Advanced Card Systems Ltd.
Card & Reader Technologies

ACR1283L

脱机式
非接触读写器

应用程序编程接口 V1.00

info@acs.com.hk
www.acs.com.hk

目录

图目录

表目录

1.0. 简介

ACR1283L 脱机式非接触读写器是一款用于读写非接触式智能卡的设备。它的非接触式接口可用于读写符合 ISO 14443 标准的 A 类和 B 类卡以及 Mifare 系列卡片。另外它还带有安全存取模块（SAM）接口以确保非接触式智能卡应用的高度安全性。

ACR1283L 可以支持连机和脱机两种工作模式。在脱机模式下，ACR1283L 无需主机端 PC 发送命令即可对非接触式智能卡进行操作。本手册详细介绍了 ACR1283L 如何在脱机模式下进行非接触式应用。

2.0. 特性

- 双工作模式：
  - 连机
  - 脱机
- 连机操作：
  - USB 2.0 全速接口
  - 符合 CCID 标准
  - 支持 PC/SC
  - 支持 CT-API（通过 PC/SC 上一层的封装）
- 脱机操作：
  - 支持第三方应用程序编程
  - 超过 400 KB 的第三方应用程序存储空间
  - 超过 500 KB 的数据存储空间
  - 支持的开发平台：
    - IAR 嵌入式工作台（5.50 或以上版本）
    - CoIDE(GCC)（1.3.0 或以上版本）
- 智能卡读写器：
  - 读写速率高达 848 kbps
  - 内置天线用于读写非接触式标签，读取智能卡的距离可达 50 mm（视标签的类型而定）
  - 支持 ISO 14443 第 4 部分的 A 类和 B 类卡，以及 Mifare 系列卡
  - 内建防冲突特性（任何时候都只能访问 1 张标签）
  - 4 个符合 ISO 7816 标准的 SAM 卡卡槽
- 内置外围设备：
  - 两行图形液晶显示屏
  - 4 个用户可控的 LED 指示灯
  - 1 个用户可控的蜂鸣器
  - 12 键电容式触摸键盘
- 带独立备用电池的实时时钟（RTC）
- 设备内 AES（128 或 256）、DES 和 3DES 加密算法
- 支持 Android™ OS 3.1 及以上版本
- 具有 USB 固件升级能力

- 符合下列标准：
  - ISO 14443
  - CE
  - FCC
  - PC/SC
  - CCID
  - Microsoft® WHQL
  - RoHS
  - USB 2.0 全速

3.0. 符号和缩写

缩写	含义
APDU	应用协议数据单元 Application Protocol Data Unit
APDU 命令	APDU 的四个起始字节（例如：CLA INS P1 P2）Four-byte sequence that begins with APDU (e.g., CLA INS P1 P2)
命令头	(ISO/IEC 7816-4§5.3.1)
ATR	复位应答 Answer To Reset
CCID	符合 CCID 规范的集成电路卡接口设备 Integrated Circuit(s) Cards Interface Device conforming to this specification
API	应用程序编程接口 Application Programming Interface
DES	数据加密标准（FIPS Pub 46） Date Encryption Standard (FIPS Pub 46)
3DES	3 倍 DEA 算法（对每个数据块应用三次 DEA 加密算法）Triple DES (which applies the DES cipher algorithm three times to each data block)
AES	高级加密标准 Advanced Encryption Standard
PnP	即插即用 Plug-and-Play
FRAM	非易失性铁电存储器 Ferroelectric non-volatile RAM

表 1：符号和缩写

4.0. USB 通信协议

通信数据的格式包符合 CCID_Rev110 标准。更多信息请参考以下的注释及示例描述。

注：*不支持扩展的 APDU。*

4.1. 数据包格式

数据包由 10 个字节的报头及其后面的用户数据组成。10 个字节的报头包含如下信息：

偏移量	数据域	大小
0	*bMessageType*	1
1	*dwLength*	4
5	*bSlot*	1
6	*bSeq*	1
7	*bBWI*	1
8	*wLevelParameter*	2
10	*adData*	字节型数组

bSlot 0 = PICC 卡槽

1 = SAM1 卡槽

2 = SAM2 卡槽

3 = SAM3 卡槽

4 = SAM4 卡槽

注：

1. *数据长度不包含 10 字节的报头。*
2. *先传输 LSB，后传输 MSB。*

4.2. 示例代码

命令：6F 05 00 00 00 011100 00 00 80 84 00 00 08

说明：

6F	05 00 00 00	01	11	00 00 00	80 84 00 00 08

信息类型： 6Fh

注：*关于 PC_to_RDR_XfrBlock，请参阅 CCID_Rev110。*

数据长度： 00000005 >> 5 个字节的用户数据

卡槽： 01h >> 发送数据到 SAM1

序列号： 11h

特殊用法： 00 00 00h

用户数据： 80 84 00 00 08h

5.0. 第三方编程体系结构

5.1. 体系结构

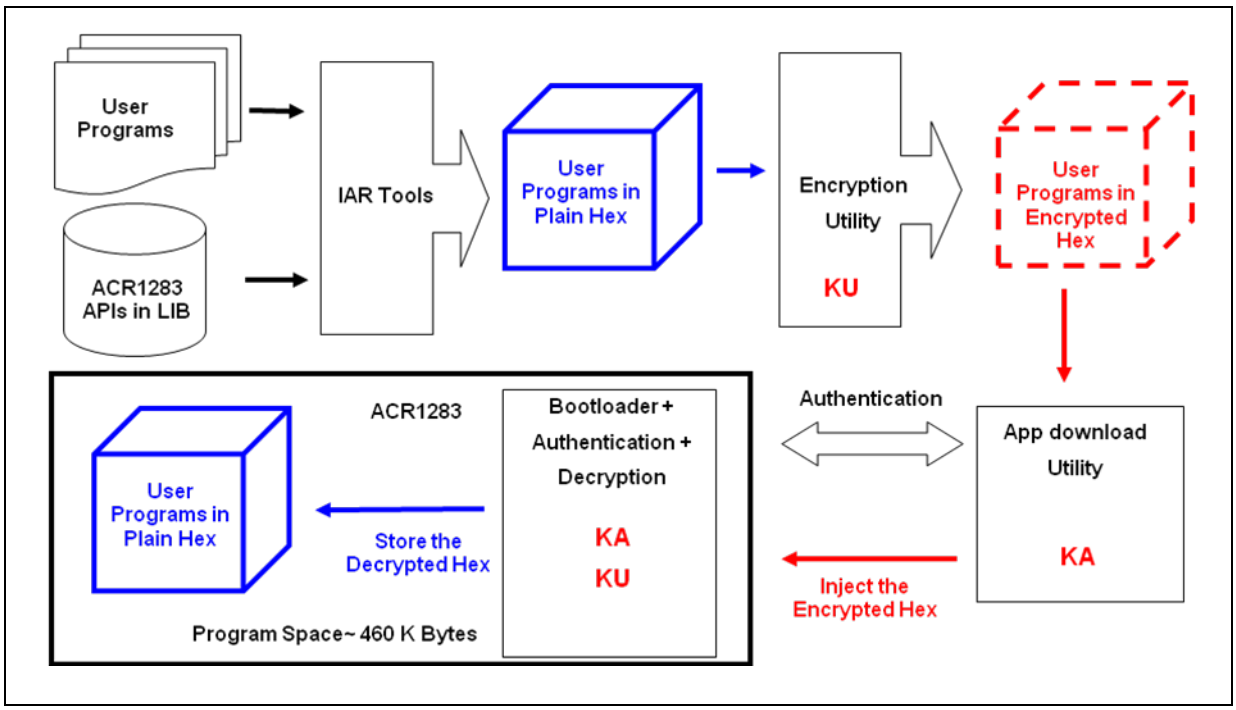

图 1：第三方编程体系结构

User Programs	由第三方编写的程序，执行用户自定义的应用。
ACR1283 APIs in LIB	内部程序，用于控制所有的外设（非开源），并为用户编程提供 API。
IAR Tools	编译工具。用于编译 APP，支持 IAR 和 GCC 开发环境。
Encrypted Hex	通过 AES 加密来保护知识产权。
App Download Utility	嵌入了认证过程和下载过程。

5.2. 程序入口

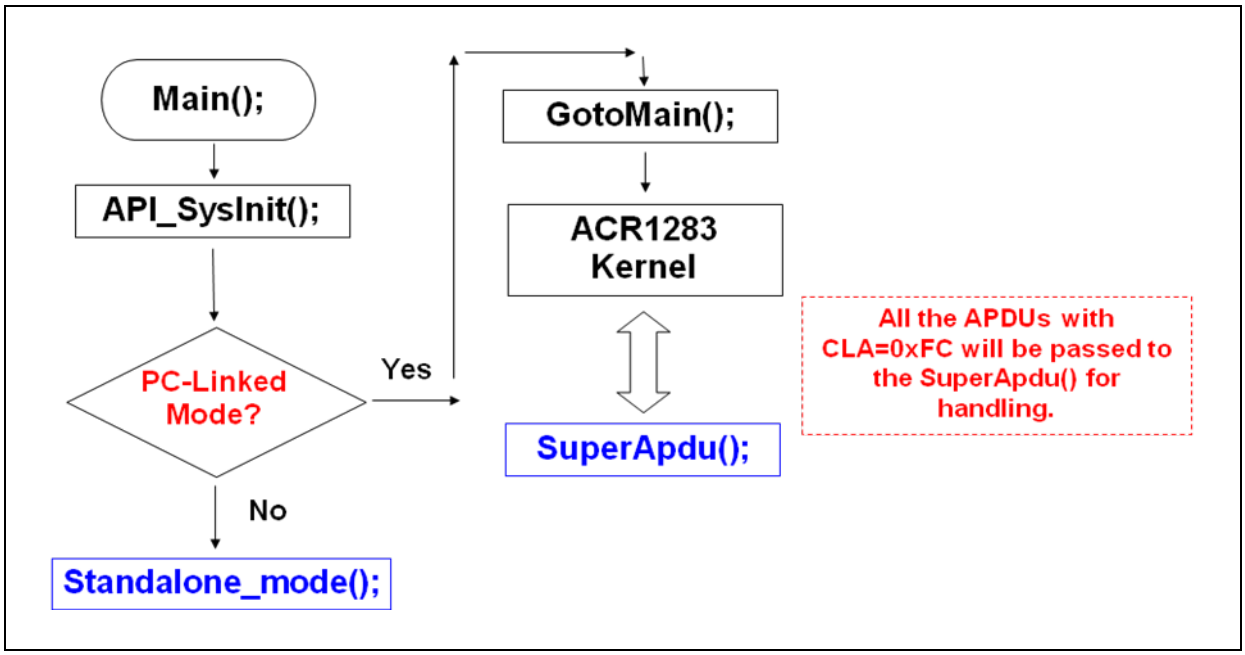

图 2：程序入口流程图

6.0. 应用程序编程接口

本节介绍了不同的 API 函数及其所需的参数和返回值，同时提供了一些简单的源代码来进行说明。用户需要根据具体要求对其进行修改。

6.1. 数据类型

- **INT8** 带符号 8 位整数
- **UINT8** 无符号 8 位整数
- **INT16** 带符号 16 位整数
- **UINT16** 无符号 16 位整数
- **INT32** 带符号 32 位整数
- **UINT32** 无符号 32 位整数

6.2. 卡片操作的 API 函数

本节列出了与卡片操作相关的 API 函数。对于这些命令，我们必须通过一个参数来设置要使用的卡片接口。相关的设置如下表。

ucSlot	卡槽
01h	PICC
02h	SAM1
03h	SAM2
04h	SAM3
05h	SAM4

6.2.1. 数据交换（Data Exchange）

这个函数用于存取 APDU 和私有 APDU，然后接收返回的报文。

```
UINT8 API_ExAPDU(UINT8 ucSlot,  UINT8 *pucApdu,  UINT16 uiApduLen,
UINT8 *pucRsp,  UINT16 *puiRspLen, UINT8 *pucSw)
```

参数

ucSlot 卡槽编号

pucApdu 存放数据的缓冲区地址；发送至

uiApduLen 数据的长度（16 位）

pucRsp 存放返回消息的缓冲区地址

puiRspLen 存放返回消息长度的缓冲区地址（16 位）

pucSw 存放返回状态（SW0 SW1）的缓冲区地址

注：*返回状态的长度为 2 个字节。*

返回参数

返回值表示命令执行的状态。有关状态码的说明，请参阅**附录 A**。

示例代码

```
void GetRandomNbr( void )
{
UINT8  status, slot, response_array[8], sw[2];
UINT16 senddatalen, recdatalen;
static UINT8 apdu_array[] = { 0x80, 0x84, 0x00, 0x00, 0x08 };

slot = 0x01;
senddatalen = 0x05;

status              =              API_ExAPDU(slot,apdu_array,senddatalen,
response_array,&recdatalen, sw);
/*status = 00, mean success, if failed, please refer to appendix 1 for
the status code
Content of response_array:8 bytes random number;
recdatalen = 0x0A; 8 bytes random number + SW1 SW2
*/
……
 }
```

含义

向 SAM1 发送 *Get Challenge* 命令以获取 8 个字节的随机数。

APDU = “80 84 00 00 08h”

响应 = 8 字节随机数

SW1 SW2 = “90 00h”

6.2.2. 激活卡片（Activating a Card）

这个函数用于激活卡片并接收 ATR：

```
UINT8 API CardPowerOn(UINT8 ucSlot, UINT8 *pcuATR, UINT8 *pucAtrLen)
```

参数

ucSlot	卡槽编号
pcuATR	存放卡片返回的 ATR 的缓冲区地址
pucAtrLen	存放 *pcuATR* 长度的缓冲区地址

返回参数

返回值表示卡片激活的结果。有关状态码的说明，请参阅**附录 A**。

示例代码

```
void PowerOn( void )
 {
UINT8  status,  artlen, atr_array[64];
```

```
status = API_CardPowerOn ( 0x02, atr_array , &artlen ),

/* status = 00, mean success, if failed, please refer to appendix 1 for
the status code contend of atr_array, for example:0x3b 0x8f 0x80 0x01
0x80 0x4f 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00
0x00 0x6a
artlen = 0x14 */
……
 }
```

含义

激活 SAM2

ATR 示例：“3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah”

6.2.3. 取消激活卡片（Deactivating a Card）

此函数用在完成对卡片的操作后，取消激活卡片。

```
UINT8 API_CardPowerOff(UINT8 ucSlot)
```

参数

ucSlot 卡槽编号

返回参数

成功 API_OK

失败 API_ERR_INDEX/API_ERR_CMDFAIL

6.2.4. 重新激活卡片（Reactivating a Card）

此函数用于激活卡片并返回卡槽状态：

```
UINT8 API_InterfaceRefresh(UINT8 ucSlot, UINT8 *pucStatus);
```

注：此函数不会返回 ATR。

参数

ucSlot 卡槽编号

pucStatus 卡槽的当前状态

pucStatus 位定义：

位	卡槽
Bit-0	PICC
Bit-1	SAM1
Bit-2	SAM2
Bit-3	SAM3
Bit-4	SAM4

注：*全部为 0 表示没有卡片，或者未能激活卡槽。*

返回参数

返回值表示卡片激活的结果。有关状态码的含义，请参阅**附录 A**。

6.2.5. 设置 PICC 通信速率（Setting PICC Communication Speed）

此函数用于设置 PICC 通信速率。若用户设置的速率高于卡片所支持的最大速率，则会使用卡片支持的最大速率进行通信。

```
Void API_SetPPSPicc(UINT8 ucPar);
```

参数

ucPar 用户设置的通信速率

- 00h = 106k bps
- 01h = 212k bps
- 02h = 424k bps（默认设置）
- 03h = 848k bps

返回参数

无。

6.2.6. 设置 SAM1~SAM4 通信速率（Setting SAM1~SAM4 Communication Speed）

FiDi 设置 ICC 通信速率：

```
UINT8 API_SetPPSIcc(UINT8 ucSlot,UINT8 pucFiDi,UINT8 ucPar);
```

Fi **and** ***f*** **(max.)**

Bits 8 to 5	0000	0001	0010	0011	0100	0101	0110	0111
Fi	372	372	558	744	1116	1488	1860	RFU
f (max.) MHz	4	5	6	8	12	16	20	—

Bits 8 to 5	1000	1001	1010	1011	1100	1101	1110	1111
Fi	RFU	512	768	1024	1536	2048	RFU	RFU
f (max.) MHz	—	5	7.5	10	15	20	—	—

Di

Bits 4 to 1	0000	0001	0010	0011	0100	0101	0110	0111
Di	RFU	1	2	4	8	16	32	64

Bits 4 to 1	1000	1001	1010	1011	1100	1101	1110	1111
Di	12	20	RFU	RFU	RFU	RFU	RFU	RFU

参数

ucSlot 卡槽编号

pucFiDi 卡片和读写器之间的通信速率

- bit7 至 bit4：Fi
- bit3 至 bit0：Di

ucPar 速率类型

- 0 表示数据通信速率
- 1 表示获取 ATR 的速率

应当在激活前设置好 ATR，方法是：

```
UINT8 API_CardPowerOn(UINT8 ucSlot,UINT8 *pcuATR,UINT8 *pucAtrLen)
```

如果用户想要设置 PPS，应首先使用 API，`UINT8 API_AutoPPSSet(UINT8 ucSlot,UINT8 ucPar)`，并设置 PPS 为必须在激活前完成。

返回参数

成功 API_OK

失败 API_ERR_INDEX/API_ERR_CMDFAIL

6.2.7. 设置自动 PPS（Setting Auto PPS）

此函数用于启用/禁用自动 PPS。

```
UINT8 API_AutoPPSSet(UINT8 ucSlot,UINT8 ucPar);
```

参数

ucSlot 卡槽编号

ucPar 启用/禁用参数

- 0 = 禁用自动 PPS
- 1 = 启用自动 PPS

返回值

成功 00h

失败 01h

6.2.8. 获取 ATR （Getting ATR）

此函数用于返回 ATR，但不会激活卡槽：

```
UINT8 API_GetATR(UINT8 ucSlot, UINT8 *pucATR, UINT8 *pucATRLen)
```

参数

ucSlot 卡槽编号

pucATR 存放当前卡片 ATR 的缓冲区地址

pucATRLen 存储 ATR 长度的缓冲区地址

注：*如果没有发现卡片或者出现其它错误，返回 **00h**。*

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

示例代码

```
void Get_ATR( void )
 {
UINT8  status,  artlen, atr_array[64];
status = GetATR ( 0x02, atr_array , &artlen ),
/* status = 00, mean success, if failed, please refer to appendix 1 for
the status code contend of atr_array, for example:0x3b 0x8f 0x80 0x01
0x80 0x4f 0x0c 0xa0 0x00 0x00 0x03 0x06 0x03 0x00 0x01 0x00 0x00 0x00
0x00 0x6a
artlen = 0x14 */
……
}
```

含义

激活 SAM2

ATR 示例：“3B 8F 80 01 80 4F 0C A0 00 00 03 06 03 00 01 00 00 00 00 6Ah”

6.2.9. 获取卡槽状态（Getting Card Slot Status）

此函数用于获取卡槽的状态：

```
UINT8 API_GetSlotStatus(UINT8 ucSlot, UINT8 *pucStatus);
```

参数

ucSlot 卡槽编号

pucStatus 卡槽状态

位	卡槽
Bit-0	PICC
Bit-1	SAM1
Bit-2	SAM2
Bit-3	SAM3
Bit-4	SAM4

注：*全部为 **0** 表示没有卡片，或者未能读取卡槽。*

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.10. 逃逸函数（Escape Function）

此函数用于处理自定义的消息（例如：读写器设置、自定义命令等）。

```
UINT8 API_EscapeCmd(UINT8 ucSlot, UINT8 *pucCmd, UINT16 uiCmdLen, UINT8
*pucRsp, UINT16 *puiRspLen);
```

参数

ucSlot 卡槽编号

- 08 = 读写器命令

pucCmd 数据地址

uiCmdLen 数据长度

pucRsp 返回数据地址

puiRspLen 返回数据地址长度

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.11. 获取 UID（Getting UID）

此函数用于获取卡片的 UID：

```
void API_GetUID(UINT8 * pucUIDBuf, UINT8 * pucUIDLength);
```

参数

pucUIDBuf 存放卡片 UID 的缓冲区

pucUIDLength 卡片 UID 的长度

返回值

无。

6.2.12. 获取易失性密钥（Getting Volatile Key）

此函数用于获取卡片的易失性密钥。

```
void API_GetMFVolatileKey(UINT8 * pucKeyBuf);
```

参数

pucKeyBuf 存放 6 字节易失性密钥的缓冲区

返回值

无。

6.2.13. 存储易失性密钥（Saving Volatile Key）

此函数用于保存卡片的易失性密钥：

```
void API_StoreMFVolatileKey(UINT8 * pucKeyBuf);
```

参数

pucKeyBuf 存放 6 字节易失性密钥的缓冲区

返回值

无。

6.2.14. 轮询卡片（Polling a Card）

此函数用于轮询卡片：

```
UINT8 API_PollPICC(UINT8 ucTagType, UINT8 ucPollRetry, UINT8
ucPollInterval, UINT8 * pucUIDBuf, UINT8 * pucUIDLength);
```

参数

ucTagType 要轮询的卡片类型：

- 0 = TypeA
- 1 = TypeB

ucPollRetry 轮询卡片的次数

- 1 = 轮询一次
- 2 = 轮询两次等等（如果 *ucPollRetry* = 0，则会轮询 256 次）

ucPollInterval 轮询卡片间隔，单位为 10ms，0 表示一直轮询卡片）

pucUIDBuf 存放卡片 UID 的缓冲区

pucUIDLength 卡片 UID 的长度

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.15. 设置 Mifare 卡认证（Setting Mifare Authentication）

此函数用于设置对 Mifare 卡的认证：

```
void API_MFAuthNeed(UINT8 ucCmd);
```

参数

ucCmd 0 = 无需对 Mifare 卡进行认证

1 = 要求对 Mifare 卡进行认证

返回值

无。

6.2.16. 设置卡片轮询参数（Setting Parameter for Polling a Card）

此函数用于设置卡片轮询的参数：

```
void API_PICCPollingSet(UINT8 ucAutoPolling, UINT8 ucCardTypes, UINT8
ucPart4Enter, UINT8 ucPollingInterval);
```

参数

ucAutoPolling 0 = 关闭自动卡片轮询

1 = 开启自动卡片轮询

ucCardTypes 要轮询的卡片的类型

- 01h = Type A
- 02h = Type B
- 03h = Type A + Type B

ucPart4Enter 进入 OSP/IEC 14443 第 4 部分，

- 0 = 不进入第 4 部分
- 1 = 进入第 4 部分

ucPollingInterval 卡片轮询的时间间隔，单位为 125 ms

- 0 = 持续地轮询卡片

返回值

无。

6.2.17. 终止 Mifare 卡（Halting a Mifare Card）

此函数用于使选中的 Mifare 卡进入中止（HALT）状态：

```
UINT8 API_HaltMF(void);
```

参数

无参数。

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.18. 唤醒 Mifare 卡（Waking up a Mifare Card）

此函数用于唤醒处于 HALT 状态的 Mifare 卡：

```
UINT8 API_WupMF(void)
```

参数

无参数。

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.19. 检查 SAM 文件状态（Checking SAM File Status）

此函数用于检查 SAM 卡密钥文件是否开放或关闭（用于测试）：

```
UINT8 API_GetSAMFileStatus(UINT8 ucSlot);
```

参数

ucSlot 卡槽编号

返回值

返回状态。有关状态码的说明，请参阅**附录 A**。

6.2.20. 设置 SAM 文件状态（Setting SAM File Status）

此函数用于设置 SAM 卡密钥文件的开启或关闭状态：

```
UINT8 API_SAMFileStatusConfigure(UINT8 ucSlot, UINT8 ucCmd);
```

参数

ucSlot 卡槽编号

ucCmd 0 = 密钥文件关闭

1 = 密钥文件开启

返回值

成功 API_OK

失败 API_ERR_INDEX

6.3. 读写器的 API 函数

本节介绍了用于设置读写器和获取读写器状态的 API 函数。

6.3.1. 查看固件版本（Checking the Firmware Version）

此函数用于读取固件的版本号：

```
void API_GetFWVersion(UINT8 *pucBuffer, UINT8 *pucLen);
```

参数

pucBuffer 存放固件版本的缓冲区

固件版本的长度为 17 个字节，格式如下：

ACR1283U Vx.x.x.x；其中 x 表示版本号。

注：字节的每个字符均使用 ASCII 码（例如：Y = 59, S = 53, T = 54）

pucLen 固件版本号的长度

返回值

无。

6.3.2. 复位读写器（Resetting the Reader）

此函数用于复位读写器：

```
void API_SystemReset(void);
```

参数

无参数。

返回值

无。

6.3.3. 进入固件升级模式（Entering Firmware Upgrade Mode）

此命令用于使读写器进入固件升级模式。

```
UINT8 API_EnterDfuMode(void);
```

参数

无参数。

返回值

成功 无返回

失败 返回 API_ERR_CMDFAIL

6.3.4. 控制蜂鸣器（Controlling the Buzzer）

此函数用于设置蜂鸣器的运行模式：

```
void API_BuzzerCtrl(UINT8 ucOnTime,UINT8 ucOffTime,UINT8 ucCnt,UINT8 ucMode);
```

参数

ucOnTime 蜂鸣器鸣响时间

ucOnTime 设置范围为 01h 至 FFh；单位为 10 ms

（例如：如果 *ucOnTime* 等于 64h，则蜂鸣器鸣响时间为 100 x 10 ms = 1000 ms）

ucOffTime 蜂鸣器关闭时间

- *ucOffTime* 的设置范围为 01h 至 FFh；单位为 10 ms。

ucCnt 重复时间

- 00h = 蜂鸣器关闭
- 01h~FEh = 蜂鸣器重复响音次数
- FFh = 蜂鸣器持续鸣响

ucMode 模式设置

- 0 = 阻塞模式（蜂鸣器响音结束后，软件才会运行）
- 1 = 非阻塞模式

返回值

无。

示例代码

```
void BuzzerTest(void)
{
//100ms on, 100ms off, repeat 10 times in blocking mode
API_BuzzerCtrl(10,10,10,0);
// 100ms on, 200ms off, repeat 5 times in blocking mode
API_BuzzerCtrl(10,20,5,0);
// 100ms on, 500ms off, repeat 5 times in unblocking mode
API_BuzzerCtrl(10,50,5,1);
}
```

6.3.5. 设置 LED 状态（Setting LED Status）

此函数用于设置 LED 1 - 4 的状态：

```
void API_LedCtrl(UINT8 ucStatus,UINT8 ucNbr);
```

参数

ucStatus 开启/关闭 LED 1 - 4

ucNbr 0 = 全部 D1 - D4

1 = D1

2 = D2

3 = D3

4 = D4

如果 *ucNbr* 的值为 1-4，则将 *ucStatus* 的最低有效位置为 1 会开启相应的 LED，而置为 0 会关闭该 LED。

如果 *ucNbr* 的值为 0，则可以同时对四个 LED 进行控制。*ucStatus* 的四个最低有效位 b0、b1、b2 和 b3 分别控制 D1、D2、D3 和 D4。如果该位置为 1/0，则相应的 LED 会开启/关闭。

返回值

无。

示例代码

```
void DelayLedTest(void)
{
API_LedCtrl(0x00,0); // all off

API_LedCtrl(0x01,1); // on D1
API_LedCtrl(0x01,2); // on D2
 API_LedCtrl(0x01,3); // on D3
 API_LedCtrl(0x01,4); // on D4

 API_LedCtrl(0x0f,0); // all on

 API_LedCtrl(0x00,1); // off D1
 API_LedCtrl(0x00,2); // off D2
 API_LedCtrl(0x00,3); // off D3
 API_LedCtrl(0x00,4); // off D4
}
```

6.3.6. 读取 LED 当前状态（Reading LED Current Status）

此函数用于读取 LED 的当前状态：

```
UINT8 API_LedStatus(void);
```

参数

无参数。

返回值

返回当前的 LED 状态。

- bit0 <-------> LED1 (PCB 标记： D1)
- bit1 <-------> LED2 (PCB 标记： D2)
- bit2 <-------> LED3 (PCB 标记： D3)
- bit3 <-------> LED4 (PCB 标记： D4)
- “1” = LED 开启
- “0” = LED 关闭

6.4. LCD 的 API 函数

这里介绍了与 LCD 相关的 API。

6.4.1. 设置 LCD 背光（Setting LCD Backlight）

此函数用于设置 LCD 的背光状态：

```
void APT_LcdBackLightCtrl(UINT8 ucCtrlCode);
```

参数

ucCtrlCode 0 = 关闭背光

1 = 开启背光

返回值

无。

6.4.2. LCD 显示字符（Displaying Characters in LCD）

此函数用于根据字符表在 LCD 上显示字符：

```
UINT8 API_LcdDisplay(UINT8 Table, UINT8 *pLCDBuf, UINT8 r len, unsigned
char X, UINT8 Y)
```

参数

Table Table = 0X 表示 ASCII - 表 1

Table = 1X 表示 ACSII - 表 2

Table = 2X 表示 ACSII - 表 3

Table = 4X 表示 GB

Table = 0xX1 加粗字符

Table = 0xX0 常规字符

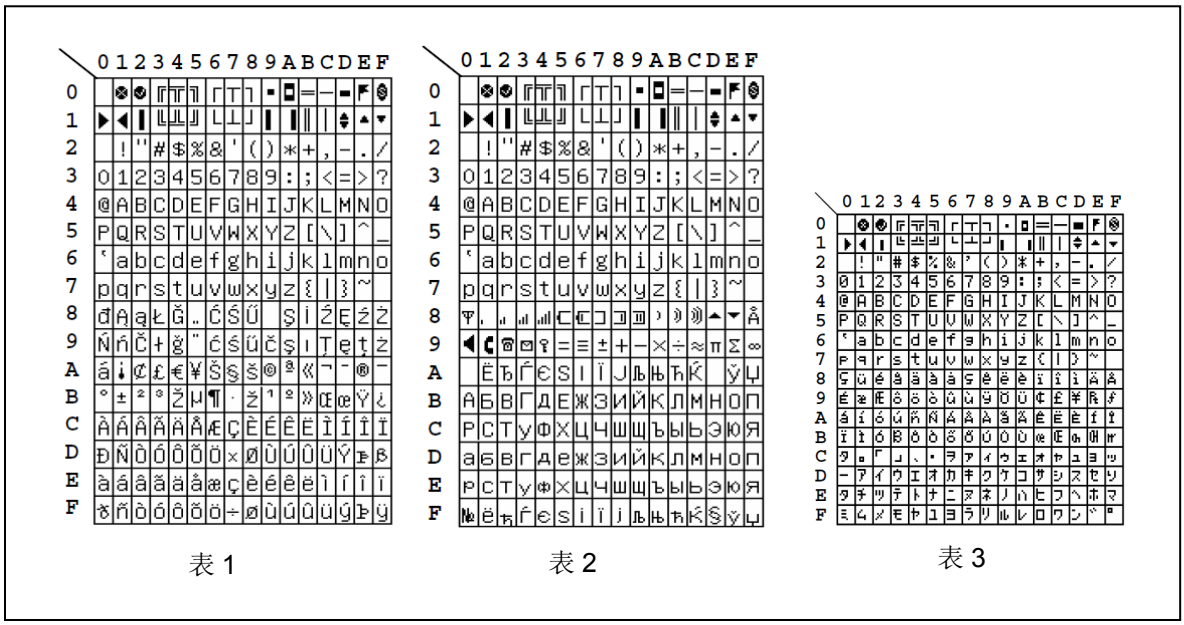

图 3：字符显示表

pLCDBuf 显示消息的地址

len 显示消息的长度（最长 16 个字节）

- X: x 坐标（0 - 15）
- Y: y 坐标（0 - 1）

返回值

成功 API_OK

失败　API_ERR_CMDFAIL

示例代码

```
API_LcdDispay(0x10,"ACR1283",7,5,0); //display the message ”ACR1283”
```

6.4.3. LCD 显示图形（Displaying Graphs in LCD）

此函数用于在 LCD 上显示图形：

```
UINT8 API_LcdDraw(UINT8 line, UINT8 len, UINT8 * pMessage);
```

参数

Line 行号（参阅下表）

Len 像素数据的长度

pMessage 存储消息的缓冲区

	字节 00h (X = 00h)								字节 01h (X = 01h)								…	字节 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

6.4.4. 控制 LCD 滚动功能（Controlling the LCD Scroll）

此函数用于控制 LCD 显示屏的字符滚动功能：

```
UINT8 API_LcdScroll(UINT8 ScrollDirect, UINT8  ScrollSpeed,
                    UINT8 X_Start, UINT8 Y_Start,
                    UINT8 X_end, UINT8 Y_end);
```

参数

ScrollDirect 滚动方向

Bit1	Bit0	滚动方向
0	0	从左至右

Bit1	Bit0	滚动方向
0	1	从右至左
1	0	从上至下
1	1	从下至上

ScrollSpeed 滚动速度

Bit0~Bit3 – 滚动前像素数量

Bit7	Bit6	Bit5	Bit4	滚动周期
0	0	0	0	1 Unit
0	0	0	1	3 Units
0	0	1	0	5 Units
0	0	1	1	7 Units
0	1	0	0	17 Units
0	1	0	1	19 Units
0	1	1	0	21 Units
0	1	1	1	23 Units
1	0	0	0	129 Units
1	0	0	1	131 Units
1	0	1	0	133 Units
1	0	1	1	135 Units
1	1	0	0	145 Units
1	1	0	1	147 Units
1	1	1	0	149 Units
1	1	1	1	151 Units

X_Start X 坐标开始

Y_Start Y 坐标开始

X_end X 坐标结束

Y_end Y 坐标结束

LCD 显示位置（LCD 总尺寸：128x32）

	字节 00h (X = 00h)								字节 01h (X = 01h)								…	字节 0Fh (X = 0Fh)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…	7	6	5	4	3	2	1	0
00h																									
01h																									
02h																									

	字节 00h (X = 00h)								字节 01h (X = 01h)								…	字节 0Fh (X = 0Fh)							
03h																									
04h																									
05h																									
06h																									
07h																									
08h																									
09h																									
…	…																								
1Fh																									

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

6.4.5. 清除 LCD（Clearing the LCD）

此函数用于清除 LCD 显示的内容：

```
UINT8 API_LcdCLear(void) ;
```

参数

无参数。

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

6.4.6. 暂停 LCD 滚动（Pausing the LCD Scroll）

此函数用于暂停 LCD 显示屏的字符滚动功能：

```
UINT8 API_LcdPauseScroll(void) ;
```

参数

无参数。

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

6.4.7. 停止 LCD 滚动（Stopping the LCD Scroll）

此函数用于停止 LCD 显示屏的字符滚动功能并返回初始状态：

```
UINT8 API_LcdStopScroll(void);
```

参数

无参数。

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

6.5. 键盘的 API 函数（Keypad API Functions）

这里介绍了与键盘相关的 API。

6.5.1. 复位键盘 FLASH（Resetting Keypad Flash）

此函数用于复位键盘在 FLASH 中的参数：

```
UINT8 API_ResetKeypadFlash(void)
```

参数

无参数。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.2. 配置键盘 FLASH 参数（Configuring Keypad Flash Parameters）

此函数用于配置键盘在 FLASH 中的参数：

```
UINT8 API_ConfigKeypadFlashParam(UINT8* pfbuffer)
```

参数

pfbuffer　键盘参数

注：*键盘参数包括：*

- 11 个基准控制参数（MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT）
- 24 个临界参数（E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH）
- 2 个去抖参数（Debounce Touch, Debounce Release）

- 24 个充电/放电电流参数以及充电/放电时间参数（CDC0, CDC1, CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11, CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11）

参数的长度必须等于 61，否则 API 会执行失败。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.3. 配置键盘参数（Configuring Keypad Parameters）

此函数用于配置键盘参数：

```
UINT8 API_ConfigKeypadParam(UINT8* pebuffer)
```

参数

pebuffer 键盘参数

注：键盘参数包括：

- 11 个基准控制参数（MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT）
- 24 个临界参数（E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH）
- 2 个去抖参数（Debounce Touch, Debounce Release）
- 24 个充电/放电电流参数以及充电/放电时间参数（CDC0, CDC1, CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11, CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11）

参数的长度必须等于 61，否则 API 会执行失败。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.4. 复位键盘（Resetting the Keypad）

此函数用于复位键盘：

```
UINT8 API_ResetKeypad(void)
```

参数

无参数。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.5. 配置基准控制参数（Configuring Baseline Control Parameters）

此函数用于配置基准控制参数：

```
UINT8 API_ConfigBaselineConParam(UINT8* pbbuffer)
```

参数

pbbuffer 基准控制参数

注：基准控制参数包括：

- MHDR, NHDR, NCLR, FDLR, MHDF, NHDF, NCLF, FDLF, NHDT, NCLT, FDLT

参数的长度必须等于 11，否则 API 会执行失败。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.6. 配置临界参数（Configuring Threshold Parameters）

此函数用于配置临界参数：

```
UINT8 API_ConfigThresholdParam(UINT8* psbuffer)
```

参数

psbuffer 临界参数

注：临界参数包括：

- E0TTH, E0RTH, E1TTH, E1RTH, E2TTH, E2RTH, E3TTH, E3RTH, E4TTH, E4RTH, E5TTH, E5RTH, E6TTH, E6RTH, E7TTH, E7RTH, E8TTH, E8RTH, E9TTH, E9RTH, E10TTH, E10RTH, E11TTH, E11RTH

参数的长度必须等于 24，否则 API 会执行失败。

返回参数

成功　API_OK

失败　API_ERR_CMDFAIL

6.5.7. 配置 CDC 参数（Configuring CDC Parameters）

此函数用于配置充电/放电电流参数：

```
UINT8 API_ConfigCDCParam(UINT8* pcbuffer)
```

参数

pcbuffer 充电/放电电流参数

注：*充电/放电电流参数包括：*

- CDC0, CDC1, CDC2, CDC3, CDC4, CDC5, CDC6, CDC7, CDC8, CDC9, CDC10, CDC11

参数的长度必须等于 12，否则 API 会执行失败。

返回参数

成功 API_OK

失败 API_ERR_CMDFAIL

6.5.8. 配置 CDT 参数（Configuring CDT Parameters）

此函数用于配置充电/放电时间参数：

```
UINT8 API_ConfigCDTParam(UINT8* ptbuffer)
```

参数

ptbuffer 充电/放电时间参数

注：*充电/放电时间参数包括：*

- CDT0, CDT1, CDT2, CDT3, CDT4, CDT5, CDT6, CDT7, CDT8, CDT9, CDT10, CDT11

参数的长度必须等于 12，否则 API 会执行失败。

返回参数

成功 API_OK

失败 API_ERR_CMDFAIL

6.5.9. 读取按键状态（Reading Key Status）

此函数用于读取按键的状态：

```
void API_ReadKeyStatus(void)
```

参数

无参数。

返回参数

按键状态

6.5.10. 读取按键（Reading Keys）

此函数用于读取按键的顺序，并且会保存触摸到的按键：

```
void API_ReadKeys(UINT8 mode, UINT8 X, UINT8 Y, UINT8* ptkey, UINT8
keylength, UINT8 leavecondition, UINT8 waittime)
```

参数

Mode	0X：NORMAL，只保存触摸到的按键 1X：DISPLAY，显示并保存触摸到的按键 2X：PASSWARD，显示'*'并保存触摸到的按键 4X：键按下后发出哔的一声 X0：按键 1, 2 …9, *, 0, # X1：按键 A, B…I, *, _, # X, Y： 参阅"LCD 字符显示"
ptkey	触摸到的按键位置
keylength	按键长度
Leavecondition	完成标志 • 01：按键-10 • 02：按键-11 • 04：按键-12
waittime	用户触摸按键的时间。范围是 0~254s，FFh 表示无限的。

返回参数

API_OK

6.6. 闪存的 API 函数（Flash Memory API Functions）

这里介绍了用于访问闪存的 API。

6.6.1. 写入闪存（Writing to Flash Memory）

此函数用于向闪存写入数据：

```
UINT8 API_FlashMemoryWrite(UINT8* pBuffer, UINT32 WriteAddr, UINT16
NumByteToWrite)
```

参数

pBuffer	获取数据的缓冲区地址
ReadAddr	写入数据的地址（请参阅 M25P40 数据手册）
NumByteToWrite	写入数据的长度

返回值

成功 API_OK

失败 API_ERR_CMDFAIL

示例代码

```
//Write 6 bytes data from Cmdbuffer to 0x000000fd
API_FlashMemoryWrite(Cmdbuffer, 0x000000fd, 0x06)
```

6.6.2. 读取闪存（Reading Flash Memory）

此函数用于从闪存中读取数据：

```
UINT8 API_FlashMemoryRead (UINT8* pBuffer, UINT32 ReadAddr, UINT16
NumByteToRead)
```

参数

pBuffer	存放数据的缓冲区地址
ReadAddr	读取数据的地址（请参阅 M25P40 数据表）
NumByteToWrite	读取数据的长度

返回值

成功 API_OK

失败 API_ERR_CMDFAIL

示例代码

```
//Read 6 bytes data from Resbuffer to 0x000000fd
API_ FlashMemoryRead(Resbuffer, 0x000000fd, 0x06)
```

6.6.3. 擦除闪存（Erasing Flash Memory）

此函数用于擦除闪存：

UINT8 API_FlashMemoryErase(UINT8 sector)

参数

sector sector = 0 擦除整个闪存

sector = 1 擦除第 1 个扇区

sector = 2 擦除第 2 个扇区

sector = 3 擦除第 3 个扇区

sector = 4 擦除第 4 个扇区

sector = 5 擦除第 5 个扇区

sector = 6 擦除第 6 个扇区

sector = 7 擦除第 7 个扇区

sector = 8 擦除第 8 个扇区

sector > 8 忽略

注：请参阅 M25P40 数据手册。

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

示例代码

```
API_FlashMemoryErase(0x04) // erase sector 4
```

6.7. 备份注册表的 API 函数

ACR1283L 提供了一个 84 字节的备份注册表，如果有电池并且防窃启开关是闭合的，就可以用于存储重要的数据。因此数据在系统或者电源重启时仍然保留在该位置，而一旦外壳被强制打开（即防窃启开关开启），数据就会被删除。

6.7.1. 向备份注册表写入数据（Writing Data to Backup Registry）

此函数用于向备份注册表存入数据：

```
UINT8 API_BkpStoreData(UINT8 Addr,UINT8 *pData,UINT8 Len)
```

参数

Addr　数据的地址（0-83）

pData　待写入的数据的指针地址

Len　数据长度

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

示例代码

```
// store 2 bytes data from Cmdbuffer to backup registry address 0
API_BkpStoreData(0, Cmdbuffer, 2)
```

6.7.2. 从备份注册表中读取数据（Reading Data from the Backup Registry）

此函数用于从备份注册表中读取数据：

```
UINT8 API_BkpReadData(UINT8 Addr,UINT8 *pData,UINT8 Len)
```

参数

Addr　数据的地址（0-83）

pData　待读取的数据的指针地址

Len　数据长度

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

示例代码

```
// Read 2 bytes data from backup registry address 0 to Resbuffer
API_BkpReadData(0, Resbuffer, 2)
```

6.8. 加密操作的 API 函数

这部分介绍了 DEC、3DEC 和 AES 函数。

6.8.1. DES 加密操作（Encrypting DES）

此函数用于对数据进行 DES 加密或解密：

```
void API_DES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey, UINT8
*pucInitialVector, UINT8 ucMode);
```

参数

pucData 加密/解密后数据（操作结果）的存放地址

ulDataLen 加密/解密后数据的长度

pucKey 加密密钥（64 位）的存放地址

pucInitialVector 初始向量值的存放地址

ucMode 操作模式：

- mode = 0Eh（加密）
- mode = 0Dh（解密）
- mode = E0h (EBC)
- mode = D0h (CBC)

返回值

无。

6.8.2. 3DES 加密（Encrypting 3DES）

此函数用于 3DES 加密或解密过程：

```
void API_3DES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey, UINT8
*pucInitialVector, UINT8 ucMode);
```

参数

pucData 加密/解密后数据（操作结果）的存放地址

ulDataLen 加密/解密后数据的长度

pucKey 加密密钥（192 位）的存放地址

pucInitialVector 初始向量值的存放地址

ucMode 操作模式：

- mode = 0Eh（加密）
- mode = 0Dh（解密）
- mode = E0h (EBC)
- mode = D0h (CBC)

返回值

无。

6.8.3. AES 加密（Encrypting AES）

此函数用于 AES 加密或解密过程：

```
void API_AES(UINT8 *pucData, UINT32 ulDataLen, UINT8 *pucKey, UINT16
uiKeyLen,UINT8 *pucInitialVector, UINT8 ucMode);
```

参数

pucData 加密/解密后数据（操作结果）的存放地址

ulDataLen 加密/解密后数据的长度

pucKey 加密密钥的存放地址

uiKeyIen 加密密钥的长度（128 位或者 256 位）

pucInitialVector 初始向量值的存放地址

ucMode 操作模式：

- mode = 0Eh（加密）
- mode = 0Dh（解密）
- mode = E0h (EBC)
- mode = D0h (CBC)

返回值

无。

6.9. 其它 API 函数

6.9.1. 延迟毫秒（Delaying ms）

此函数用于延迟 N 个毫秒。N 的值取决于所传入的参数。

```
void API_DelayNms(UINT16 uiVal,UINT8 ucMode);
```

参数

uiVal 延迟的毫秒值（例如：0Ah 表示延迟 10 ms）

ucMode 延迟模式：

- 0 = 阻塞模式（延迟时间结束后，软件才会运行）
- 1 = 非阻塞模式（软件可以执行其它部分，`UINT8 API_BlockingDelayStatus(void)` 会返回延迟时间是否结束。）

返回值

无。

6.9.2. 延迟微秒（Delay us）

此函数延迟 N 个微秒。N 的值取决于所传入的参数。

```
void API_DelayNus(UINT16 uiVal, UINT8 ucMode);
```

参数

uiVal 延迟的微秒值（例如：0Ah 表示延迟 10 us）

ucMode 延迟模式：

- 0 = 阻塞模式（延迟时间结束后，软件才会运行）
- 1 = 非阻塞模式（软件可以执行其它部分，`UINT8 API_BlockingDelayStatus(void)` 会返回延迟时间是否结束。）

返回值

无。

6.9.3. 非阻塞模式延迟时间查询（Unblocking Mode Delay Time Query）

此函数用于查看非阻塞模式延迟函数设置的延迟时间是否已经结束。

```
UINT8 API_BlockingDelayStatus(void);
```

参数

无参数。

返回值

零 延迟时间结束

非零 延迟时间尚未结束

6.9.4. 通过串行端口传输数据（Transferring Data through Serial Port）

此函数用于通过串行端口打印 CCID *XfrBlock* 数据并返回消息或用户定义的数据。输出格式可以选择 HEX 或 ASCII。

```
UINT8 API_UsartTransmit(UINT8 Optl, UINT8 *pData, UINT32 Len)
```

参数

Optl 0xxx xxxxb：ACSII 格式输出

1xxx xxxxb：HEX 格式输出

x000 0000b：打印 CCID *XfrBlock* 数据

x000 0001b：打印 CCID *XfrBlock* 返回信息

x000 0002b：打印用户定义的数据

pData 用户数据指针

Len 用户数据长度

返回值

成功 API_OK

失败 API_ERR_CMDFAIL

6.9.5. 获取当前时间（Getting the Current Time）

此函数用于获取实时时钟的当前时间。

```
void API_GetTime(struct TIME *psTime);
```

参数

psTime 存储时间的可变地址

```
struct TIME{
UINT8 Second;
UINT8 Minute;
UINT8 Hour;
UINT8 Day;
UINT8 Month;
 INT8 Weekday;
UINT16 Year;
UINT8 yDay;
};
```

返回值

无。

示例代码

```
TIME tCurentTime;
API_GetTime(&tCurentTime);
```

6.9.6. 设置时间（Setting the Time）

此函数用于设置实时时钟的当前时间。

```
void API_SetTime(struct TIME *psTime);
```

参数

psTime 存储时间的可变地址

```
struct TIME{
UINT8 Second;
UINT8 Minute;
```

```
UINT8 Hour;
UINT8 Day;
UINT8 Month;
 INT8 Weekday;
UINT16 Year;
UINT8 yDay;
};
```

返回值

成功　API_OK

失败　API_ERR_CMDFAIL

示例代码

```
TIME tCurentTime;
tCurentTime = {59; 59; 23; 31; 12; 6, 2011, 365};
API_ SetTime (&tCurentTime);
```

附录 A. 状态码

状态码（Hex）	描述		说明
FFh	API_ERR_CMDABORT	命令中止	读写器停止了当前命 z 令
FEh	API_ERR_MUTE	无返回	CCID 超时
FDh	API_ERR_PARITE	奇偶校验错误	与卡片通信时发生奇偶校验错误
FCh	API_ERR_LENGTH	长度错误	数据长度错误信息
FBh	API_ERR_HARDWARE	硬件错误	-
F8h	API_ERR_TS	TS 错误	-
F7h	API_ERR_TCK	TCK 错误	-
F6h	API_ERR_PROTOCOL	协议不支持	-
F5h	API_ERR_CLASS	类别不支持	-
E1h~F4h	RFU	未定义	供将来使用
E0h	API_ERR_SLOT_BUSY	卡槽处于忙碌状态	先前的命令正在运行，并且接收到新的命令
DFh~87h	RFU	未定义	供将来使用
86h	API_ERR_TIMEOUT	超时	执行命令超时
85h	API_ERR_CMDFAIL	命令失败	-
84h	API_ERR_FRAM_LEN	读取/写入 FRAM 的长度错误	起始地址与长度之和超过 1FFFh
83h	API_ERR_FRAM_ADDR	读取/写入 FRAM 地址错误	起始地址超过 1FFFh
82h	API_ERR_CMDNOTSUPPORT	不支持此命令	-
81h	API_OK	成功	命令执行成功
80h	API_ERR_INDEX	卡槽编号错误	Slot_ Number > 05h except 08h
7Fh~00h	RFU	未定义	供将来使用

表 2：状态码