Application Note

ZSN700 读卡功能 API 手册

基于ARM Cortex M0 核的32 位微控制器 AN01010700 1.0.01 Date:2020/12/12

类别	内容
关键词	内置读卡功能、API手册
摘要	主要介绍读卡操作的API函数功能

修订历史

版本	日期	原因
1.0.00	2020/04/18	创建文档
1.0.01	2020/12/12	修改文档模板

目 录

1. 适用范围.....1
2. 相关 API 接口.....2
2.1 初始化函数接口.....2
2.2 配置类接口.....2
2.2.1 读寄存器.....2
2.2.2 写寄存器.....2
2.2.3 关闭芯片.....3
2.2.4 设置协议接口.....3
2.2.5 复位芯片.....3
2.2.6 设置天线驱动模式.....3
2.2.7 设置载波暂停.....4
2.3 卡片操作类接口.....4
2.3.1 A 型卡请求.....4
2.3.2 A 型卡防碰撞.....4
2.3.3 A 型卡选择.....5
2.3.4 A 型卡挂起.....5
2.3.5 A 型卡激活.....6
2.3.6 S50 卡/密钥验证.....6
2.3.7 S50 卡/读数据块.....7
2.3.8 S50 卡/写数据块.....7
2.3.9 S50 卡/块值操作.....7
2.3.10 S50 卡/块值设置.....8
2.3.11 S50 卡/块值读取.....8
2.3.12 A 型卡 RATS.....8
2.3.13 A 型卡 PPS.....9
2.3.14 A 型卡取消激活.....9
2.3.15 Ultralight C 类卡数据传输命令.....9
2.3.16 T=CL 命令.....10
2.3.17 数据交互命令.....10
2.3.18 B 型卡请求.....11
2.3.19 B 型卡修改传输属性.....11
2.3.20 B 型卡激活.....12
2.3.21 B 型卡挂起.....12
2.3.22 读取身份证物理 ID.....12
3. 免责声明.....14

1. 适用范围

本文档内容只适用于对 ZSN700 内部的读卡芯片进行操作。操作接口以静态库的形式发布，本文将详细介绍 ZSN700 中读卡功能中所有会用到 API 接口，用户无需过多了解芯片，即可通过本文提供的 API 接口完成对 ZSN700 内部读卡功能的相关操作。对应驱动静态库文件及简单应用示例代码都放 AMetal 仓库中，AMetal 开源于 Github，其开源网址为“https://github.com/zlgopen/ametal”，用户可直接进入此页面下载 AMtel 开源代码包来获取 ZSN700 SDK 的相关最新资料，具体操作参见《AMetal 代码仓库使用说明(TortoiseGit)》。

2. 相关 API 接口

2.1 初始化函数接口

在使用读卡功能之前，可直接通过初始化函数完成对读卡功能的初始化，初始化函数定义如下：

```
/**\brief zsn700_reader 初始化函数 */
am_zsn700_reader_handle_t am_zsn700_reader_init (am_zsn700_reader_dev_t                *p_dev,
                                                 const am_zsn700_reader_devinfo_t      *p_info);
```

该函数将返回一个用户用以操作读卡功能的标准服务句柄 handle，其类型为 am_zsn700_reader_handle_t，具体定义如下：

```
/**
 *\ brief ZSN700_READER 设备句柄
 */
typedef   am_zsn700_reader_dev_t*   am_zsn700_reader_handle_t;
```

由此可见函数返回值为指向读卡功能设备的的指针，也就相当于返回的 handle 等效于 p_dev。

初始化函数共有两个参数：p_dev 和 p_info，其中 p_dev 为用户定义的设备结构体，p_info 为用户定义的相关参数，其具体定义如下：

```
/**
 *\ brief ZSN700_READER 设备信息结构体
 */
typedef struct am_zsn700_reader_devinfo {
    am_zsn700_reader_prot_type_t                iso_type;           /**< \brief 初始使用的协议类型 */
    const   am_zsn700_reader_lpcd_cfginfo_t     *p_lpcd_cfg_info;   /**< \brief lpcd 模式配置信息 */
} am_zsn700_reader_devinfo_t;
```

iso_type 是 ISO/IEC 14443 协议类型参数，分为类型 A 和类型 B 两种，用户根据实际模板卡片支持的协议进行修改。p_lpcd_cfg_info 是读卡功能低功耗卡片侦测功能的初始化配置信息，此类配置信息一般使用默认值，不建议用户进行修改。

2.2 配置类接口

2.2.1 读寄存器

调用以下函数进行寄存器读操作：

```
/**\brief 读 zsn700_reader 寄存器 */
uint8_t   am_zsn700_reader_get_reg (am_zsn700_reader_dev_t   *p_dev,   uint8_t   nregadr);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nregadr：寄存器地址；

返回值是 zsn700_reader 设备对应寄存器的值。

2.2.2 写寄存器

调用以下函数进行寄存器写操作：

```
/**\brief 写 zsn700_reader 寄存器 */
```

```
am_err_t am_zsn700_reader_set_reg (am_zsn700_reader_dev_t    *p_dev,
                                   uint8_t                   nregadr,
                                   uint8_t                   nregval);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nregadr：寄存器地址；

nregval：写入的值，写入值需参考寄存器描述；

成功返回 AM_OK；

参数错误返回 -AM_EINVAL；

传输错误返回 -AM_EIO。

2.2.3 关闭芯片

调用以下函数完成关闭读卡功能的操作：

```
/**\brief 关闭 zsn700_reader */
am_err_t    am_zsn700_reader_close (am_zsn700_reader_dev_t    *p_dev);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

成功返回 AM_OK；

失败返回 AM_ERROR。

2.2.4 设置协议接口

调用以下函数完成设置操作：

```
/**\brief 设置 zsn700_reader 设备 */
uint8_t am_zsn700_reader_config (am_zsn700_reader_dev_t             *p_dev,
                                 am_zsn700_reader_prot_type_t       type);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

type：协议类型，可设置为 AM_ZSN700_READER_PROT_TYPE_ISO14443A_106 或 AM_ZSN700_READER_PROT_TYPE_ISO14443B_106

返回状态见 am_zsn700_reader.h 中的状态码定义。

2.2.5 复位芯片

调用以下函数完成复位芯片操作：

```
/**\brief 复位 zsn700_reader */
uint8_t am_zsn700_reader_reset (am_zsn700_reader_dev_t    *p_dev);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

返回状态见 am_zsn700_reader.h 中的状态码定义。

2.2.6 设置天线驱动模式

调用以下函数完成设置操作：

```
/**\brief 设置天线驱动模式 */
am_err_t am_zsn700_reader_set_tx_mode (am_zsn700_reader_dev_t    *p_dev,    uint8_t    mode);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

mode：驱动模式，0x00：关闭 TX1 和 TX2；

0x01：仅打开 TX1；

0x02：仅打开 TX2；

0x03：同时打开 TX1 和 TX2。

成功返回 AM_OK；

参数错误返回 -AM_EINVAL；

传输错误返回 -AM_EIO。

2.2.7 设置载波暂停

调用以下函数完成设置操作：

```
/**\brief 设置载波暂停 */
am_err_t am_zsn700_reader_pause_carrier (am_zsn700_reader_dev_t    *p_dev,
                                         uint8_t                    pause_ms,
                                         uint8_t                    wait_ms);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

pause_ms：载波暂停时间，单位毫秒；

wait_ms：载波重新开启后的阻塞等待时间，单位毫秒。

成功返回 AM_OK；

参数错误返回 -AM_EINVAL；

传输错误返回 -AM_EIO。

这个命令常用于卡片操作过程中给卡片“断电”，让卡片复位。等待时间可根据卡的功耗而定，如 Mifare1 卡的功耗较小，等待 2~5 毫秒即可，而 CPU 卡功耗较大，需要等待 10 毫秒左右，这段时间其实是给卡片预留足够的上电时间。

2.3 卡片操作类接口

2.3.1 A 型卡请求

调用以下函数完成请求操作：

```
/**\brief A 型卡请求 */
uint8_t am_zsn700_reader_picca_request (am_zsn700_reader_dev_t    *p_dev,
                                        uint8_t                    req_mode,
                                        uint8_t                    p_atq[2]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

req_mode：请求模式可选：AM_ZSN700_READER_PICCA_REQ_IDLE
AM_ZSN700_READER_PICCA_REQ_ALL

p_atq：获取请求应答信息（ATQ）的指针，其值为 16 位；

返回状态见 am_zsn700_reader.h 中的状态码定义。

请求命令是卡激活操作的第一个步骤，处于 IDLE 模式的卡片会响应任意一种请求模式，但处于挂起模式的卡片仅会响应 ALL 类型的请求命令。

2.3.2 A 型卡防碰撞

调用以下函数完成 A 型卡防碰撞操作：

```
/**\brief A 型卡防碰撞 */
uint8_t am_zsn700_reader_picca_anticoll (am_zsn700_reader_dev_t    *p_dev,
                                         uint8_t                    anticoll_level,
                                         uint8_t                   *p_uid,
                                         uint8_t                   *p_real_uid_len);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

anticoll_level：防碰撞等级，可选：ZSN700_READER_PICCA_ANTICOLL_1
ZSN700_READER_PICCA_ANTICOLL_2
ZSN700_READER_PICCA_ANTICOLL_3

p_uid：防碰撞后获取到的卡序列号；

p_real_uid_len：卡序列号的实际大小（字节数）；

返回状态见 am_zsn700_reader.h 中的状态码定义。

请求命令成功之后需要再发送防碰撞命令用于进一步获得卡片序列号，符合 ISO14443A 标准卡的序列号都是全球唯一的，正是这种唯一性，才能实现防碰撞的算法逻辑，理论上若有若干张卡同时在天线感应区内，则这个函数能够找到一张序列号较大的卡来操作。

2.3.3 A 型卡选择

调用以下函数完成 A 型卡选择操：

```
/**\brief A 型卡选择 */
uint8_t am_zsn700_reader_picca_select (am_zsn700_reader_dev_t    *p_dev,
                                       uint8_t                    anticoll_level,
                                       const uint8_t             *p_uid,
                                       uint8_t                    uid_len,
                                       uint8_t                   *p_sak);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

anticoll_level：防碰撞等级，可选：ZSN700_READER_PICCA_ANTICOLL_1
ZSN700_READER_PICCA_ANTICOLL_2
ZSN700_READER_PICCA_ANTICOLL_3

p_uid：前一个防碰撞函数获取的 UID；

uid_len：前一个防碰撞函数获取的 UID 的长度（字节数）；

p_sak：返回的信息，若 bit2 为 1，则表明 UID 不完整；

返回的信息，若 bit2 为 1，则表明 UID 不完整。

卡选择命令是卡激活操作的最后一个步骤，主要是用于确认是否已经收到了卡片的完整序列号。卡的序列号长度有三种：4 字节、7 字节和 10 字节。 4 字节的只要用一级选择即可得到完整的序列号，如 Mifare1 S50/S70 等；7 字节的要用二级选择才能得到完整的序列号，前一级所得到的序列号的最低字节为级联标志 0x88，在序列号内只有后 3 字节可用，后一级选择能得到 4 字节序列号，两者按顺序连接即为 7 字节序列号。

2.3.4 A 型卡挂起

调用以下函数完成 A 型卡挂起操作：

```
/**\brief A 型卡挂起 */
```

```
uint8_t    am_zsn700_reader_picca_halt (am_zsn700_reader_dev_t     *p_dev);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

返回状态见 am_zsn700_reader.h 中的状态码定义。

用于将卡片置于 HALT 状态，在 HALT 状态下，卡将不响应读卡器发出的 IDLE 模式的请求，除非将卡复位或离开天线感应区后再进入，但它会响应读卡器发出的 ALL 请求。一般对一张卡片完成读写操作后，就会将卡片挂起。

2.3.5 A 型卡激活

调用以下函数完成 A 型卡激活操作：

```
/**\brief A 型卡激活 */
uint8_t am_zsn700_reader_picca_active (am_zsn700_reader_dev_t     *p_dev,
                                       uint8_t                     req_mode,
                                       uint8_t                     p_atq[2],
                                       uint8_t                     p_uid[10],
                                       uint8_t                    *p_uid_real_len,
                                       uint8_t                     p_sak[1]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

req_mode：请求模式可选：AM_ZSN700_READER_PICCA_REQ_IDLE
AM_ZSN700_READER_PICCA_REQ_ALL

p_atq：获取请求应答信息（ATQ）的指针，其值为 16 位；

p_uid：存放序列号的缓冲区，长度应该与序列号长度保持一致；

p_uid_real_len：序列号的实际长度；

p_sak：最后一次选择应答 SAK；

返回状态见 am_zsn700_reader.h 中的状态码定义。

A 型卡激活命令是卡请求——防碰撞——卡选择三条命令的集合，相当于内部已经自动做了处理，简化了操作流程，方便用户通过此命令直接获取卡片的序列号。

2.3.6 S50 卡/密钥验证

调用以下函数完成对 S50 卡的密钥验证操作：

```
/**\brief S50 卡密钥验证 */
uint8_t am_zsn700_reader_picca_authent (am_zsn700_reader_dev_t    *p_dev,
                                        uint8_t                    key_type,
                                        const uint8_t              p_uid[4],
                                        const uint8_t              p_key[6],
                                        uint8_t                    nblock);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

key_type：密钥类型，可选值：AM_ZSN700_READER_IC_KEY_TYPE_A
AM_ZSN700_READER_IC_KEY_TYPE_B

p_uid：卡序列号，4 字节；

p_key：密钥，6 字节。默认的密钥都是全 1，即 6 个 0xff；

nblock：需要验证的卡块号，取值范围与卡类型有关，通常取值范围是 0~63；

返回状态见 am_zsn700_reader.h 中的状态码定义。

密钥验证前，需要先完成卡片激活操作，从密钥验证开始的任何操作，包括后续可能的读写数据块的操作，任何一次操作失败了，都需要重新激活卡片并验证密钥。

2.3.7 S50 卡/读数据块

调用以下函数完成 S50 卡的读数据块操作：

```
/**\brief S50 卡读数据块 */
uint8_t am_zsn700_reader_picca_read (am_zsn700_reader_dev_t    *p_dev,
                                     uint8_t                   nblock,
                                     uint8_t                   p_buf[16]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nblock：需要验证的卡块号，取值范围与卡类型有关，通常取值范围是 0~63；

p_buf：存放读取的数据，共 16 字节；

返回状态见 am_zsn700_reader.h 中的状态码定义。

操作前提是对扇区进行了正确的密钥验证，用于对数据块写入 16 字节数据。

2.3.8 S50 卡/写数据块

调用以下函数完成 S50 卡的写数据块操作：

```
/**\brief S50 卡写数据块 */
uint8_t am_zsn700_reader_picca_write (am_zsn700_reader_dev_t    *p_dev,
                                      uint8_t                   nblock,
                                      const uint8_t             p_buf[16]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nblock：需要验证的卡块号，取值范围与卡类型有关，通常取值范围是 0~63；

p_buf：写入数据缓冲区，共 16 字节；

返回状态见 am_zsn700_reader.h 中的状态码定义。

操作前提是对扇区进行了正确的密钥验证，用于读出数据块内的 16 字节内容。

2.3.9 S50 卡/块值操作

调用以下函数完成 S50 卡的块值操作：

```
/**\brief S50 卡块值操作 */
uint8_t am_zsn700_reader_picca_val_operate (am_zsn700_reader_dev_t    *p_dev,
                                            uint8_t                   mode,
                                            uint8_t                   nblock,
                                            uint8_t                   ntransblk,
                                            int32_t                   value);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

mode：值操作的模式，可加或减，可选类型：

加值：AM_ZSN700_READER_PICCA_PICC_ADD

减值：AM_ZSN700_READER_PICCA_PICC_SUB

nblock：进行值操作的块号；

ntransblk：目标传输块号，计算结果值存放的块号；

value：4 字节有符号数；

返回状态见 am_zsn700_reader.h 中的状态码定义。

操作前提是对扇区进行了正确的密钥验证，同时要进行此类操作，块数据必须要有值块的格式，可以使用若值操作的块号与目标传输块号相同，则将操作后的结果写入原来的块内；若值操作块号与目标传输块号不相同，则将操作后的结果写入目标传输块内，目标传输块内的数据被覆盖，原块内的值不变。

2.3.10 S50 卡/块值设置

调用以下函数完成 S50 卡的块值设置：

```
/**\brief S50 卡块值设置 */
uint8_t am_zsn700_reader_picca_val_set (am_zsn700_reader_dev_t    *p_dev,
                                        uint8_t                    nblock,
                                        int32_t                    value);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nblock：待设置块值的数据块号；

value：设置的值；

返回状态见 am_zsn700_reader.h 中的状态码定义。

操作前提是对扇区进行了正确的密钥验证，该函数使用直接写数据的方式，将指定的块格式转换为数值块，并初始化数值块的值，也可以使用该函数改变数值块的值。

2.3.11 S50 卡/块值读取

调用以下函数完成 S50 卡的块值读取：

```
/**\brief S50 卡块值读取 */
uint8_t am_zsn700_reader_picca_val_ get (am_zsn700_reader_dev_t    *p_dev,
                                         uint8_t                    nblock,
                                         int32_t                   *p_value);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

nblock：待读取块值的数据块号；

p_value：获取值的指针；

返回状态见 am_zsn700_reader.h 中的状态码定义。

操作前提是对扇区进行了正确的密钥验证，用于读取数据块里的块值，待读取的数据块的数据存放格式必须是块值格式。

2.3.12 A 型卡 RATS

调用以下函数完成 A 型卡的 RATS 命令发送：

```
/**\brief A 型卡 RATS 命令 */
uint8_t am_zsn700_reader_picca_rats_get (am_zsn700_reader_dev_t    *p_dev,
                                         uint8_t                    cid,
                                         void                      *p_rats,
```

```
                                        uint32_t                buf_size,
                                        uint32_t                *p_nbytes);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

cid：卡识别号，可设定范围 0—14；

p_rats：请求应答与响应信息；

buf_size：响应信息的缓冲区大小；

p_nbytes：请求应答与响应命令的返回数据的字节长度；

返回状态见 am_zsn700_reader.h 中的状态码定义。

RATS 是 ISO14443-4 协议命令，这各命令用于激活 ISO14443-4 协议当中定义的半双工传输协议，卡片被激活后才能使用这条命令，同时一般只有针对 A 类的 CPU 卡才会使用到这条命令。

2.3.13 A 型卡 PPS

调用以下函数完成 A 型卡的 PPS 命令发送：

```
/**\brief A 型卡 PPS 命令 */
uint8_t am_zsn700_reader_picca_pps_set (am_zsn700_reader_dev_t *p_dev,
                                        uint8_t                 flags);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

flags：输入参数，通信速率设置的编码字节：b3:2 -- PCD -> PICC 的位率编码

b1:0 -- PICC -> PCD 的位率编码

- (00)b -> 106Kb
- (01)b -> 212Kb
- (10)b -> 424Kb
- (11)b -> 847Kb

返回状态见 am_zsn700_reader.h 中的状态码定义。

PPS 命令是 ISO14443-4 协议命令，成功发送 RATS 命令后才可以选择性的发送这条命令，用于设定读卡器与卡片之间的通信速率。

2.3.14 A 型卡取消激活

调用以下函数对 A 型卡进行去激活操作：

```
/**\brief A 型卡取消激活命令 */
uint8_t   am_zsn700_reader_picca_deselect (am_zsn700_reader_dev_t   *p_dev);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

返回状态见 am_zsn700_reader.h 中的状态码定义。

该命令是 ISO14443-4 协议命令，用于将已经激活了 ISO14443-4 协议的卡片设置成 HALT 状态，因此只有成功执行过 RATS 命令后才能使用这条命令。处于这个状态的卡片可以用请求码为 ALL 的卡激活命令去重新激活。

2.3.15 Ultralight C 类卡数据传输命令

调用以下函数进行 Ultralight 卡数据传输操作：

```
int am_zsn700_reader_mifare_ultralight_transfer (am_zsn700_reader_handle_t     handle,
                                                 const void                   *p_txbuf,
                                                 void                         *p_rxbuf,
                                                 uint32_t                     *p_rx_nbytes);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

txbuf：传输数据缓冲区；

p_rxbuf：接收数据缓冲区；

p_rx_nbytes：接收到的数据字节数；

该命令属于芯片扩展功能，用于芯片向卡片发送任意长度组合的数据串，例如针对 NXP 新推出的 NTAG213F 是属于 Ultralight C 系列卡片，但是该卡片又新添加了扇区数据读写密钥保护功能。而这个密钥验证命令即可利用此命名传输命令来实现。

2.3.16 T=CL 命令

调用以下函数进行传输操作：

```
/**\brief 传输命令 */
uint8_t am_zsn700_reader_picca_transfer (am_zsn700_reader_dev_t *p_dev,
                                         const void              *p_txbuf,
                                         uint32_t                 n_tx,
                                         void                    *p_rxbuf,
                                         uint32_t                 buf_size,
                                         uint32_t                *p_n_rx);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

p_txbuf：传输数据缓冲区；

n_tx：需要传输数据的字节数；

p_rxbuf：接收数据缓冲区；

buf_size：接收缓存区大小；

p_n_rx：接收到的字节数；

返回状态见 am_zsn700_reader.h 中的状态码定义。

T=CL 是半双工分组传输协议，ISO14443-4 协议命令，用于读写器与卡片之间的数据交互，一般符合 ISO14443 协议的 CPU 卡均用该协议与读写器通信。调用该命令时只需要将 CPU 卡 COS 命令的数据作为输入即可，其他的如分组类型、卡标识符 CID、帧等待时间 FWT、等待时间扩展倍增因子 WTXM（waiting time extensionmultiplier），等等由该命令自动完成。

2.3.17 数据交互命令

调用以下函数进行传输操作：

```
/**\brief 数据交互命令 */
uint8_t am_zsn700_reader_exchange_block (am_zsn700_reader_dev_t *p_dev,
                                         uint8_t                 *sendbuf,
                                         uint32_t                 sendlen,
                                         uint8_t                 *recvbuf,
                                         uint32_t                 recvbufsize,
                                         uint16_t                *recvlen,
```

```
                                        uint8_t         nwtxm_crc,
                                        uint8_t         nfwi);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

sendbuf：待发送的数据；

sendlen：发送数据的长度；

recvbuf：待接收的数据缓存区；

recvbufsize：接收缓存区大小；

recvlen：接收到的数据长度；

nwtxm_crc：是否使能 CRC 校验，1 为使能，0 为禁能；

nfwi：超时时间设置（实际超时时间为 1 << nfwi ）；

返回状态见 am_zsn700_reader.h 中的状态码定义。

该命令用于读写器与卡片的数据交互，能传输任意数据，并控制是否在数据后面自动添加 CRC 校验。与 T=CL 传输命令不同，数据交互命令发送的数据不遵循半双工传输协议的帧格式。

2.3.18 B 型卡请求

调用以下函数完成 B 型卡请求操作：

```
/**\brief B 型卡请求命令 */
uint8_t am_zsn700_reader_piccb_request (am_zsn700_reader_dev_t *p_dev,
                                        uint8_t         req_mode,
                                        uint8_t         afi,
                                        uint8_t         slot_time,
                                        uint8_t         atqb[12]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

req_mode：请求模式，可选值：ZSN700_READER_PICCB_REQ_IDLE

ZSN700_READER_PICCB_REQ_ALL

afi：应用标识，一般情况下为 0x00 即可；

slot_time：时隙总数，0 ~ 4；

atqb：返回对 REQB 命令的响应，长度为 12；

返回状态见 am_zsn700_reader.h 中的状态码定义。

该函数作为卡的请求操作，只要符合 ISO14443B 标准的卡都应能发出响应。在调用该命令前，需要先将使用设置协议接口命令，将协议接口类型设置为 Type B 类型。

2.3.19 B 型卡修改传输属性

调用以下函数完成 B 型卡请求操作：

```
/**\brief B 型卡修改传输参数 */
uint8_t am_zsn700_reader_piccb_attrib (am_zsn700_reader_dev_t   *p_dev,
                                       const uint8_t            pupi[4],
                                       uint8_t                  cid,
                                       uint8_t                  protype);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

pupi：大小为 4 字节的标识符；

cid：取值范围为 0 - 14, 若不支持 CID，则设置为 0；

protype：bit3 : 1-PCD 中止与 PICC 继续通信,0-PCD 与 PICC 继续通信；

bit2 ~ bit1 : 11 : 10 etu + 512 / fs

10 : 10 etu + 256 / fs

01 : 10 etu + 128 / fs

00 : 10 etu + 32 / fs

bit0 : 1 遵循 ISO14443-4，0 不遵循 ISO14443-4（身份证该位必须为 1）

返回状态见 am_zsn700_reader.h 中的状态码定义。

该命令用于 B 型卡修改传输属性，执行该命令时必须先成功执行一次 request 命令。

2.3.20 B 型卡激活

调用以下函数完成 B 型卡激活操作：

```
/**\brief B 型卡激活 */
uint8_t am_zsn700_reader_piccb_active (am_zsn700_reader_dev_t *p_dev,
                                       uint8_t                 req_mode,
                                       uint8_t                 afi,
                                       uint8_t                 uid[12]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

req_mode：请求模式，可选值：ZSN700_READER_PICCB_REQ_IDLE

ZSN700_READER_PICCB_REQ_ALL

afi：应用标识，一般情况下为 0x00 即可；

uid：UID 信息，前 4 字节为 PUPI 中间 4 字节为应用参数，后 4 字节为参数信息；

返回状态见 am_zsn700_reader.h 中的状态码定义。

该命令用于激活 B 型卡片，在调用该命令前，需要先将使用设置协议接口命令，将协议接口类型设置为 Type B 类型。

2.3.21 B 型卡挂起

调用以下函数完成 B 型卡挂起操作：

```
/**\brief B 型卡挂起 */
uint8_t am_zsn700_reader_piccb_halt (am_zsn700_reader_dev_t    *p_dev,
                                     const uint8_t             pupi[4]);
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

pupi：大小为 4 字节的标识符；

返回状态见 am_zsn700_reader.h 中的状态码定义。

该函数用于 B 型卡挂起，在执行挂起命令前，必需先执行成功过一次请求命令。执行挂起命令成功后，卡片处于挂起状态，模块必需通过 ALL 方式请求卡片，而不能用 IDLE 方式请求。

2.3.22 读取身份证物理 ID

调用以下函数完成身份证 ID 读取操作：

```
/**\brief    身份证物理 ID 读取  */
uint8_t am_zsn700_reader_idcard_readr_id (am_zsn700_reader_dev_t *p_dev,
                                          uint8_t                 id[10])
```

其中 p_dev：初始化函数返回的标准服务句柄（即 handle）；

id：身份证 ID 的数据缓冲区

返回状态见 am_zsn700_reader.h 中的状态码定义。

该函数用于身份证 ID 读取，执行该函数之前，需要执行 B 卡激活操作后，方可读取成功，其中返回数据前 8 位为身份证 ID，后两位为操作结果标识，若操作成功则会返回 0x90,0x00。

3. 免责声明

本着为用户提供更好服务的原则，广州致远微电子有限公司（下称“致远微电子”）在本手册中将尽可能地为用户呈现详实、准确的产品信息。但介于本手册的内容具有一定的时效性，致远微电子不能完全保证该文档在任何时段的时效性与适用性。致远微电子有权在没有通知的情况下对本手册上的内容进行更新，恕不另行通知。为了得到最新版本的信息，请尊敬的用户定时访问立功科技官方网站或者与致远微电子工作人员联系。感谢您的包容与支持！

专业 . 专注成就梦想

Dreams come true with professionalism and dedication.