AN2598

应用笔记

STM32F101xx 和 STM32F103xx 中的智能卡接口

简介

本文档描述了基于 STM32F10xxx USART 外围模块的固件和硬件智能卡接口解决方案。该固件和硬件包的目的是提供相应的资源，使用户在智能卡模式下使用 USART 外围模块的应用程序的开发变得更加便捷。

这个固件接口包括支持 ISO 7816-3/4 规格的库文件。同时提供应用程序示例。

本文档及其相关的固件可以从ST的网站下载：www。st。com

目录

AN2598 1

应用笔记 1

STM32F101xx和STM32F103xx中的智能卡接口 1

1 智能卡接口表述 5

1.1 简介 5

1.2 外部接口 5

1.3 协议 6

1.4 智能卡时钟发生器 7

2 智能卡阅读器硬件连接 7

3 ISO 7816：协议概述 8

3.1 简介 9

3.2 ISO 7816-2 – 引脚分布 9

4 ISO 7816 – 3 – 电信号和传输协议 10

4.1 智能卡上电启动和重置 11

4.2 数据传输 12

4.3 回复重置信号（ATR） 14

5 ISO 7816 – 4 – 智能卡命令 16

5.1 T0 协议 16

5.2 应用层协议 19

5.2.1 ISO 7816-4 APDU 20

5.2.2 文件系统API 21

5.2.3 ISO 7816-4 函数 23
5.2.4 安全API 25
6 智能卡接口库：描述 27
6.1 文件组织 27
6.2 智能卡接口库函数 27
6.2.1 SC_Handler function 28
6.2.2 SC_PowerCmd 32
6.2.3 SC_Reset 33
6.2.4 SC_PTSConfig 34
6.3 怎样发送APDU命令给智能卡 35
6.3.1 SC_GET_A2R 35
6.3.2 SELECT_FILE 36
6.3.3 SC_GET_RESPONSE 36
6.3.4 SC_READ_BINARY 37
6.3.5 SC_CREATE_FILE 38
6.3.6 SC_UPDATE_BINARY 38
6.3.7 SC_VERIFY 39
6.4 奇偶错误管理 40
6.4.1 数据由卡发送到阅读器 40
6.4.2 数据由阅读器发送到卡 40
7 智能卡接口示例 40
7.1 固件包描述 41
7.1.1 FWLib文件夹 41
7.1.2 Smartcard_AN文件夹 41
7.2 固件描述 42
7.2.1 智能卡启动：重置应答（A2R） 42
7.2.2 按指定路径读一个文件 43

7.2.3 使能/禁止PIN1（CHV1）编码 44
7.2.4 校验PIN1（CHV1）编码 44
8 结束语 45
9 版本历史 45
10 版权声明： 45

1 智能卡接口表述

1.1 简介

智能卡接口是在 USART 智能卡模式下开发的。关于 USART 寄存器的描述 ,请参阅 STM32F10xxx 用户参考手册。USART 智能卡模式支持 ISO 7816-3 标准中定义的异步协议智能卡。

在智能卡模式使能的情况下，USART 必须配置如下：

- 8 位数据位加上奇偶校验
- 0。5 或 1。5 位停止位

一个 5 位的预分频器和智能卡时钟生成器为智能卡提供时钟。

与软件协力工作的 GPIO 引脚用来提供与智能卡交互的其他功能。

软件中不处理 ISO 7816-3 中定义的反向信号传输约定,反转数据和最高有效位优先。

1.2 外部接口

Table 1 智能卡引脚

STM32F10xxx 引脚	智能卡引脚	功能
USART CK	CLK	智能卡时钟
USART CK	IO	IO 串行数据：漏极开路输出
Any GPIO	RST	重置智能卡

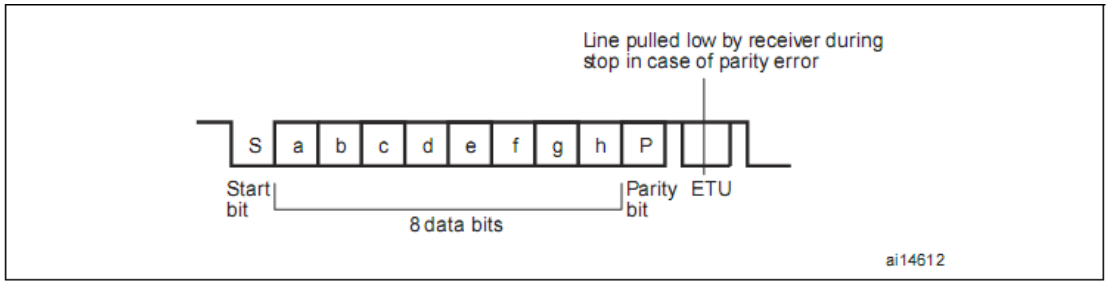
Any GPIO	V_{CC}	提供电压
Any GPIO	V_{PP}	编程电压

Smartcard_RST (智能卡重置)、 Smartcard_3/5V (3 V 或 5 V)、Smartcard_CMDVCC (管理 V cc)以及 Smartcard_OFF 信号 (智能卡检测信号) 由软件控制下 I/O 端口的 GPIO 位提供。把端口的 GPIO 位编程为漏极开路转换功能输出模式,会使 USART_TX 数据信号以正确的驱动连接到智能卡 IO 引脚,使时钟产生器连接到配置为转换功能推挽输出模式的 Smartcard_CLK 引脚。

1.3 协议

ISO 7816-3 标准以时间单位的形式，为异步协议定义了称作 ETUs (elementary time units)的位时间，它与智能卡的时间频率输入相联系。一个 ETU 的长度是一个位时间。USART 接收器和发送器在内部通过 Rx_SW 相连接。将数据从 STM32F10xxx 传输到智能卡，USART 必须被设置成智能卡模式。

图 1　　ISO 7816-3 异步协议



1.4　智能卡时钟发生器

智能卡时钟发生器为已连接的智能卡提供时钟信号。智能卡使用这个时钟产生在智能卡与另一个 USART 之间进行串行通信的波特率。如果卡上有 CPU，该时钟同样为被 CPU 使用。

当卡上的 CPU 在运行代码时，波特率可以改变，或者智能卡的性能可以被提升，这就要求对智能卡接口进行操作时,时钟速率可以做出调整。协商时钟速率和改变时钟速率的协议，在 ISO7816-3 标准中有详细描述。

这个时钟被用作智能卡的 CPU 时钟，所以更新微控制器时钟频率必须和智能卡时钟同步，使得时钟高或低脉冲宽度不小于新值或旧值中的一个。

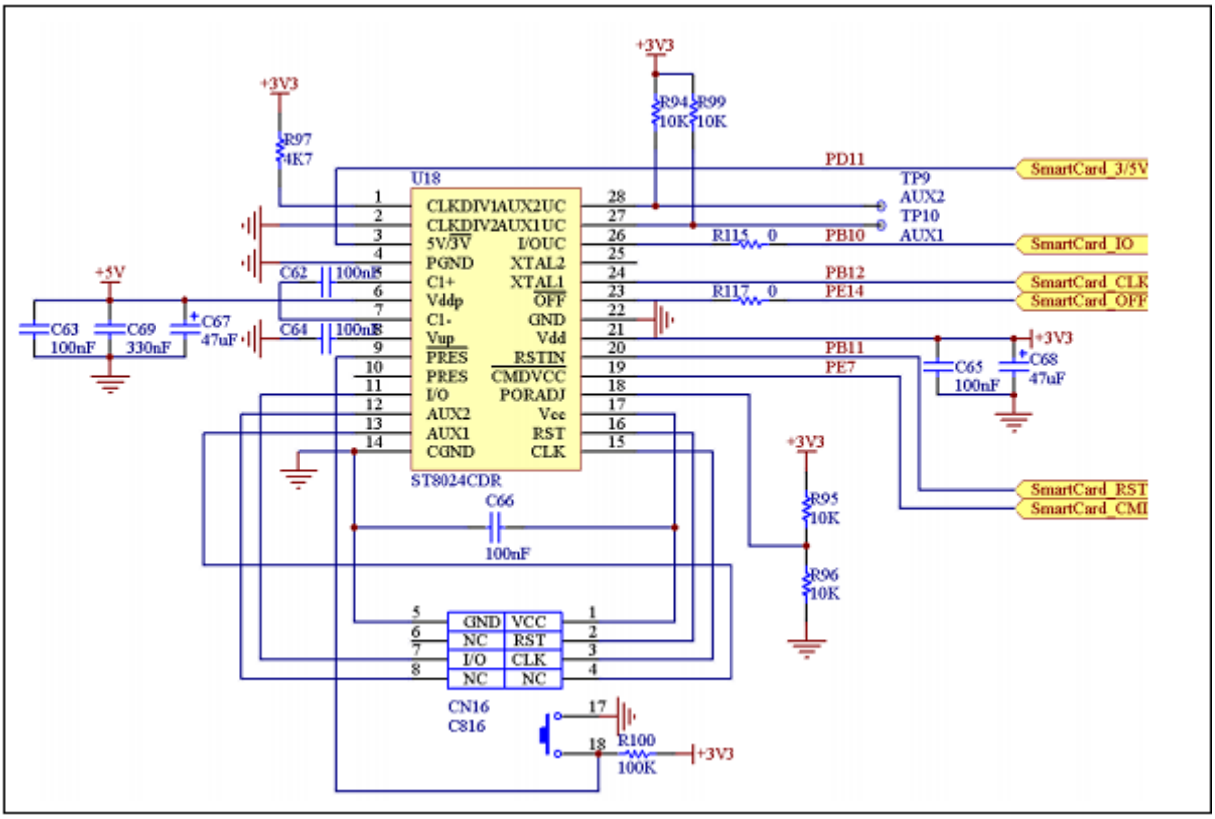
2 智能卡阅读器硬件连接

与智能卡进行交互，需要使用 ST8024 设备。ST8024 是一个针对于异步 3V、5V 智能卡的完善的低成本模拟接口。它连接在智能卡和 STM32F10xxx 之间，并且只需要很少的外部部件来实现保护和控制功能。

Table 2　STM32F10xxx 和智能卡连接

STM32F10xxx 引脚	智能卡引脚	功能
USART3_CK: PB12	C3: CLK	智能卡时钟：alternate function push-pull
USART3_TX: PB10	C7: IO	IO 串行数据：alternate function open drain
PB。11	C2: RST	重置智能卡：推挽输出
PE。07	C1: V_{CC}	提供电压：推挽输出
PE。14	OFF	智能卡检测：悬浮输入
PD。11	3/5V	3V 或 5V：推挽输出

图 2　　智能卡接口硬件连接



3 ISO 7816：协议概述

3.1 简介

“ISO 7816 :身份证--带有触点的集成电路卡“ 能够把相对简单的, 容易被伪造,偷盗,丢失的识别卡转变为一个防篡改的智能集成电路卡,人们一般称之为智能卡。ISO 7816 包括至少 6 个经审核的部分和有一些新增部分，如下：

- 第一部分：物理特性
- 第二部分：Dimensions and location of the contacts
- 第三部分：电器接口和传输协议
- 第四部分：协议类型选择的更正版，第 2 版
- 第五部分：相互的组织、安全和命令
- 第六部分：应用提供商的登记

3.2 ISO 7816-2 – 引脚分布

ISO 7816-2 指定一个有 8 电器触点的 ICC ,它位于智能卡表面的标准位置。他们分别是 C1 至 C8。其中一些触点与嵌在智能卡内微处理器芯片相连 ;另一些触点目前没有使用 ,他们是用作今后扩展的。图 3 给出了触点的位置。

图 3　　智能卡触点

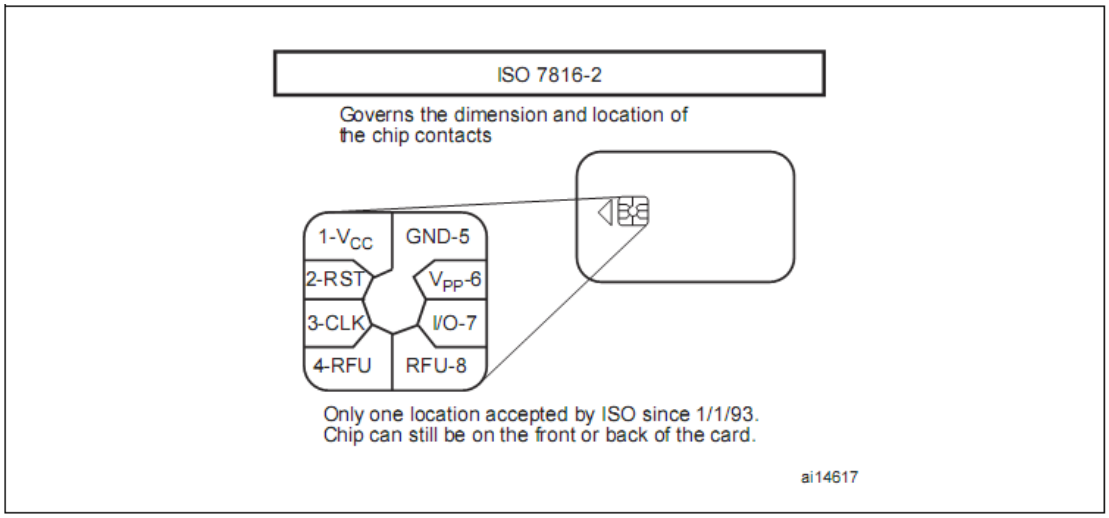


表 3　　引脚分配

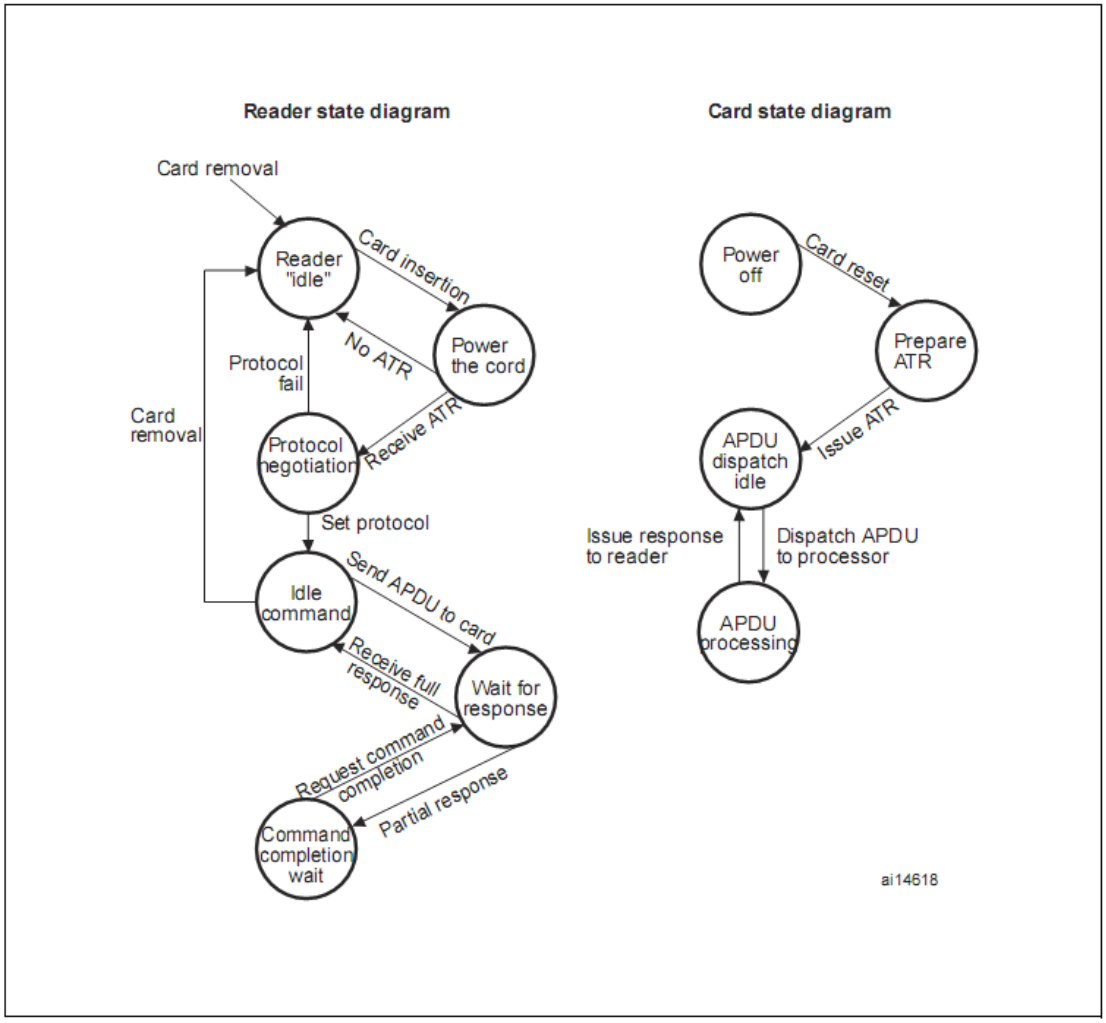
引脚	功能
C1	Vcc=5V 或者 3。3V
C2	重置
C3	时钟
C4	RFU
C5	GND
C6	Vpp
C7	I/O
C8	RFU

4 ISO 7816 – 3 – 电信号和传输协议

IS0 7816-3 开始研究智能卡智能方面的规范。该标准描述了智能卡和阅读器之间的关系，其中智能卡作为从设备，阅读器作为主设备。交流是建立在阅读器通过触点给智能卡发送信号，然后智能卡作出回应的基础上。

智能卡和阅读器的交流过程，依照图 4 中的不同状态转换。

图 4　　阅读器和智能卡状态机



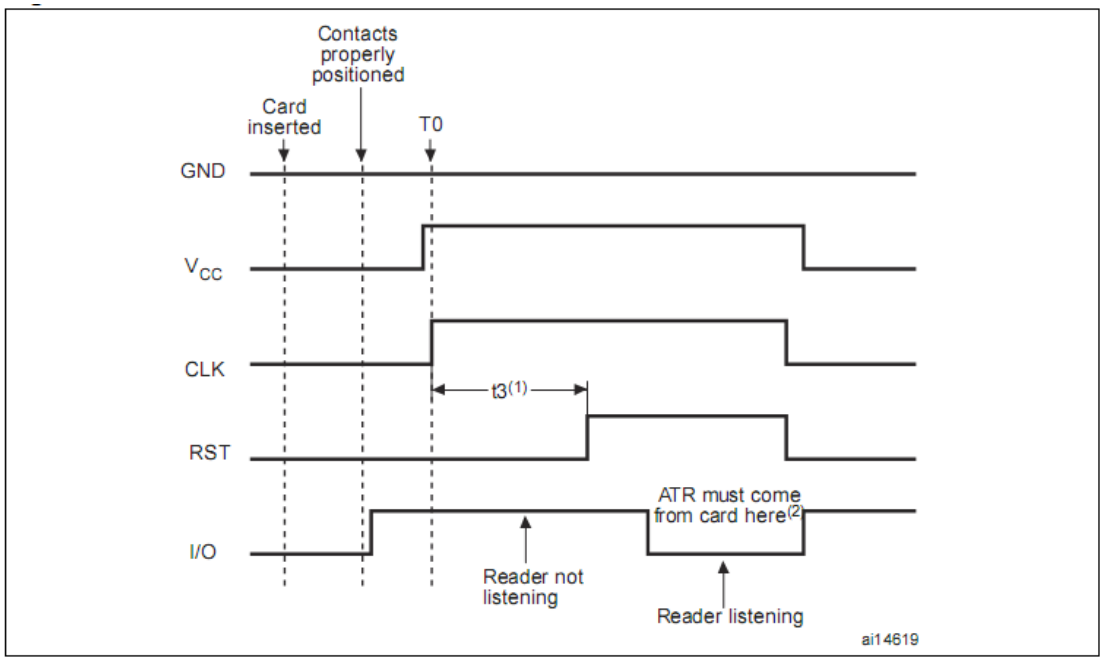
交流信道是单线程的。一旦阅读器给智能卡发送一个命令，阅读器将被阻塞，直到接受到回复。

4.1　智能卡上电启动和重置

当智能卡被插入到一个阅读器，没有电源提供给任何触点。如果把电源提供给了错误的触点，卡上的芯片可能被严重损坏，这种情形很可能发生在触点已经上电的情况下去插智能卡。在可接受的触点机械容许条件下（针对于阅读器），在边沿探测器尚未决定智能卡是否正确分配前,触点保持无电状态。

当阅读器发现智能卡已经正确地插入，电源就会提供给智能卡。首先，触点进入一个空闲状态，如表 3 所示。然后一个重置信号通过 RST 接触线发送给智能卡。空闲状态发生在电源（Vcc）触点进入稳定 5V 工作电压状态的时候。初始化电压 5V 总是最先提供，虽然有些微处理器芯片在 I/O 状态时工作在 3V 电压下。阅读器的 I/O 触点设置成接收模式，并且提供稳定的时钟（CLK）。重置引线处于低状态。在阅读器启动一个将重置线置位高状态的有效重置序列前，重置线在低状态至少要保持 40000 个 CLK 周期。

图 5　　回应重置



t3 = 40 000 时钟周期

智能卡在 RST 升高后，必须在 400 时钟周期和 40 000 时钟周期之间发出 ATR 信号。

4.2　数据传输

在阅读器和智能卡之间进行数据传输，需要通过两个触点引线的协商：CLK 和 I/O。I/O 引线每一个单位时间传送一个比特的信息，其中单位时间由相对于 GND 的电压产生的 CLK 定义的。一个’1’

位可以通过+5V 或 0V 电压来传送。现实中的使用是由智能卡决定的，并且通过 ATR“初始化字符”，传送给阅读器，这也被称作 TS。传送一个字节的信息，在 I/O 引线上有 10 个比特通过；最先的是“开始位”，最后一位通常是偶校验位。I/O 引线可以是（一个位期间）高（H）或者低（L）状态，TS 字符为 HLHHLLLLLH 时，智能卡用它来表示“翻转约定”，这意味着 H 表示 0，L 表示 1。TS 符号为 HLHHLHHHLLH 时，智能卡用它来表示“直接约定”，这意味着 H 表示 1，L 表示 0。

直接约定和翻转约定同样也控制在智能卡和阅读器之间传输的每个字节的比特顺序。在直接约定下，起始位后的第一个比特是字节的低次序位，然后依次是高次序位。在翻转约定下，起始位后的第一个比特是字节的高次序位，然后依次是低次序位。每个传输的字节必须使用偶校验；这意味着字节中所有 1 的个数，包括校验位，必须是偶数个。

I/O 引线包含一个半双工信道；这表示，智能卡或者阅读器可以在同一个信道上传输数据，但是两者不能同时传输。所以作为启动顺序的一部分，阅读器和智能卡都进入接收状态，侦听 I/O 引线。当重置操作开始，阅读器仍旧在接收状态，但是智能卡必须进入发送状态，为的是能够发送 ATR 至阅读器。从这点上来说，信道两端在发送和接收两个状态之间互相交替。在半双工信道中，没有一种可靠的方法使得任何一方可以异步改变状态，从发送状态改变到到接收状态，或者从接收状态到发送状态。如果需要使用这种改变，那一方需要进入接收状态，并且允许过程操作超时；然后阅读器一方总是会尝试进入发送状态，重新建立一个认可的序列。CLK 和 I/O 引线支持不同数据传输速度的范围非常广。速度由智能卡定义，并且通过 ATR 中的可选项字符传送给阅读器。传输速度在 I/O 引线上通过一个比特时间设定，这意味着，通过采样 I/O 引线去读取一个比特以及每个后继比特，来建立时间间隔。这个时间定义为一个基本时间单位（ETU），它建立在几个因数之间的线性关系基础上。需要注意的是，TS 字符在 ETU 定义产生之前已经返回。这可能是因为 ETU 在 ATR 序列过程中总是被指定为 ETU0=372/CLK 频率，其中 CLK 频率总是在 1MHz 和 5MHz 之间。实际上，这个频率几乎总是选择数据传输的初始速度，9600 比特每秒。

4.3 回复重置信号（ATR）

一旦 RST 信号从阅读器发送至智能卡，智能卡必须必须在接受到 ART 第一个字符的 40 000CLK 周期内回复。智能卡可能会因为某些原因不回复 ATR，其中最可能的原因是智能卡没有正确插入阅读器（可能上下颠倒）。在某些实例中，智能卡可能因为被损坏而无法工作。不管是什么情况，只要 ATR 没有在预计的时间内返回，阅读器就应该发送一个序列关闭智能卡。在这个序列中，阅读器置 RST、CLK 和 I/O 引线为低，并且降低 Vcc 引线至名义上的 0（即少于 0。4V）。

ATR 是一串在成功的启动序列后，从智能卡返回至阅读器的字符序列。ISO/ICE 7816-3 中定义，ATR 包括 33 或者更少的字符，包括以下内容：

- TS - 一个强制的初始化字符
- T0 – 一个强制的格式字符
- TAi TBi TCi TDi – 可选的接口字符
- T1，T2，TK – 可选的历史相关字符
- TCK – 一个有条件的检测字符

历史相关字符用来指明智能卡制造商或者提供商。这些字符通常用来传递类型、型号和具体智能卡的使用方法。当在这种方式下使用时，历史相关字符提供了一种机制，在这种机制下，系统能够自动检测插入的智能卡的使用方法（在系统内），并且相应地初始化其他操作（或软件）。检测字符提供了一个检查 ATR 完整性的机制；即指从智能卡发送字符到阅读器的过程中是否有传输错误。

ATR 的结构在表 4 中详细给出。正如上面讨论的，初始化 TS 字符用来建立阅读器和智能卡之间的比特信号、比特次序约定。T0 字符用来标识接下来的接口字符或者历史相关字符是否存在。接口字

符用来给出 I/O 信道特性，包括智能卡和阅读器交换命令（阅读器至智能卡）和回答（智能卡至阅读器）过程中使用的协议。历史相关字符假如存在，被用来把智能卡制造商具体信息，从智能卡传递给阅读器，然后再给阅读器的应用系统。没有一个已确定的标准来指定历史相关字符中的内容。

ATR 序列的总长度被限制在 33 个字节以内，并且遵循以下格式：

表 4　　ATR 结构

	字符 ID	描述
初始化字段	TS	强制初始化字符
格式字段	T0	指示接口字符的存在
接口字段	TA1	全局的，编码 F1 和 D1
	TB1	全局的，编码 11 和 PI1
	TC1	全局的，编码 N
	TD1	编码 Y2 和 T
	TA2	专用的
	TB2	全局的，编码 PI2
	TC2	专用的
	TD2	编码 Y3 和 T
	TA3	TAi，Tbi 和 TCi 是专用的
	…TDi	编码 Yi+1 和 T
	T1	智能卡专用信息
历史相关字	…TKi	（最多 15 个字符）

段		
检测字段	TCK	可选的检测字符

5 ISO 7816 – 4 – 智能卡命令

先前的的章节讨论了应答复位机制（ATR），这个机制建立了智能卡与阅读器之间的基本交流信道。这个信道是一个半双通的物理通道。这一章将介绍在这个物理顶层的更多复杂的协议的使用。

连接层的通信协议停留在物理信道的顶层，提供了在阅读器和智能卡之间的无差错通信。一旦连接层的协议建立，应用层的协议也就可以定义了。ISO 7816-4 定义了两个这样的应用层协议：

文件系统 API 提供了一系列操作文件的函数（例如，读，写，选择等）。

安全服务 API 允许智能卡和阅读器相互鉴别它们自己，并把智能卡和阅读器之间交换的数据加密。

ISO 7816 - 4 定义了一个支持应用协议 API 的协议信息结构。这个信息结构包括应用协议数据单元（APDUs），这个数据单元是用来通过连接层协议在阅读器应用层和智能卡应用层交换的。

这一章将提供文件访问和安全 API 的概况。

5.1 T0 协议

T0 协议是一个面向字节的协议，也就是说一个字符通过在阅读器和智能卡之用的信道进行传输。另外通过查看奇偶效验位来处理每个字节上的错误，如果接受数据的奇偶位与发送数据的不否，这个错误是一定会发生的。在 T0 协议中，在检验到奇偶效验出错的情况下，将会产生一个重传信号。，这是通过保持 I/O 线为低电平来完成的（通常 I/O 线在传送字符之前是高电平的）。当发送端检验到这个后，发送端将会重新发送没有正确接收的数据。

阅读器和智能卡交换被称为传送协议数据单元(TPDU)的数据结构。它包括两个独立的结构：

从阅读器发送到智能卡上的命令。

从智能卡发送到阅读器上的应答。

命令头包括下面的五个域，每个域一个字节：

CLA：设置建立一个指令集的一类命令指示。

INS：说明来自指令集内的指令。

P1：用来说明由[CLA，INS]指令使用的地址。

P2：也是用来说明由[CLA，INS]指令使用的地址。

P3：作为[CLA，INS]指令的执行的一部分,说明传送或来自卡上的数据字节数。

CLA 的每一个值说明了一个专门应用的指令集。**表 5** 介绍了指令集的一些值。

表 5。 CLA 指令集定义

CLA 字节	指令集
0x	ISO 7816 - 4 指令（文件和安全）
10 to 7F	将来使用的保留值
8x or 9x	ISO 7816 - 4 指令
Ax	说明应用/厂商的指令
B0 to CF	ISO 7816 - 4 指令
D0 to FE	说明应用/厂商的指令
FF	协议类型选择的保留值

Ins 字节用来指出通过 CLA 值区分的一类指令中的具体指令。**表 6** 列出了 ISO 7816 - 4 标准中用来访问文件系统和安全功能的指令。

表 6。 ISO 7816 - 4 INS 码

INS 值	命令名	INS 值	命令值
0E	清除二进制位	C0	获得响应
20	校验	C2	封装
70	管理信道	CA	获得数据
82	外部鉴别	D0	写二进制位
84	获得查询码	D2	写记录
88	内部鉴别	D6	更新二进制位
A4	选择文件	DA	放入数据
B0	读二进制位	DC	更新记录
B2	读取记录	E2	附加记录

参数 P1 和 P2 定义在连接层，实际上却取决于应用层的具体指令。它们对各种应用层指令提供控制和地址参数。例如，在选择文件指令中，P1 用来指出文件如何被涉及（通过标识，名字，路径等），P2 提供了更加详细的说明，如选择哪一个文件。P3 定义了 INS 具体指令执行期间传送字节数量。数据移动通常是以卡为中心的，也就是说,流出是指数据从卡输送到阅读器，流入是指数据从阅读器输送到卡上的。

对于从阅读器发送的每一个 TDPU 命令，响应的 TPDU 是由卡发送的。响应包括三个必需域和一个可选域（所有域都是一个字节长）

ACK：指出卡收到了[CLA，INS]命令。

NULL：用来卡上 I/O 信道的流控制信号。（发送到阅读器上的）信号表示卡仍然在处理命令，因此阅读器在发送其它信号前必须等待。

SW1：当前命令的状态响应。

SW2:（可选的）也传送到阅读器的状态响应。

ACK 字节是来自命令 TPDU 的 INS 字节的重复。如果响应没有在规定时间到达阅读器，阅读器将会初始化一个 RST 序列号来重启在阅读器和卡之间的协议。若阅读器从卡接收到一个以上的 NULL 字节,这种情况就可以避免。SW1 通知阅读器请求的指令结果。SW1 允许值是作为应用层协议的一部分定义的。一些指令需要卡发送数据给阅读器。在这种情况下 SW2 返回到阅读器中，触发阅读器执行 GetResponse 命令。智能卡将会返回上一指令执行生成的数据字节。

5.2 应用层协议

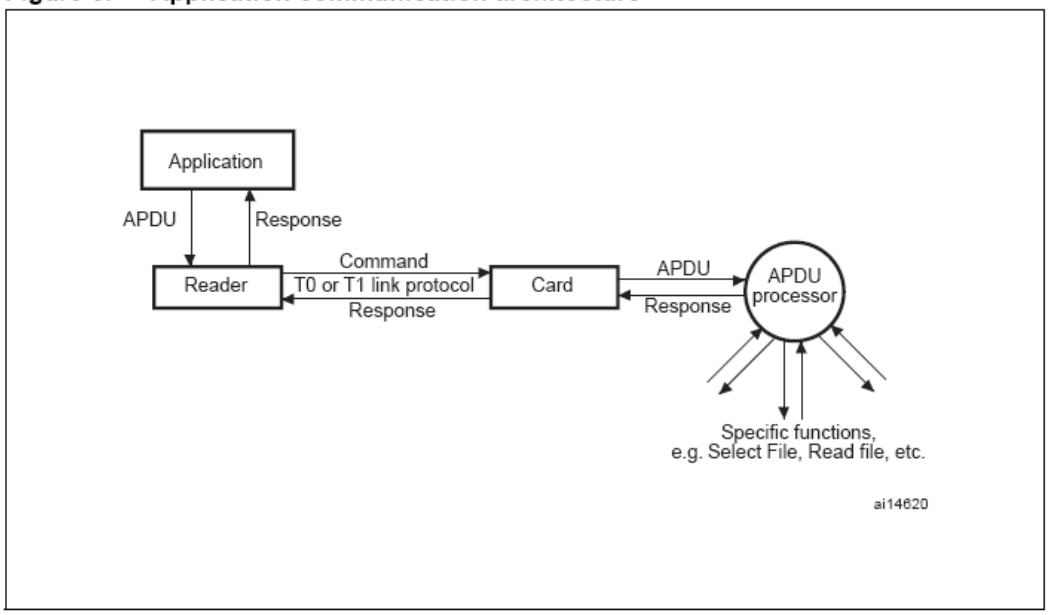
ISO 7816-4 标准对应用软件分配了两个功能域。

文件系统:以 API 形式提供的一系列函数。通过在阅读器这一端使用 API 应用软件可以访问文件系统的文件。

安全函数:这个可以限制对应用软件或卡上文件的访问。

T0 协议用来支持在智能卡应用层和阅读器应用层之间的应用层协议。这些应用层协议交换被称为协议数据单元(APDU)的数据结构。下面的图说明了这个构架:

Figure 6. Application communication architecture



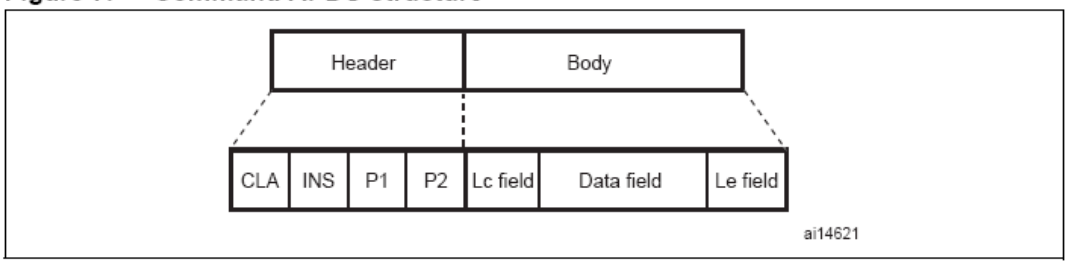
ISO 7816-4 中定义的 APDU 结构与 T0 协议中使用的 TPDU 结构是非常类似的。事实上,当一个

APDU 由 T0 协议传输时,APDU 中的元素直接覆盖了 TPDU 中的元素。

5.2.1 ISO 7816-4 APDU

有两种信息用来支持 ISO 7816-4 应用层协议:命令 APDU (从阅读器发送到卡上的)和响应 APDU(从卡发送到阅读器上的)。

Figure 7. Command APDU structure



命令 APDU 包括头和主体(这可以在上面的图中看到)。头包括 CLA,INS,P1 和 P2 域。同 T0 协议一样,CLA 和 INS 说明了应用的分类和指令。P1 和 P2 用来详细说明具体指令,并由每一条[CLA,INS]指令给出具体定义。APDU 的主体的长度可以改变,它可以作为命令的一部分传送数据到卡的 APDU 处理器上,也可以用于传达一个从卡到阅读器的响应。Lc 域说明了作为指令一部分的传送到卡上的字节数。也就是数据域的长度。数据域包括一定要传送到卡上的信息。该信息允许 APDU 处理器执行 APDU 中说明的命令。Le 域以响应 APDU 形式说明了返回到阅读器的字节数量。

APDU 的主体以四种形式存在:

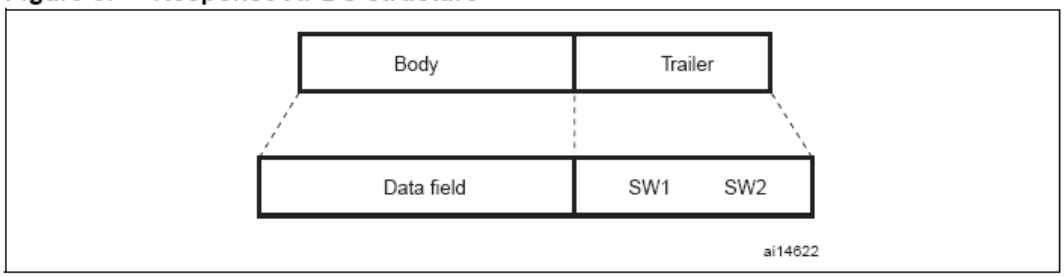
情况 1:没有数据发送或从卡上接收,所以 APDU 只包括头。

情况 2:没有数据传送到卡上,但有数据从卡上返回。APDU 的主体上只包含非空 Le 域。

情况 3:数据传送到卡上,但没有数据从卡上返回。APDU 的主体包括 Lc 和数据域。

情况 4:数据传送到卡上,同时有数据作为命令的结果从卡上返回。APDU 的主体包括 Lc,数据和 Le 域。

Figure 8. Response APDU structure



响应 APDU 的结构比命令 APDU 的结构简单的多。它包括主体和尾部。主体可以是 null 也可以包括数据域--------决定于具体命令。数据域的长度由相应的命令 APDU 的 Le 域决定。尾部包括两个状态信息域,分别为 SW1 和 SW2。这些域返回状态码,一个字节用来说明错误种类。另一个字节用来说明具体的命令状态或错误标志。

5.2.2 文件系统 API

文件系统用在非易失性存贮器或 EEPROM 上。它被定义为简单的等级结构(类似于传统的文件系统)。文件系统可以包括三种类型的文件(通过 2 个字节的标识符来标识):

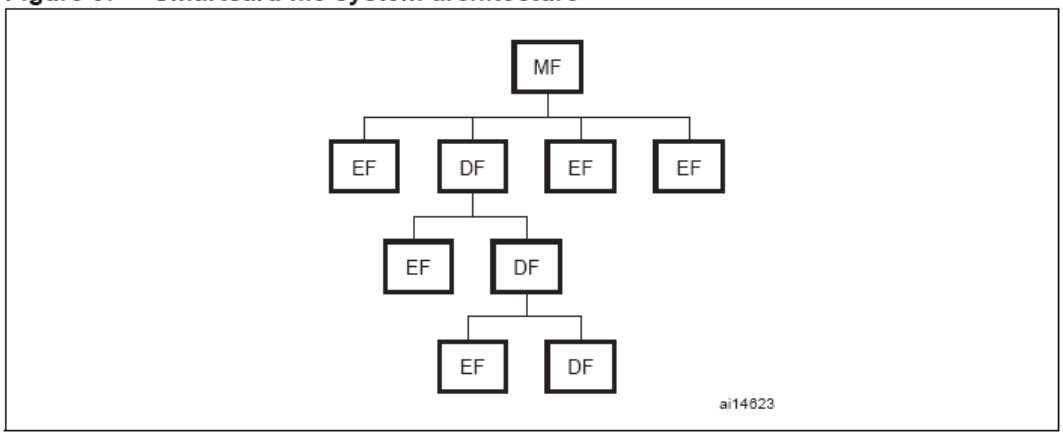
主文件(MF)

目录文件(DF)

基本文件(EF)

在每一个智能卡上都可以看见一个主文件,同时也是文件系统的根。一个主文件可以包括专有文件或基本文件。保留给主文件的标识符是 3F00。目录文件本质上是基本文件的容器(或目录),一个 DF 可以包括零个或更多个基本文件。目录文件把智能卡成为有有序结构的基本文件的集合。对于目录文件或主文件包含的目录文件一定要分配一个独特的文件标识-----允许对每一个文件分配唯一的路径。目录文件同样可以通过名字来参考(1 到 16 个字节长)。命名规则可以查看 ISO 7816-5 。基本文件是层次目录的叶子结点,并且包含实际的数据。基本文件可以在目录文件中通过 5 位的标识符来标识。。文件系统的层次结构见**表 9。**

Figure 9. Smartcard file system architecture(1)



MP=主文件,DF=目录文件,EF=基本文件。

有四种基本文件:

透明文件。

线性的,固定长度的记录文件。

线性的,可变长度的记录文件。

循环的,固定长度记录文件。

透明文件实质上是一个字符串,它是一个非结构化的二进制文件。因此在数据读出或写入这类文件时,必须在文件的开始处产生一个字节的偏移量。另外,透明文件的读写命令将包含要读出或写入文件的字符串的长度。

固定的或变长度的文件包含通过一系列数字来标识的记录。在固定长度的记录文件中,所有的记录包括同样数目的字节,相反,可变长度的记录文件包含长度可以变化的记录。结果,可变长度的记录文件的读写访问次数更多,在文件系统中需要更高层次的管理。

循环文件允许应用以固定的或透明的方式访问记录。它可以被认为是文件的环结构。写操作在环的下一个物理记录上实现。

5.2.3 ISO 7816-4 函数

下面将简单讨论一下 ISO 7816 - 4 中定义的几个用于选择，阅读，写入文件的新函数。

选择文件

这个命令建立一个指向智能卡文件系统中特殊文件的指针。这个指针可用于任何的文件处理操作。对智能卡文件系统的访问不是多线程的，然而可以在任意点同时定义几个文件的指针。这是通过管理信道命令完成的，这个命令是在阅读器这端的应用层和卡之间建立了多个逻辑信道。这允许处于不同状态的卡上文件同时被阅读器的应用层访问。文件的标识可以按下面的方式提供：

文件标识（两个字节值）

DF 名字（一个字符串）

路径（文件标识的串联）

短的 ID

注意：并不是所有的智能卡都支持这四种命名机制。

读二进制数

阅读器这端的应用层使用该命令来获得到卡上的 EF 文件的部分内容。然而 EF 一定是一个透明的文件（不是面向记录的）。如果试图向一个面向记录的 EF 发出读二进制数的命令，将会从卡上返回错误,该命令中止。

阅读二进制命令有两个参数：从文件开始到要被读到的最初字节的偏移量指针，以及将要读取的并从阅读器返回的字节数。

写入二进制数

这个命令用来在卡上的透明 EF 中插入数据。这个命令可以用来在 EF 上设置一系列的字节（也就是说将具体字节上的选择位设置为 1），清除一系列字节或执行在 EF 上的一系列写操作。

更新二进制数

在阅读器这端可以利用这个命令来直接删除和保存卡上透明 EF 邻近的字节序列。它的有效功能是像写命令那样在卡上的 EF 中写入命令中提供的一串字节。输入的参数包括从文件开始处的偏移指针和写入的字节数。

删除二进制数

删除二进制数命令用来清除在卡上的透明 EF 中的字节。

和前面的指令一样，输入参数为从 EF 开始到将要删除的字节片段的偏移量和将要删除的字节数。

阅读记录

这个命令用来读并返回在卡上 EF 中的一个或更多的记录的内容。不同于前面的指令，读记录命令有关的 EF 一定是面向记录的文件。如果应用在透明 EF 中，阅读器将返回的错误,该命令中止。

下面是从这个命令返回的内容，它决定于输入的参数：

一个具体的记录

从文件开始到特定记录间的所有记录

从特定记录到文件结尾间的所有记录

写入记录

这个命令用来把一条记录写入到面向记录的 EF 中，和写二进制命令一样，这个命令用来向 EF 中写入记录，设置或清除在 EF 中特定记录的特定位。

附加记录

附加记录命令用来添加一个记录到线性的，面向记录的 EF 中，或向一个循环，面向记录的卡上的 EF 中写第一个记录。

更新记录

这个命令用来写一个具体记录到面向记录的卡上的 EF 中，和更新二进制命令一样，老的记录被删除，新的记录写入 EF 中。

获得数据

这个命令读取并返回在卡上的文件系统中存贮的数据对象的内容。由于不同卡的数据对象是不同的，因此获得数据命令是与具体的卡相关的。

放入数据

这个命令（如名字所暗示的）把一个信息放入卡上的数据对象中。和前面的命令一样，这也是一个卡上的特有指令（不同卡上是不同的）。

5.2.4 安全 API

在智能卡上的文件系统上每一个组件都有一个相应的访问属性列表。这个访问属性保证了只有授权的部分才允许访问文件系统中的特殊组件。这个授权可以是简单的就像要求阅读器提供一个预先定义的标识数字（PIN）。然而，也可能是更加复杂的，就像需要阅读器通过加密或解密一串由卡提供的字节来证明它和卡之间共享一些秘密(例如一个密码)。

安全 API 提供的几个函数将会在下面简单介绍一下。

校验

这个命令是由阅读器端应用层发送到卡上的安全系统的。目的是使卡确信阅读器知道保存在卡上的密码，从而限制对保存在卡上的敏感信息的访问。密码的类型信息和特有的文件或文件层次结构的一些或全部相关。如果效验命令失败，如：阅读器提供一个错误的密码，将会有一个错误返回到阅读器中。

内部鉴别

这个命令允许卡通过证明和阅读器共享密钥来向阅读器证明自己。阅读器应用层软件首先生成一个随机数并且用卡和阅读器都知道的一些算法加密。这就形成了一个对卡的询问。卡然后用共享的密钥解密这个询问（存在卡中）并且发送结果数据到阅读器中。如果从阅读器中接收到的数据和它生成

的随机数匹配，那么阅读器应用层软件确保了卡的身份。

外部鉴别

这个命令和获得询问命令一起使用,来使阅读器应用层软件向卡证明自己。阅读器收到来自卡上并用密钥加密的询问数据（一个随机数）。然后用外部鉴别命令发送到卡上。卡解密这个数据并且用它和先前用获得询问指令生成的随机数比较。如果匹配，卡就确认了阅读器应用层的身份。

获得询问

这个指令是由阅读器发送到卡上的。用来向阅读器应用层提供一个由智能卡生成的随机数。和前面描述的一样，这个数是用在外部鉴别命令中的。

管理信道

信道管理命令是阅读器应用层用来打开并且关闭在它与卡之间的逻辑通信信道的。最初，卡通过完成 ATR 序列和阅读器应用层建立应用层协议,从而打开一个基本通信信道。这个信道通过信道管理命令来打开或关闭额外的逻辑信道。

封装

这个命令支持使用 T0 协议的安全信息的使用。它使 APDU 被加密并合并到封装命令的数据段(它的 APDU 的)。在卡上的 APDU 处理器然后取出并执行这个命令。

获得响应

和前面的命令一样，获得响应命令允许传送 APDU 的 T0 协议的使用。T0 协议不支持 APDU 的第 4 种类型。如：不能发送一块数据到卡上，并返回一块数据。所以在使用 T0 协议时，最初的指令导致一个响应，它指出有更多数据等着卡来发送。获得响应指令然后就用来获取数据。

6 智能卡接口库：描述

用户可以直接使用应用层访问一个智能卡。它允许发送到(或接收来自)使用下列用户接口的智能卡的 ADPU 命令：

6.1 文件组织

表 7 介绍了智能卡库模块：

表 7. 文件库描述

文件	描述
Smartcard.h , smartcard.c	- 智能卡定义，类型定义和函数原型 - T0 协议管理 - 物理层

6.2 智能卡接口库函数

表 15. 列出了智能卡库的各种函数。

表 8 . 智能卡函数

函数名	描述
SC_Handler	处理所有的智能卡状态和发送接收所有智能卡和读卡器之间通讯数据的服务。
SC_PowerCmd	使能或禁止智能卡的电源。

SC_Reset	设置或清除智能卡重置引脚。
SC_ParityErrorHandler	重新发送智能卡没有正确接收的数据位。
SC_PTSConfig	配置 IO 速度（波特率）通讯。

6.2.1 SC_Handler function

表 9 中被描述了它。

表 9. SC_Handler

函数名	SC_Handler
函数原型	void SC_Handler(SC_State *SCState, SC_ADPU_Commands *SC_ADPU, SC_ADPU_Responce *SC_Response)
行为描述	处理所有的智能卡状态和发送接收所有智能卡和读卡器之间通讯数据的服务。
输入参数 1	SCState：指向一个包含智能卡状态的 SC_State 的指针。
输入参数 2	SC_ADPU：指向一个将被初始化的 SC_ADPU_Commands 结构体的指针
输入参数 3	SC_Response：指向一个将被初始化的 SC_ADPU_Responce 结构体的指针
输出参数	无
返回值	无
必需前提	无

调用函数	无

SCState

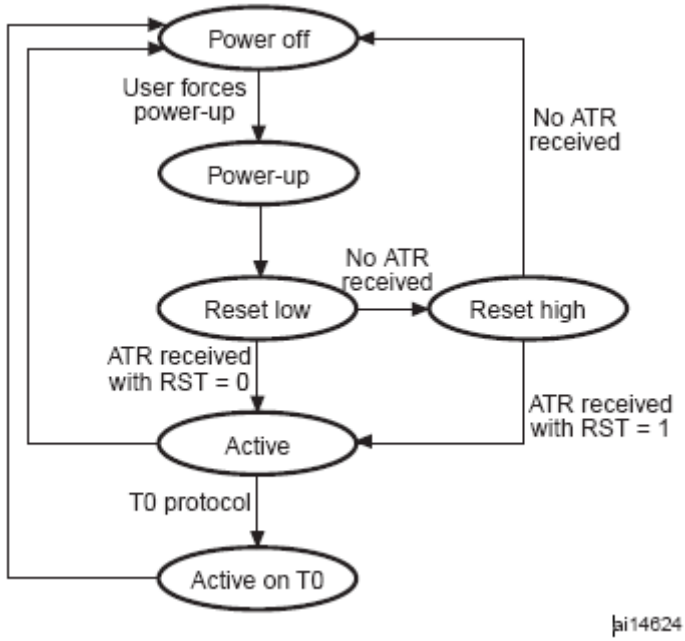
SCState 通知使用者智能卡状态,允许用户关闭智能卡。它能取在表 10 中定义的值。

表 10。 SCState

SCState	意义
SC_POWER_OFF	没有电源提供该给智能卡（Vcc=0）；STM32F10xxx 智能卡接口禁止。没有时钟提供给智能卡。
SC_POWER_ON	智能卡外围设备被使能和初始化；没有电源提供给智能卡；没有时钟提供给智能卡。
SC_RESET_LOW	在这个状态下，RST 智能卡引脚（引脚 2）被置低（RST=0）。Vcc=5V 提供给智能卡；时钟 CLK 提供给智能卡。重置应答（ATR）程序开始。读卡器等待来自智能卡的 ATR 帧。
SC_RESET_HIGH	如果没有接收到应答，读卡器强制重置引脚 RST 为高（RST = 1）并且保持它为高知道接收到重置的应答。
SC_ACTIVE	如果收到重置应答，读卡器进入激活状态并且解码 ATR 帧。它返回关于被使用协议的信息。
SC_ACTIVE_ON_T0	如果使用的协议是 T0，读卡器进入 SC_ACTIVE_ON_T0 状态，并且命令那是可

	以被发送到读卡器。

图 10. 智能卡操作状态机



SC_ADPU_Commands

SC_ADPU_Commands 结构体在 smartcard.h 中被定义：

```
typedef struct
{
SC_Header Header;
SC_Body Body;
} SC_ADPU_Commands;
```

头

指明 ADPU 命令头。它是 SC_Header 类型，被定义在 smartcard.h：

```
typedef struct
{
u8 CLA; /* Command class */
u8 INS; /* Operation code */
u8 P1; /* Selection Mode */
u8 P2; /* Selection Option */
} SC_Header;
```

Header

指定命令集的类别,用于建立一个指令的集合。(Specifies the class designation of the command set to

establish a collection of instructions。)

INS

从指令集中指明特定的指令。

P1

指明被[CLA,INS]指令使用的寻址。

P2

指明被[CLA,INS]指令使用的寻址。

Body

指明 ADPU 命令体，它时候 SC_Body 类型，被定义在 smartcard.h 中：

```
typedef struct
{
u8 LC; /* Data field length */
u8 Data[LCmax]; /* Command parameters */
u8 LE; /* Expected length of data to be returned */
} SC_Body;
```

LC

指明作为[CLA,INS]指令执行的一部分,被转移到卡上的数据字节的数目。

数据

指明转移到卡中的的数据缓存的指针。

LE

指明作为[CLA,INS]指令执行的一部分,从卡上转移的数据字节的数目。

SC_Response

指明 ADPU 指令响应。它是 SC_ADPU_Response 类型，被定义在 smartcard.h 中：

```
typedef struct
{
u8 Data[LCmax]; /* Data returned from the card */
u8 SW1; /* Command Processing status */
u8 SW2; /* Command Processing qualification */
} SC_ADPU_Responce;
```

Data

指明指向包含返回卡数据的数据缓存的指针。

SW1

指明第一状态编码的字节。该字节存储错误的种类。

SW2

指明第二状态编码的字节。该字节存储具体命令状态或错误指示。

例子：

```
/* Select the Master Root MF */
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_SELECT_FILE;
SC_ADPU.Header.P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x02;

for(i = 0; i < SC_ADPU。Body。LC; i++)

{
SC_ADPU. Body. Data[i] = MasterRoot[i];
}
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = 0;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

6.2.2 SC_PowerCmd

在表 11 中定义了它。

表 11。 SC_PowerCmd

函数名	SC_Handler
函数原型	void SC_PowerCmd(FunctionalState NewState);
行为描述	使能或禁止智能卡的电源。
输入参数	NewState：智能卡电源供应的新状态。这个参数可以：使能或禁止。

输出参数	无
返回值	无
必需前提	无
调用函数	无

例子：

```
/* Power ON the card */
SC_PowerCmd(ENABLE);
```

6.2.3 SC_Reset

在表 12 中定义了它。

表 12。 SC_Reset

函数名	SC_Handler
函数原型	void SC_Reset(BitAction ResetState);
行为描述	设置或清除智能卡重置引脚。
输入参数	重置状态：这个参数指明智能卡重置引脚的状态。BitVal 必须是 BitAction enum 的值： Bit_RESET:清除端口引脚。 Bit_SET:设置端口引脚。
输出参数	无
返回值	无
必需前提	无

调用函数	无

例子：

/* Set the Smartcard reset pin */
SC_Reset(Bit_SET);

SC_ParityErrorHandler

在表 13 中定义了它。

表 13. SC_ParityErrorHandler

函数名	SC_Handler
函数原型	void SC_ParityErrorHandler(void);
行为描述	重新发送智能卡没有正确接收的数据位。
输入参数	无
输出参数	无
返回值	无
必需前提	无
调用函数	无

例子：

/* Resend the byte to the Smartcard */
SC_ParityErrorHandler();

6.2.4 SC_PTSConfig

在表 14 中描述了它。

表 14。 SC_PTSConfig

函数名	SC_Handler
函数原型	void SC_PTSConfig(void);
行为描述	配置 IO 速度（波特率）通讯。
输入参数	无
输出参数	无
返回值	无
必需前提	必须在 ATR 队列之后立即调用。
调用函数	无

例子：

```
/* Configures the baudrate according to the card TA1 value */
SC_PTSConfig();
```

6.3 怎样发送APDU命令给智能卡

关于怎样使用 SC_Handler( )函数发送 ADPU 命令给智能卡并且得到卡的回应将在下面被详细描述。使用必须根据通过智能卡说明书来更新 SC_CLA 和智能卡指令值。

6.3.1 SC_GET_A2R

```
SC_ADPU. Header. CLA = 0x00;
SC_ADPU. Header. INS = SC_GET_A2R;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x00;

while(SCState != SC_ACTIVE_ON_T0)
{
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
}
```

SCState: 它存储当前的智能卡状态。

SC_ATR_Table: 指向一个包含智能卡 ATR 帧的数组（由 SC_Handler 函数填充）

6.3.2 SELECT_FILE

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_SELECT_FILE;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x02;
for(i = 0; i < SC_ADPU. Body. LC; i++)
{
SC_ADPU. Body. Data[i] = FileName[i];
}
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = 0;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

FileName: 它包含了 16 位的文件定义。

SCState:它存储了当前的状态。

SC_Response->SW1 and SC_Response->SW2: 他们返回智能卡的回应给 SC_SELECT_FILE 命令。

6.3.3 SC_GET_RESPONSE

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_GETRESPONSE;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x00;
i = 0;
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
```

```
}
SC_ADPU. Body. LE = SC_Responce。SW2;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

SCState:存储了当前的状态。

SC_Response->SW1 and SC_Response->SW2: 它们返回智能卡的回应给 SC_GET_RESPONSE 命令。

SC_Response->Data: 他们返回智能卡的回应给 SC_GET_RESPONSE 命令。

6.3.4 SC_READ_BINARY

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_READ_BINARY;
SC_ADPU. Header. P1 = OFFSET_MSB;
SC_ADPU. Header. P2 = OFFSET_LSB;
SC_ADPU. Body. LC = 0x00;
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = LENGTH;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

SCState: 存储了当前的状态。

OFFSET_MSB:用于读取数据的偏移量的最重要的位。

OFFSET_LSB: 用于读取数据的偏移量的最不重要的位。

LENGTH: 它包含了要被读（仅对基本文件有效）的区域的大小（以字节计）。

SC_Response->Data: 返回要被读的智能卡的数据。

SC_Response->SW1 and SC_Response->SW2: 它们返回智能卡回应给 SC_READ_BINARY 的命令。

6.3.5 SC_CREATE_FILE

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_CREATE_FILE;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x10;

for(i = 0; i < SC_ADPU. Body。LC; i++)

{
SC_ADPU. Body. Data[i] = FileParameters[i];
}
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = 0;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

FileParameters:它包含了 16 位的文件参数（文件 ID，文件载入情况）。

SCState: 它存储当前的智能卡状态。

SC_Response->SW1 and SC_Response->SW2:它们返回智能卡回应给 SC_CREATE_FILE 的命令。

6.3.6 SC_UPDATE_BINARY

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_UPDATE_BINARY;
SC_ADPU. Header. P1 = OFFSET_MSB;
SC_ADPU. Header. P2 = OFFSET_LSB;
SC_ADPU. Body. LC = LENGTH;
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = 0x00;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

SCState: 它存储当前的智能卡状态。

OFFSET_MSB: 用于读取数据的偏移量的最重要的位。

OFFSET_LSB: 用于读取数据的偏移量的最不重要的位。

LENGTH: 它包含了要被读（仅对基本文件有效）的区域的大小（以字节计）。

SC_Responce->Data: 它包含了要写的智能卡数据。

SC_Responce->SW1 and SC_Responce->SW2: 它们返回智能卡回应给 SC_UPDATE_BINARY 的命令。

6.3.7 SC_VERIFY

```
SC_ADPU. Header. CLA = SC_CLA;
SC_ADPU. Header. INS = SC_VERIFY;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x08;
for(i = 0; i < SC_ADPU. Body。LC; i++)
{
SC_ADPU. Body. Data[i] = CHV1[i];
}
while(i < LCmax)
{
SC_ADPU. Body. Data[i++] = 0;
}
SC_ADPU. Body. LE = 0;
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
```

CHV1:它包含了 8 字节的 CHV1 编码。

SCState:它存储了当前智能卡的值。

SC_Responce->SW1 and SC_Responce->SW2: 它们返回智能卡回应给 SC_VERIFY 的命令。

6.4 奇偶错误管理

在 T0 协议中，通过观察每个字节的奇偶位来实现错误处理。如果实际的奇偶位与传送的数据的奇偶位不相符，那么一定产生了一个错误；若检测到一个奇偶错误,接受端将发出要求重传的信号。这是通过保持 I/O 线为低（通常 I/O 线在一个字节传输前被设置为高）来实现。当传送端探测到这个情况，它会重发没有被正确接收的位。

6.4.1 数据由卡发送到阅读器

STM32F10xxx 事能够通过拉低在停止位期间的数据线,用硬件检测出接收数据的奇偶错误，。

6.4.2 数据由阅读器发送到卡

反之亦然,若智能卡通过拉低 I/O 线来通知发生了奇偶错误，STM32F10xxx 可通过硬件来检测帧的错误。智能卡库使用 SC_ParityErrorHandler()函数来检查奇偶错误是否发生,若有错误发生还对错误进行处理。

当调用 SC_ParityErrorHandler 函数时,若检测到错误将重发字节。

微控制器发送一个字节到卡上后,智能卡捕获发送到 I/O 线上的数据。若卡上检测到奇偶错误,在停止位期间 I/O 线将被拉低。若发生了帧错误,相关的 IRQ 事件将触发 SC_ParityErrorHandler()函数来重发最后的数据。

7 智能卡接口示例

提供了与智能卡库关联的一个例子,以帮助使用者开发自己的应用程序。

这个例子提供了一个与 ISO 7816-3/4 兼容的 GSM11。11 智能卡的操作，比如文件系统的探测，引脚 1 使能/禁止，对文件的读/写操作和受保护文件中的引脚校验。

这个例子是在 STM3210B-EVAL 评估板上开发运行的 ,这块板提供所有同智能卡接口的所需的硬件。

7.1 固件包描述

STM32F10xxx 智能卡应用程序固件包包括智能卡库和在这个章节中描述的例子。

这个固件包包含下面的子文件夹：

7.1.1 FWLib 文件夹

FWLib 文件夹包含组成 STM32F10xxx 固件库核心的所有子目录和文件:

inc 子文件夹包含固件库的头文件。

src 子文件夹包含固件库的源文件。

7.1.2 Smartcard_AN 文件夹

Smartcard_AN 文件夹包含组成智能卡接口应用实例核心的所有子目录和文件:

include 子文件夹包含示例头文件。

source 子文件夹包含示例源文件。

project 子文件夹包含两个编译了示例文件的工程：

– ***EWARM***: 包含用于 EWARM 工具链的工程。

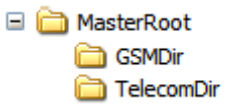
– ***RVMDK***: 包含用于 RVMDK 工具链的工程

7.2 固件描述

为 GSM 智能卡目录树定义了 3 个目录：

MasterRoot[3]={0x3F, 0x00};
GSMDir[3]={0x7F, 0x20};
TelecomDir[3]={0x7F, 0x10};

图 11。 智能卡示例：文件系统描述



在这个例子的最后：

在主根目录下的 ICCID 文件是可读的。

在 GSMDir 目录下的 IMSI 文件(通过 PIN1 拥有安全访问权限),是可读的。

引脚 1 可以被使能/禁止。

7.2.1 智能卡启动：重置应答（A2R）

阅读器访问卡的第一个动作是执行重置应答程序。为了这个目的,SC_Handler 被调用如下：

```
SC_ADPU. Header. CLA = 0x00;
SC_ADPU. Header. INS = SC_GET_A2R;
SC_ADPU. Header. P1 = 0x00;
SC_ADPU. Header. P2 = 0x00;
SC_ADPU. Body. LC = 0x00;
while(SCState != SC_ACTIVE_ON_T0)
{
SC_Handler(&SCState, &SC_ADPU, &SC_Responce);
}
```

当一个卡被检测时,将产生,接收和解码 A2R 序列。如果经过验证的协议是 T0 协议，阅读器的智

能卡状态是活跃的(SC_ACTIVE_ON_T0)并且智能卡可用于文件系统的操作。

程序类型的选择（PTS）将在 ATR 使用 SC_PTSConfig()函数后被应用。为了使用 GSM 卡，PTS 程序如下：

PTSS = 0xFF
PTS0 = 0x10
PTS1 = 0x95
PCK = 0x7A

PTS1 = 0x95, F = 9 and D = 5, Fi = 512, Di = 16, BaudRate = 112500 baud。

7.2.2 按指定路径读一个文件

指明的读取路径是被假设为：MasterRoot/GSMDir/IMSI

要到达它，要进行下面的操作：

使用文件选择 APDU 命令选择 GSMDir。

使用文件选择 APDU 命令选择 IMSI。

使用者必须获得文件的特征以检查它的访问条件。IMSI 文件有一个 CHVI（PIN1）读条件，所以在读取 PIN1 前必须检查它。这个检验命令必须在包含要读取的文件的目录下执行，GSMDir 必须在例子中被选择。

获取文件特征：

选择特性满足需要的文件。

在发布文件选择 APDU 命令后,通过发送一个 GETRESPONSE 命令来获得返回的数据。

ISMI 文件由 9 个数据位，所以要正确的运行 READ_BINARY 命令，使用下列的参数：

P1 = 0x00
P2 = 0x00
LE = 0x09

7.2.3 使能/禁止 PIN1（CHV1）编码

在应用程序的开始，PIN1（CNV1）状态是被检查的。在成功的主根选择后该操作可通过一条 GET_RESPONSE 命令实现。

在这个例子中，PIN1 状态被检查，如果它被使能，PIN1 会被禁止访问所有有安全访问要求的文件。

为了检查 PIN1 的状态，过程如下：

选择使用选择文件 APDU 命令的主根文件。

在发布主根文件选择命令后,通过发送一个 GETRESPONSE 命令来获得返回的数据。

检查 14 个接受到的字节的 bit8 位：如果 bit8 是 0：PIN1 被使能否则被禁止。

要使能或禁止 PIN1，CHV1 编码必须被发送到卡。CHV1 编码是 8 字节的长度。

7.2.4 校验 PIN1（CHV1）编码

一些文件有受限的存取条件，例如：IMSI，CHV1 文件，CHV2。这些文件的存取条件必须与用户被允许的受限的执行操作权限相符（读，更新，更改 CHV1）。

要校验 PIN1（CHV1）情况，过程如下：

去包含要被访问文件的目录。

通过提交 CHV1 编码使用校验 APDU 命令。

选择文件,执行选定操作。

8 结束语

有了支持 ISO 7816-3/4 规范的 STM32F10xxx USART 智能卡模式，使用者能够用较少的固件和硬件资源开发基于智能卡的应用程序

9 版本历史

表 15。 文档版本历史

日期	版本	更改
2007-8-3	1	初始版本

10 版权声明：

MXCHIP Corporation 拥有对该中文版文档的所有权和使用权

意法半导体（ST）拥有对英文原版文档的所有权和使用权