/
uchar Read_FIFO(uchar idata *buff)
{
    uchar temp;
    uchar i;

    temp = FIFO_Length;
    if (temp == 0)
    {
        return 0;
    }
    if (temp >= 24)                    //temp=255 时,会进入死循环
    {                                  //因此增加 FIFO_Length 越限判断
        temp = 24;                     //yanshouli,2003-12-2
    }
    for(i = 0;i < temp; i++)
```

```
    {
        *(buff + i) = FIFO;
    }
    return temp;
}
```

5.4 FM1715 卡片操作基本函数

5.4.1 HALT

```
/*****************************************************************/
/*名称: MIF_Halt                                                 */
/*功能: 该函数实现暂停 MIFARE 卡                                 */
/*输入: N/A                                                      */
/*输出:   FM1715_OK: 应答正确                                    */
/*        FM1715_PARITYERR: 奇偶校验错                           */
/*        FM1715_CRCERR: CRC 校验错                              */
/*        FM1715_NOTAGERR: 无卡                                  */
/*****************************************************************/
uchar MIF_Halt(void)
{
   uchar temp;
   uint     i;
   CRCPresetLSB = 0x63;
   CWConductance = 0x3f;
   ChannelRedundancy = 0x03;
   *buffer = RF_CMD_HALT;
   *(buffer + 1) = 0x00;
   temp = Command_Send(2, buffer, Transmit);
   if (temp == TRUE)
   {
      for (i = 0; i < 0x50; i++)
      {
         _nop_();
      }
      return FM1715_OK;
   }
   else
   {
      temp = ErrorFlag;
      if ((temp & 0x02)==0x02)
      {
         return(FM1715_PARITYERR);
      }
      if ((temp & 0x04)==0x04)
      {
         return(FM1715_FRAMINGERR);
      }
      return(FM1715_NOTAGERR);
   }
}
```

5.4.2 LOADKEY

```
/*****************************************************************/
/*名称: Load_keyE2                                               */
/*功能: 该函数实现把 E2 中密码存入 FM1715 的 keybuffer 中        */
/*输入: Secnr: EE 起始地址                                       */
/*输出: True: 密钥装载成功                                       */
/*       False: 密钥装载失败                                     */
/*****************************************************************/
uchar     Load_keyE2_CPY(uchar Secnr,uchar Mode)
{
   uchar temp;
   uchar msb = 0;
   uchar lsb = 0;

   temp = Secnr * 12;
   if (Mode == 0)
   {
      if (temp >= 0x80)                          //计算密码存放地址
      {
         lsb = temp - 0x80;
         msb = 0x01;
      }
      else
      {
         msb = 0x00;
         lsb = temp + 0x80;
      }
   }
   else
   {
      msb = 0x01;
      lsb = temp + 0x40;
   }
   buffer[0] = lsb;
   buffer[1] = msb;
   temp = Command_Send(2, buffer, LoadKeyE2);
   temp = ErrorFlag & 0x40;
   if (temp == 0x40)
   {
      return FALSE;
   }
   return TRUE;
}
```

5.4.3 REQUEST

```
/*****************************************************************/
/*名称: Request                                                  */
/*功能: 该函数实现对放入 FM1715 操作范围之内的卡片的 Request 操作  */
/*输入:   mode: ALL(监测所以 FM1715 操作范围之内的卡片)          */
/*          STD(监测在 FM1715 操作范围之内处于 HALT 状态的卡片)  */
/*输出:   FM1715_NOTAGERR: 无卡                                  */
/*          FM1715_OK: 应答正确                                  */
/*          FM1715_REQERR: 应答错误                              */
/*****************************************************************/
uchar Request(uchar mode)
{
   uchar    idata temp;

   CRCPresetLSB = 0x63;
   CWConductance = 0x3f;
   buffer[0] = mode;                                  //Request 模式选择
   Bit_Frame = 0x07;                                  //发送 7bit
   ChannelRedundancy = 0x03;                          //关闭 CRC
   temp = Control;
   temp = temp & (0xf7);
   Control = temp;                                    //屏蔽 CRYPTO1 位
   temp = Command_Send(1, buffer, Transceive);
   if(temp == FALSE)
   {
      return FM1715_NOTAGERR;
   }
   Read_FIFO(buffer);                                 //从 FIFO 中读取应答信息
   temp = Judge_Req(buffer);                          //判断应答信号是否正确
   if (temp == TRUE)
   {
      tagtype[0] = buffer[0];
      tagtype[1] = buffer[1];
      return FM1715_OK;
   }
   return FM1715_REQERR;
}
```

5.4.4 ANTICOLLISION

```
/*****************************************************************/
/*名称: AntiColl                                                 */
/*功能: 该函数实现对放入 FM1715 操作范围之内的卡片的防冲突检测   */
/*输入:   N/A                                                    */
/*输出:   FM1715_NOTAGERR: 无卡                                  */
/*          FM1715_BYTECOUNTERR: 接收字节错误                    */
/*    FM1715_SERNRERR: 卡片序列号应答错误                        */
/*    FM1715_OK: 卡片应答正确                                    */
/*****************************************************************/
```

```
uchar AntiColl(void)
{
    uchar temp;
    uchar i;
    uchar row,col;
    uchar pre_row;

    row = 0;
    col = 0;
    pre_row = 0;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    ModConductance = 0X3f;
    buffer[0] = RF_CMD_ANTICOL;
    buffer[1] = 0x20;
    ChannelRedundancy = 0x03;                          //关闭 CRC,打开奇偶校验
    temp=Command_Send(2, buffer, Transceive);

    while(1)
    {
        if(temp==FALSE)
        {
            return(FM1715_NOTAGERR);
        }
        temp = ErrorFlag;
//      if((temp & 0x02)==0x02)
//          return(FM1715_PARITYERR);
//      if((temp & 0x04)==0x04)
//          return(FM1715_FRAMINGERR);

        temp=FIFO_Length;
        if (temp==0)
        {
            return FM1715_BYTECOUNTERR;
            }
            Read_FIFO(buffer);
            Save_UID(row, col, temp);                  //将收到的 UID 放入 UID 数组中
            Show_UID();                                //显示 UID
            temp = ErrorFlag;                          //判断接收数据是否出错
            temp = temp & 0x01;
            if(temp == 0x00)
            {
                temp = Check_UID();                    //校验收到的 UID
                if(temp == FALSE)
                {
                    return(FM1715_SERNRERR);
                }
                return(FM1715_OK);
            }
            else
            {
                temp = CollPos;                        //读取冲突检测寄存器
                row = temp / 8;
                col = temp % 8;
```

```
                buffer[0] = RF_CMD_ANTICOL;
                Set_BitFraming(row + pre_row, col);              //设置待发送数据的字节数
                pre_row = pre_row + row;
                for(i = 0; i < pre_row + 1; i++)
                {
                    buffer[i + 2] = UID[i];
                }
                if(col != 0x00)
                {
                    row = pre_row + 1;
                }
                else
                {
                    row = pre_row;
                }
                temp = Command_Send(row + 2, buffer, Transceive);
            }
        }
}
```

5.4.5 SELECT

```
/*****************************************************************/
/*名称: Select_Card                                               */
/*功能: 该函数实现对放入 FM1715 操作范围之内的某张卡片进行选择    */
/*输入:   N/A                                                     */
/*输出:   FM1715_NOTAGERR: 无卡                                   */
/*        FM1715_PARITYERR: 奇偶校验错                            */
/*        FM1715_CRCERR: CRC 校验错                               */
/*        FM1715_BYTECOUNTERR: 接收字节错误                       */
/*        FM1715_OK: 应答正确                                     */
/*        FM1715_SELERR: 选卡出错                                 */
/*****************************************************************/
uchar Select_Card(void)
{
    uchar temp,i;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    buffer[0] = RF_CMD_SELECT;
    buffer[1] = 0x70;
    for(i = 0; i < 5; i++)
    {
        buffer[i+2]=UID[i];
    }
    ChannelRedundancy=0x0f;                                      //开启 CRC,奇偶校验校验
    temp=Command_Send(7,buffer,Transceive);

    if(temp==FALSE)
    {
        return(FM1715_NOTAGERR);
    }
    else
```

```
    {
        temp=ErrorFlag;
        if ((temp & 0x02)==0x02)
            return(FM1715_PARITYERR);
        if((temp & 0x04)==0x04)
            return(FM1715_FRAMINGERR);
        if ((temp & 0x08)==0x08)
            return(FM1715_CRCERR);

        temp=FIFO_Length;
        if (temp!=1)
            return(FM1715_BYTECOUNTERR);
        Read_FIFO(buffer);                                  //从 FIFO 中读取应答信息
        temp = *buffer;
        if ((temp==0x08) || (temp==0x88) || (temp==0x53)) //判断应答信号是否正确
            return(FM1715_OK);
        else
            return(FM1715_SELERR);
    }
}
```

5.4.6 AUTHENTICATION

```
/*****************************************************************/
/*名称: Authentication                                           */
/*功能: 该函数实现密码认证的过程                                 */
/*输入: UID: 卡片序列号地址                                      */
/*       SecNR: 扇区号                                           */
/*       mode: 模式                                              */
/*输出:  FM1715_NOTAGERR: 无卡                                   */
/*       FM1715_PARITYERR: 奇偶校验错                            */
/*       FM1715_CRCERR: CRC 校验错                               */
/*       FM1715_OK: 应答正确                                     */
/*       FM1715_AUTHERR: 权威认证有错                            */
/*****************************************************************/
uchar    Authentication(uchar idata *UID,uchar SecNR,uchar mode)
{
    uchar idata i;
    uchar idata temp,temp1;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    ModConductance = 0X3f;
    temp1 = Control;
    temp1 = temp1 & 0xf7;
    Control = temp1;

    if (mode == 1)                                  //AUTHENT1
        buffer[0] = RF_CMD_AUTH_LB;
    else
        buffer[0] = RF_CMD_AUTH_LA;
    buffer[1] = SecNR * 4 + 3;
    for (i = 0; i < 4; i++)
```

```
    {
        buffer[2 + i] = UID[i];
    }
    ChannelRedundancy = 0x0f;                    //开启 CRC,奇偶校验校验
    temp = Command_Send(6, buffer, Authent1);
    if (temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = ErrorFlag;
    if ((temp & 0x02) == 0x02)
        return FM1715_PARITYERR;
    if((temp & 0x04) == 0x04)
        return FM1715_FRAMINGERR;
    if ((temp & 0x08) == 0x08)
        return FM1715_CRCERR;

    temp = Command_Send(0, buffer, Authent2);    //AUTHENT2
    if(temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = ErrorFlag;
    if ((temp & 0x02) == 0x02)
        return FM1715_PARITYERR;
    if((temp & 0x04) == 0x04)
        return FM1715_FRAMINGERR;
    if ((temp & 0x08) == 0x08)
        return FM1715_CRCERR;
    temp1 = Control;
    temp1 = temp1 & 0x08;                        //Crypto1on=1，验证通过
    if (temp1 == 0x08)
    {
        return FM1715_OK;
    }
    return FM1715_AUTHERR;
}
```

5.4.7 READ

```
/*****************************************************************/
/*名称: MIF_Read                                                 */
/*功能: 该函数实现读 MIFARE 卡块的数值                           */
/*输入: buff: 缓冲区首地址                                       */
/*      Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_OK: 应答正确                                      */
/*****************************************************************/
uchar MIF_READ(uchar idata *buff,uchar Block_Adr)
{
```

```
    uchar idata temp;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    ModConductance = 0X3f;
    ChannelRedundancy = 0x0f;
    //Int_Req=0x7f;
    buff[0] = RF_CMD_READ;
    buff[1] = Block_Adr;
    temp = Command_Send(2, buff, Transceive);
    if (temp == 0)
    {
        return FM1715_NOTAGERR;
    }
    temp = ErrorFlag;
    if ((temp & 0x02) == 0x02)  return FM1715_PARITYERR;
    if((temp & 0x04) == 0x04)   return FM1715_FRAMINGERR;
    if ((temp & 0x08) == 0x08)  return FM1715_CRCERR;
    temp = FIFO_Length;
    if (temp == 0x10)                   //8K 卡读数据长度为 16
    {
        Read_FIFO(buff);
        return FM1715_OK;
    }
    else if (temp == 0x04)              //Token 卡读数据长度为 16
    {
        Read_FIFO(buff);
        return FM1715_OK;
    }
    else
    {
        return FM1715_BYTECOUNTERR;
    }
}
```

5.4.8 WRITE

```
/*****************************************************************/
/*名称: MIF_Write                                                */
/*功能: 该函数实现写 MIFARE 卡块的数值                           */
/*输入: buff: 缓冲区首地址                                       */
/*      Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_WRITEERR: 写卡块数据出错                          */
```

```
/*      FM1715_OK: 应答正确                                          */
/*****************************************************************/
uchar MIF_Write(uchar idata *buff,uchar Block_Adr)
{
    uchar idata temp;
    uchar idata *F_buff;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    F_buff = buff + 0x10;
    ChannelRedundancy = 0x07;
    *F_buff = RF_CMD_WRITE;
    *(F_buff + 1) = Block_Adr;
    temp = Command_Send(2, F_buff, Transceive);
    if (temp == FALSE)
    {
        return(FM1715_NOTAGERR);
    }
    temp = FIFO_Length;
    if (temp == 0)
    {
        return(FM1715_BYTECOUNTERR);
    }
    Read_FIFO(F_buff);
    temp = *F_buff;
    switch (temp)
    {
        case 0x00 :
            return(FM1715_NOTAUTHERR);      //暂时屏蔽掉写错误
        case 0x04:
            return(FM1715_EMPTY);
        case 0x0a:
            break;
        case 0x01:
            return(FM1715_CRCERR);
        case 0x05:
            return(FM1715_PARITYERR);
        default:
            return(FM1715_WRITEERR);
    }
        temp = Command_Send(16, buff, Transceive);
    if (temp == TRUE)
    {
        return(FM1715_OK);
    }
    else
    {
        temp = ErrorFlag;
        if ((temp & 0x02)==0x02)
            return(FM1715_PARITYERR);
```

```
        else if((temp & 0x04)==0x04)
            return(FM1715_FRAMINGERR);
        else if ((temp & 0x08)==0x08)
            return(FM1715_CRCERR);
        else
            return(FM1715_WRITEERR);
    }
}
```

5.4.9 INCREMENT

```
/*****************************************************************/
/*名称: MIF_Increment                                            */
/*功能: 该函数实现 MIFARE 卡自动增值操作                         */
/*输入: buff: 四个字节数值起始地址                               */
/*      Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_INCRERR: 卡片增款操作失败                         */
/*      FM1715_OK: 应答正确                                      */
/*****************************************************************/
uchar MIF_Increment(uchar idata *buff,uchar Block_Adr)
{
    uchar temp;
    uchar idata *F_buff;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    F_buff = buff + 4;
    *F_buff = RF_CMD_INC;
    *(F_buff + 1) = Block_Adr;
    ChannelRedundancy = 0x07;
    temp = Command_Send(2, F_buff, Transceive);
    if (temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = FIFO_Length;
    if (temp == 0)
    {
        return FM1715_BYTECOUNTERR;
    }
    Read_FIFO(F_buff);
    temp = *F_buff;
    switch(temp)
```

```
    {
        case 0x00 :
            //break;
            return(FM1715_NOTAUTHERR);      //暂时屏蔽掉写错误
        case 0x04:
            return(FM1715_EMPTY);
        case 0x0a:
            break;
        case 0x01:
            return(FM1715_CRCERR);
        case 0x05:
            return(FM1715_PARITYERR);
        default:
            return(FM1715_INCRERR);
    }

    temp = Command_Send(4, buff, Transmit);
    if (temp == FALSE)
    {
        return FM1715_INCRERR;
    }
    return FM1715_OK;
}
```

5.4.10 DECREMENT

```
/*****************************************************************/
/*名称: MIF_Decrement                                            */
/*功能: 该函数实现 MIFARE 卡自动减值操作                         */
/*输入: buff: 四个字节数值起始地址                               */
/*      Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_DECRERR: 卡片扣款操作失败                         */
/*      FM1715_OK: 应答正确                                      */
/*****************************************************************/
uchar MIF_Decrement(uchar idata *buff,uchar Block_Adr)
{
    uchar temp;
    uchar idata *F_buff;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    F_buff = buff + 4;
    *F_buff = RF_CMD_DEC;
```

```
    *(F_buff + 1) = Block_Adr;
    ChannelRedundancy = 0x07;
    temp = Command_Send(2, F_buff, Transceive);
    if (temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = FIFO_Length;
    if (temp == 0)
    {
        return FM1715_BYTECOUNTERR;
    }
    Read_FIFO(F_buff);
    temp = *F_buff;
    switch (temp)
    {
        case 0x00 :
            //break;
            return(FM1715_NOTAUTHERR);      //暂时屏蔽掉写错误
        case 0x04:
            return(FM1715_EMPTY);
        case 0x0a:
            break;
        case 0x01:
            return(FM1715_CRCERR);
        case 0x05:
            return(FM1715_PARITYERR);
        default:
            return(FM1715_DECRERR);
    }

    temp = Command_Send(4, buff, Transmit);
    if (temp == FALSE)
    {
        return(FM1715_DECRERR);
    }
    return FM1715_OK;
}
```

5.4.11 RESTORE

```
/*****************************************************************/
/*名称: MIF_Restore                                              */
/*功能: 该函数实现 MIFARE 卡自动恢复,备份操作                    */
/*输入: Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
```

```
/*      FM1715_CRCERR: CRC 校验错                                       */
/*      FM1715_PARITYERR: 奇偶校验错                                    */
/*      FM1715_RESTERR: 卡片恢复,备份操作失败                           */
/*      FM1715_OK: 应答正确                                             */
/*******************************************************************/
uchar MIF_Restore(uchar Block_Adr)
{
    uchar temp,i;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    ChannelRedundancy = 0x07;
    *buffer = RF_CMD_RESTORE;
    *(buffer + 1) = Block_Adr;
    temp = Command_Send(2, buffer, Transceive);
    if (temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = FIFO_Length;
    if (temp == 0)
    {
        return FM1715_BYTECOUNTERR;
    }
    Read_FIFO(buffer);
    temp = *buffer;
    switch (temp)
    {
        case 0x00 :
            //break;
            return(FM1715_NOTAUTHERR);      //暂时屏蔽掉写错误
        case 0x04:
            return(FM1715_EMPTY);
        case 0x0a:
            break;
        case 0x01:
            return(FM1715_CRCERR);
        case 0x05:
            return(FM1715_PARITYERR);
        default:
            return(FM1715_RESTERR);
    }

    for (i = 0; i < 4; i++)
        buffer[i] = 0x00;
    temp = Command_Send(4, buffer, Transmit);
    if (temp == FALSE)
    {
        return FM1715_RESTERR;
    }
```

```
    return FM1715_OK;
}
```

5.4.12 TRANSFER

```
/*****************************************************************/
/*名称: MIF_Transfer                                             */
/*功能: 该函数实现 MIFARE 卡电子钱包保存操作                     */
/*输入: Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_TRANSERR: 卡片恢复,备份操作失败                   */
/*      FM1715_OK: 应答正确                                      */
/*****************************************************************/
uchar MIF_Transfer(uchar Block_Adr)
{
    uchar temp;

    CRCPresetLSB = 0x63;
    CWConductance = 0x3f;
    ChannelRedundancy = 0x07;
    buffer[0] = RF_CMD_TRANSFER;
    buffer[1] = Block_Adr;
    temp = Command_Send(2, buffer, Transceive);
    if (temp == FALSE)
    {
        return FM1715_NOTAGERR;
    }
    temp = FIFO_Length;
    if (temp == 0)
    {
        return FM1715_BYTECOUNTERR;
    }
    Read_FIFO(buffer);
    temp = *buffer;
    switch (temp)
    {
        case 0x00 :
            //break;
            return(FM1715_NOTAUTHERR);      //暂时屏蔽掉写错误
        case 0x04:
            return(FM1715_EMPTY);
        case 0x0a:
            return(FM1715_OK);
        case 0x01:
```

```
            return(FM1715_CRCERR);
        case 0x05:
            return(FM1715_PARITYERR);
        default:
            return(FM1715_TRANSERR);
    }
}
```

六、例程

6.1 卡片触发

```
/*****************************************************************/
/*名称: HL_Active                                                */
/*功能: 该函数实现高级 MIFARE 卡激活命令                         */
/*输入: Secnr: 扇区号                                            */
/*      Block_Adr: 块地址                                        */
/*输出: 操作状态码                                               */
/*      读出数据存于 buffer 中                                   */
/*****************************************************************/
uchar HL_Active(uchar Block_Adr,uchar Mode)
{
    uchar temp;

    Secnr = Block_Adr/4;
    MIF_Halt();                             //Halt
    temp = Request(RF_CMD_REQUEST_STD);     //Request
    if(temp != FM1715_OK)
    {
        return(FM1715_REQERR);
    }
    temp = AntiColl();                      //AntiCol
    if(temp != FM1715_OK)
    {
        return(FM1715_ANTICOLLERR);
    }
    temp = Select_Card();                   //Select
    if(temp != FM1715_OK)
    {
        return(FM1715_SELERR);
    }
    Load_keyE2_CPY((Secnr%16), Mode);           //LoadKey
    temp = Authentication(UID, Secnr, Mode);//Authentication
    if(temp != FM1715_OK)
    {
        return(FM1715_AUTHERR);
    }
    return FM1715_OK;
}
```

6.2 INITVAL VALUE

```
/*****************************************************************/
/*名称: MIF_Initival                                             */
/*功能: 该函数实现 MIFARE 卡初始化值操作                         */
/*输入: buff: 四个字节初始化数值起始地址                         */
/*      Block_Adr: 块地址                                        */
/*输出: FM1715_NOTAGERR: 无卡                                    */
/*      FM1715_BYTECOUNTERR: 接收字节错误                        */
/*      FM1715_NOTAUTHERR: 未经权威认证                          */
/*      FM1715_EMPTY: 数据溢出错误                               */
/*      FM1715_CRCERR: CRC 校验错                                */
/*      FM1715_PARITYERR: 奇偶校验错                             */
/*      FM1715_WRITEERR: 写卡块数据出错                          */
/*      FM1715_OK: 应答正确                                      */
/*****************************************************************/
uchar MIF_Initival(uchar idata *buff,uchar Block_Adr)
{
    uchar idata temp;
    uchar i;

    for (i = 0; i < 4; i++)
    {
        *(buff + 4 + i)=~(*(buff + i));
    }
    for (i = 0; i < 4; i++)
    {
        *(buff + 8 + i)=*(buff + i);
    }
    *(buff + 12) = Block_Adr;
    *(buff + 13) = ~Block_Adr;
    *(buff + 14) = Block_Adr;
    *(buff + 15) = ~Block_Adr;
    temp = MIF_Write(buff, Block_Adr);
    return temp;
}
```

6.3 读卡

```
/*****************************************************************/
/*名称: HL_Read                                                  */
/*功能: 该函数实现高级读命令                                     */
/*输入: Secnr: 扇区号                                            */
/*      Block_Adr: 块地址                                        */
/*输出: 操作状态码                                               */
/*      读出数据存于 buffer 中                                   */
/*****************************************************************/
uchar HL_Read(uchar idata *buff,uchar Block_Adr,uchar Mode)
{
    uchar temp;

    temp = HL_Active(Block_Adr, Mode);
    if(temp != FM1715_OK)
    {
        return temp;
    }

    //Read
    temp = MIF_READ(buff,Block_Adr);
    if(temp != FM1715_OK)
    {
        return temp;
    }
    return FM1715_OK;
}
```

6.4 写卡

```
/*****************************************************************/
/*名称: HL_Write                                                 */
/*功能: 该函数实现高级写命令                                     */
/*输入: buff: 待写入数据的首地址                                 */
/*      Secnr: 扇区号                                            */
/*      Block_Adr: 块地址                                        */
/*输出:操作状态码                                                */
/*****************************************************************/
uchar HL_Write(uchar idata *buff,uchar Block_Adr,uchar Mode)
{
    uchar temp;

    temp = HL_Active(Block_Adr, Mode);
    if(temp != FM1715_OK)
    {
        return temp;
```

```
    }
    //Write
    temp = MIF_Write(buff, Block_Adr);
    if(temp != FM1715_OK)
    {
        return FM1715_WRITEERR;
    }
    return FM1715_OK;
}
```