AN2598
应用笔记

使用 STM32F101xx 和 STM32F103xx 的智能卡接口

简介

本文叙述了如何使用STM32F10xxx的USART模块实现智能卡接口解决方案，包括固件和硬件接口的实现。该固件和硬件包的目的是提供相应的资源，使用户使用USART模块的智能卡模式，更加便捷地开发应用程序。

这个固件接口包括支持ISO 7816-3/4规范的库文件，同时提供基于ST的STM3210B-EVAL和STM3210E-EVAL评估板的应用程序示例。

本文档及其相关的固件可以从ST的网站下载：www.st.com

术语表

小容量产品是指闪存存储器容量在16K至32K字节之间的STM32F101xx、STM32F102xx和STM32F103xx微控制器。

中容量产品是指闪存存储器容量在64K至128K字节之间的STM32F101xx、STM32F102xx和STM32F103xx微控制器。在ST的STM3210B-EVAL评估板上集成的即为中容量产品。

大容量产品是指闪存存储器容量在256K至512K字节之间的STM32F101xx和STM32F103xx微控制器。在ST的STM3210E-EVAL评估板上集成的即为大容量产品。

译注：

本应用笔记配套例程下载地址：

http://www.st.com/stonline/products/support/micro/files/an2598.zip

本译文的英文版下载地址为：

http://www.st.com/stonline/products/literature/an/13750.pdf

目录

1 智能卡接口

1.1 简介

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

在使能智能卡模式的情况下，必须如下配置USART模块：

- 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发生器为智能卡提供时钟。智能卡接口的其他功能则由软件配合GPIO口实现。

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的情况。

1.2 外部接口

表1 智能卡引脚

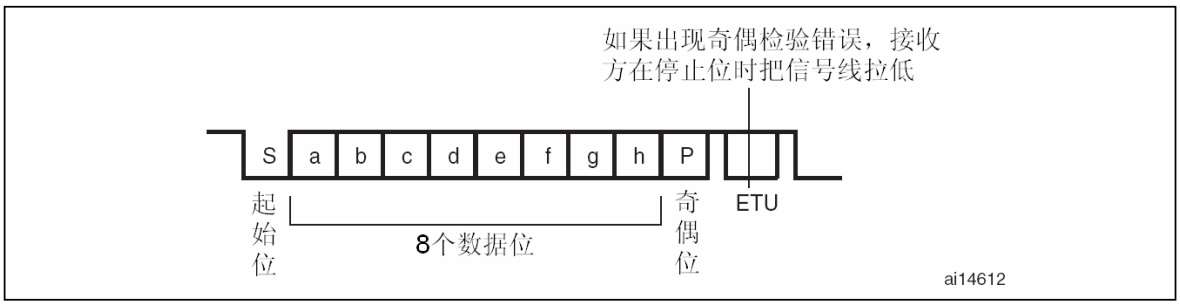
STM32F10xxx引脚	智能卡引脚	功能
USART_CK	CLK	智能卡时钟
USART_TX	IO	IO 串行数据：漏极开路输出
任意GPIO口	RST	对智能卡的复位
任意GPIO口	V_{CC}	电源
任意GPIO口	V_{PP}	编程电压

Smartcard_RST(智能卡复位)、Smartcard_3/5V(3V或5V)、Smartcard_CMDVCC (管理V_{CC})以及Smartcard_OFF信号(智能卡检测信号)由软件控制GPIO的端口实现。为了使数据信号以正确的驱动连接到智能卡IO引脚，应当把USART_TX端口的GPIO位编程为复用开漏输出模式，为把时钟发生器连接到Smartcard_CLK的引脚，USART_CK端口的GPIO位应配置为复用推挽输出模式。

1.3 协议

ISO 7816-3标准为异步协议定义了时间基准单位，称作ETU(elementary time units)，它与输入至智能卡的时钟频率有关。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信号相连接。必须将USART模块设置为智能卡模式，才能实现从STM32F10xxx向智能卡传输数据。

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与之相连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USART模块之间进行串行通信的波特率时钟。如果智能卡上有CPU，该时钟将同时提供给CPU使用。

智能卡接口操作要求，在卡上的CPU运行代码时可以调整时钟速率，这样可以改变通讯的波特率，或者可以提升智能卡的性能。在ISO7816-3标准中详细描述了，协商时钟速率和改变时钟速率的协议。

这个时钟被用作智能卡内CPU的时钟，因此更新微控制器输出的时钟频率必须和智能卡时钟同步，应注意保证没有比短周期的40%更短的脉冲。

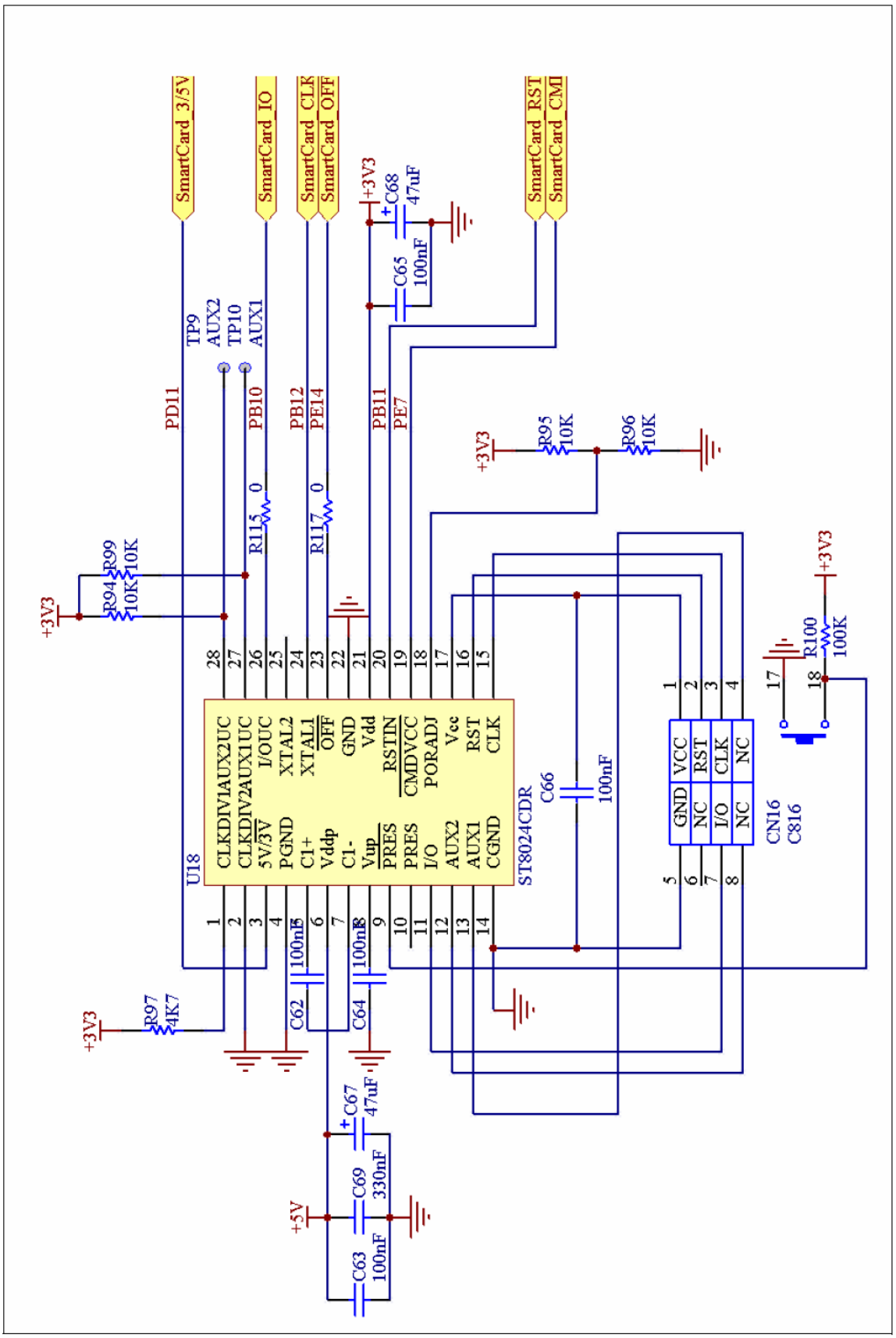
2 智能卡读卡器的硬件连接

ST8024接口芯片用于与智能卡相连。ST8024是一个针对异步3V、5V智能卡的，完善的低成本模拟接口，它位于智能卡和STM32F10xxx之间，只需要很少的外部部件来实现电源保护和控制功能。

表2 STM32F10xxx与智能卡间的连接

STM3210B-EVAL	STM3210E-EVAL	智能卡引脚	功能
USART3_CK: PB12	USART3_CK: PB12	C3: CLK	智能卡时钟：复用推挽输出
USART3_TX: PB10	USART3_TX: PB10	C7: IO	IO串行数据：复用开漏输出
PB.11	PB.11	C2: RST	至智能卡的复位：推挽输出
PE.07	PC.06	C1: V_{CC}	供电电压：推挽输出
PE.14	PC.07	OFF	智能卡检测：悬空输入
PD.11	PB.00	3/5V	3V或5V：推挽输出

图2 智能卡接口硬件连接



3 ISO7816：协议介绍

3.1 简介

“ISO 7816：识别卡——带有触点的集成电路卡”提供了把相对简单的，容易被伪造、偷盗、丢失的普通识别卡转变成一个防篡改的智能集成电路卡(Intergrated circuit card ICC)的规范，人们一般称之为智能卡。ISO 7816包括至少6个经审核的部分和一些有待审核的新增部分，如下：

- 第一部分：物理特性
- 第二部分：触点的位置和尺寸
- 第三部分：电信号和传输协议
- 第三部分：协议类型选择的更正版，第2版
- 第四部分：交换用行业间指令
- 第五部分：应用标识符的编号和登记

3.2 ISO7816-2——引脚分布

ISO 7816-2规定一个ICC，有8个电器触点，它们位于智能卡表面的标准位置，分别是C1至C8。其中一些触点与嵌在智能卡内的微处理器芯片相连；另一些触点目前没有使用，他们保留作为今后扩展。下图标示了触点的位置。

图3　智能卡触点

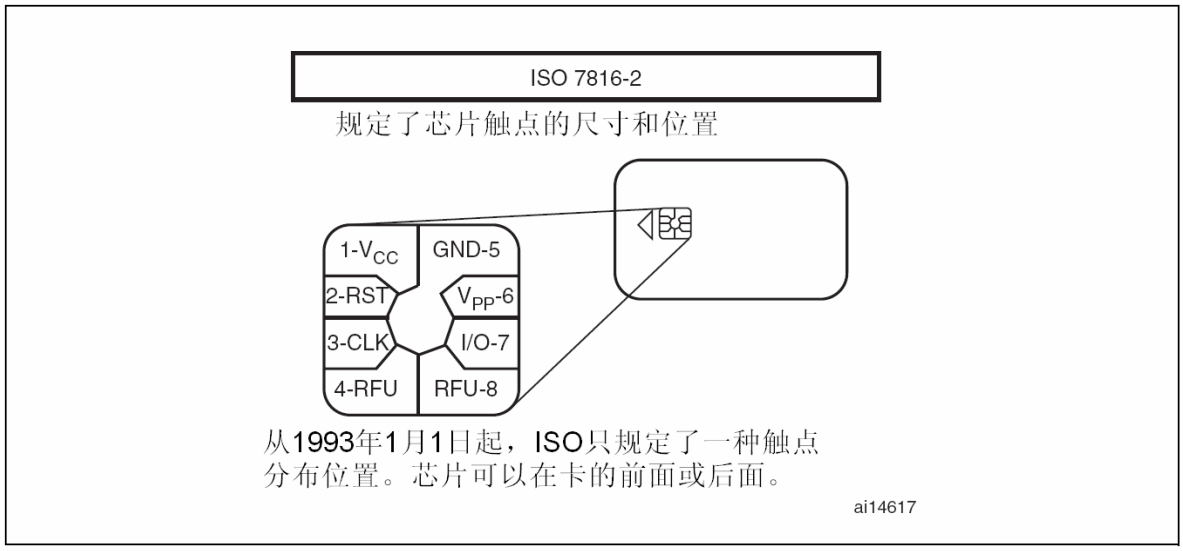


表3　引脚分配

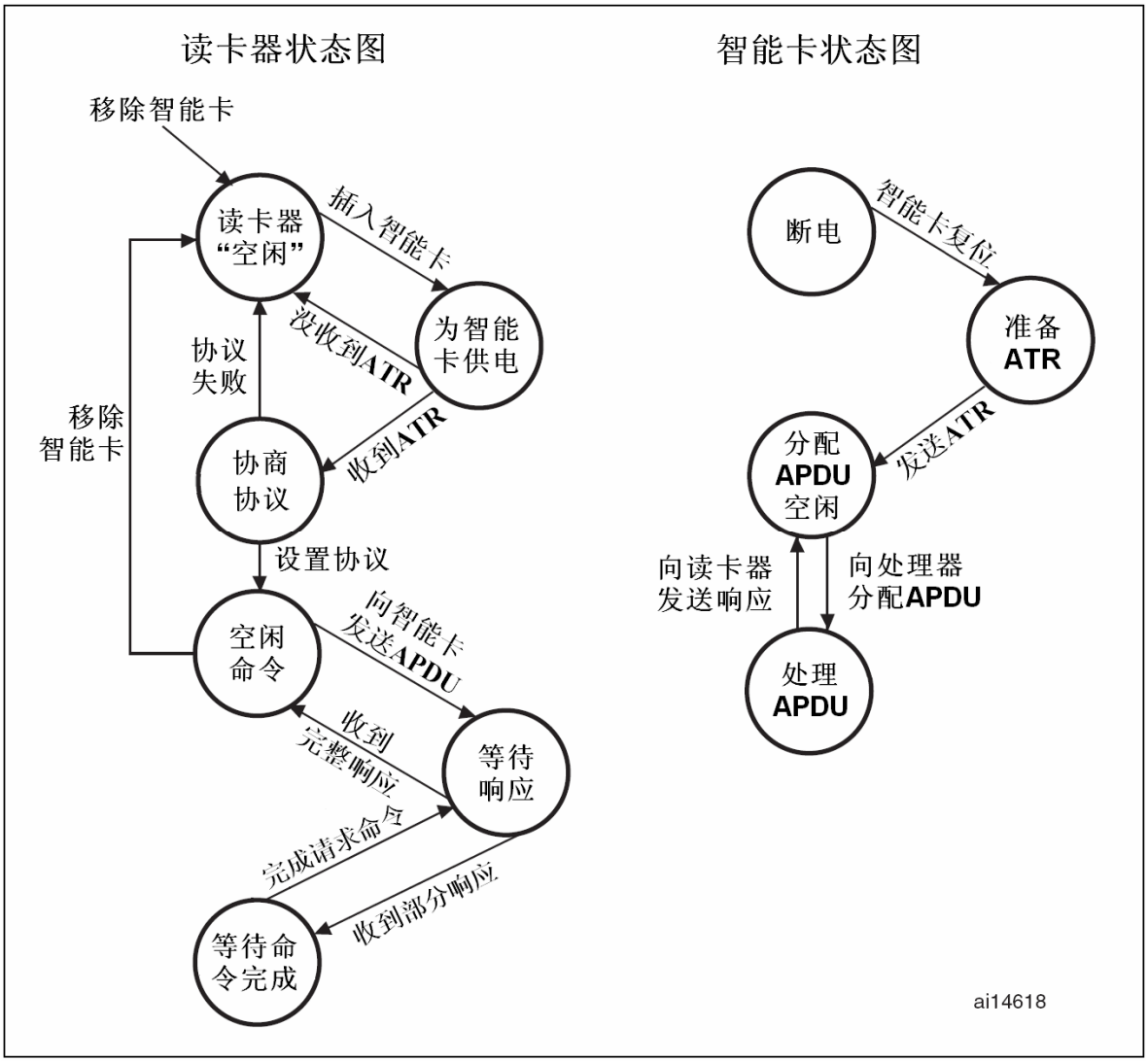
引脚	功能
C1	V_{CC} = 5V或者3.3V
C2	复位
C3	时钟
C4	保留待未来使用
C5	地
C6	编程电压
C7	输入/输出I/O
C8	保留待外来使用

4 ISO7816-3——电信号和传输协议

ISO 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读卡器之间的关系，其中智能卡作为从设备，读卡器作为主设备。通讯的基础是，读卡器通过触点给智能卡发送信号，然后智能卡作出回应，如此不断循环交互。

智能卡和读卡器的通讯过程，依照下图中的不同状态转换。

图4　读卡器和智能卡状态机



通讯通道是单线程的。一旦读卡器向智能卡发出一个命令，读卡器将等待直到收到响应回复。

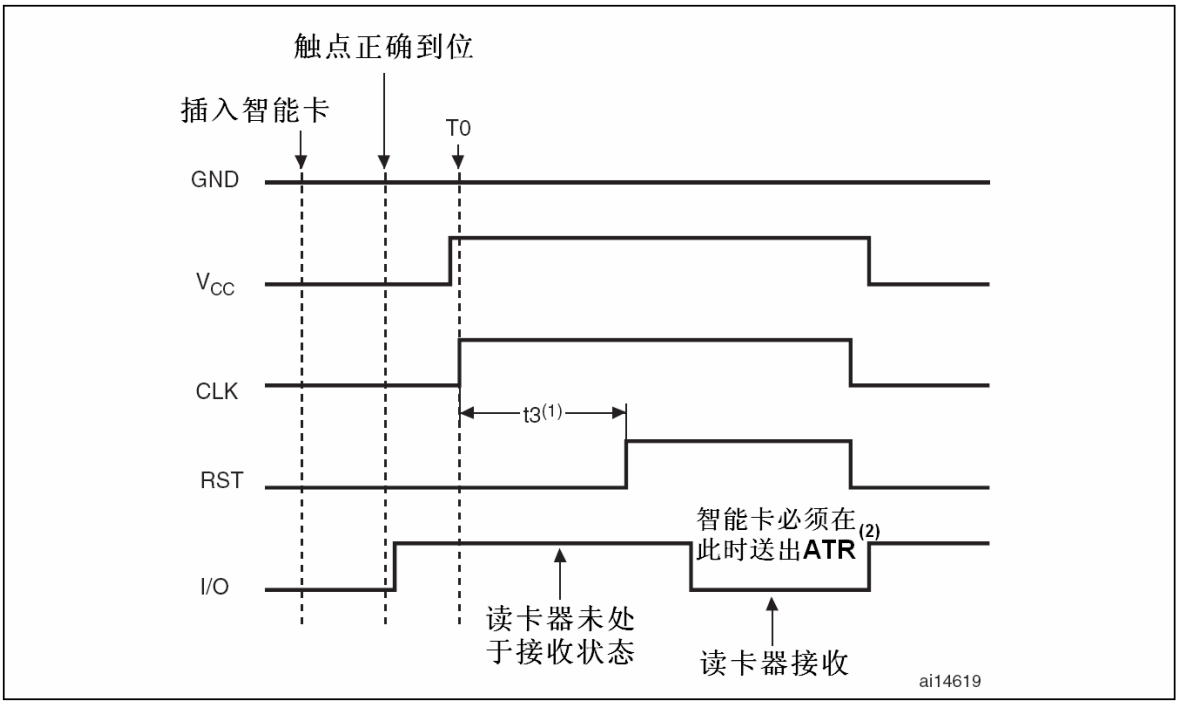
4.1 智能卡上电启动和复位

当把智能卡插入到读卡器时，不能对任何触点提供供电。因为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。触点保持无电状态，直到边沿检测器检测到智能卡的触点在(读卡器的)机械误差范围内，与各触点正确接触且位置对齐为止。

当读卡器发现智能卡已经正确地插入，就会给智能卡提供电源，智能卡首先进入的是空闲状态，如图4所示，然后读卡器通过RST线向智能卡发送一个复位信号。当电源触点(V_{CC})的5V电压进入稳定的工作状态时，智能卡进入空闲状态。尽管有些微处理器芯片的I/O状态工作在3V电压下，也总是先提供5V电压。读卡器的I/O触点设置成接收模式，同时提供稳定的时钟(CLK)。

此时复位信号线处于低电平状态。在读卡器开始有效的复位过程之前，复位线应保持至少40,000个CLK周期的低电平状态，然后再设置为高电平状态。

图5　　复位应答



(1)　t3 = 40 000个时钟周期

(2)　智能卡必须在RST变高之后的400个时钟周期至40 000个时钟周期之间送出ATR。

4.2　数据传输

在读卡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引线根据它相对于GND的电压，在每个由CLK定义的单位时间内传送一个比特的信息。可以用+5V或0V电压来表示逻辑'1'，实际操作中，这由智能卡传送给读卡器的ATR中的“初始字符” 也被称作TS所决定。在I/O引线上每传送10个比特代表一个字节的信息；最先的是“起始位”，最后一位是偶校验位。I/O信号线可以用高电平(H)也可以用低电平(L)来表示一个比特位，TS字符为HLHHLLLLLLH时，表示智能卡使用“反向约定”，即H表示'0'，L表示'1'。TS符号为HLHHLHHHLLH时，表示智能卡使用“正向约定”，即H表示'1'，L表示'0'。

反向约定和正向约定同样也控制在智能卡和读卡器之间传输的每个字节的比特顺序。在正向约定下，起始位后的第一个比特是字节的最低位，然后依次是高次序位。在反向约定下，起始位后的第一个比特是字节的最高位，然后依次是低次序位。每个传输的字节必须使用偶校验；这意味着字节中包括校验位在内，必须有偶数个'1'。

I/O信号线是一个半双工通道，这表示，智能卡或者读卡器可以在同一个通道上传输数据，但是两者不能同时传输。所以作为启动顺序的一部分，读卡器和智能卡都进入接收状态，侦听I/O信号线。在开始复位操作时，读卡器处于接收状态，而智能卡必须进入发送状态，并发送ATR至读卡器。从此之后，通道两端在发送和接收两个状态之间互相交替。在半双工通道中，没有一种可靠的方法，使得任何一方可以异步地从发送状态改变到接收状态，或者从接收状态转到发送状态。如果需要这种改变，那么要改变的一方需要进入接收状态，使得正在进行的操作过程超时；然后读卡器一方总是会尝试重新进入发送状态，来重新建立一个认可的操作序列。CLK和I/O信号线支持宽范围的不同数据传输速度。速度由智能卡定义，并且通过ATR中的可选项字符传送给读卡器。传输速度在I/O信号线上通过一个比特时间设定，这个时间决定了在I/O信号线上采样，去读取一个比特以及每个后序比特时的时间间隔。这个时间定义为一个基本时间单元(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定义产生之前已经由智能卡返回给读卡器。在传送ATR序列过程中，ETU总是被指定为ETU0=372/CLK频

率，其中CLK频率必须在1MHz至5MHz之间。实际上，总是通过设置这个频率，将数据传输的初始速度确定为9600比特/秒。

4.3 复位应答(ATR)

一旦读卡器发送RST信号至智能卡，智能卡必须在接受到RST信号后的40,000个CLK周期内回复ATR的第一个字符。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地插入读卡器(如上下颠倒)。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，读卡器就应该发送一个序列关闭智能卡，即读卡器置RST、CLK和I/O信号线为低，并且降低V_{CC}线至状态'0'(即小于0.4V)。

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

TS：必须有的一个初始化字符

T0：必须有的一个格式字符

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智能卡的类型、型号和具体应用范围。这样，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应用范围(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；即检测从智能卡发送字符到读卡器的过程中是否有传输错误。

ATR的结构在下表中详细给出。正如上面讨论的，初始化TS字符用来建立读卡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通道特性，包括智能卡和读卡器交换命令(读卡器至智能卡)和响应(智能卡至读卡器)过程中使用的协议。如果存在历史相关字符，它被用来向读卡器传递有关智能卡制造商的具体信息，然后再传送给读卡器的应用系统。目前还没有一个确定的标准，用于定义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 复位应答字符序列结构

	字符ID	描述
初始化字段	TS	必须的初始化字符
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I1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
历史相关字段	…TKi	（最多15个字符）
检测字段	TCK	可选的检测字符

5 ISO7816-4——智能卡命令

上一章讨论了复位应答机制(ATR)，它建立了智能卡与读卡器之间的基本交流通道，这是一个半双工的物理通道。这一章将介绍在这个物理层之上的更多复杂协议的使用。

连接层的通信协议处于物理通道之上，提供了在读卡器和智能卡之间的无差错通信。一旦建立起连接层的协议，就可以定义应用层的协议了。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允许智能卡和读卡器相互鉴别对方，并对智能卡和读卡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。这个信息结构由应用协议数据单元(APDUs)组成，读卡器应用层和智能卡应用层通过连接层协议交换这些数据单元。

这一章将介绍文件访问和安全API的概况。

5.1 T0 协议

T0协议是一个面向字节的协议，规定了字符在读卡器和智能卡之间传输的方式，还规定了通过奇偶效验位来处理每个字节上的错误，如果接收数据的奇偶位与发送数据的不符，表示传输发生了错误。当检验到奇偶效验出错时，T0协议要求必须重传该字节；接收方在检验到错误时，通过保持I/O线为低电平通知发送方重传(没有错误时则保持高电平)。当发送端检验到这个低电平后，它将会重新发送没有被正确接收的字节。

读卡器和智能卡数据交换结构是传输协议数据单元(TPDU)。它包括两个独立的部分：

- 从读卡器发送到智能卡上的命令。
- 从智能卡发送到读卡器上的响应。

命令头包括下面的五个域，每个域的长度是一个字节：

- CLA：指明所使用命令的命令集。
- INS：指明来自命令集内某个具体的命令。
- P1：与[CLA，INS]命令相关的参数。
- P2：与[CLA，INS]命令相关的参数。
- P3：与[CLA，INS]命令相关的参数，说明智能卡与读卡器间数据交换的字节数。

CLA用于指明所使用命令的命令集。表5列出了一些命令集对应的CLA数值。

表5 CLA命令集定义

CLA字节	命令集
0	ISO 7816-4命令(文件和安全)
10h 至 7Fh	保留值，备将来使用
8h 或者 9h	ISO 7816-4命令
Ah	应用/厂商定义的命令
B0h 至 CFh	ISO 7816-4命令
D0h 至 FEh	应用/厂商定义的命令
FFh	保留值，用作选择协议类型

CLA指定了命令所在的类，INS字节指出该类中的具体命令。表6列出了ISO 7816-4标准中用于访问文件系统和安全功能的命令。

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	管理通道	CA	获得数据
82	外部授权	D0	写二进制位
84	获得查询码	D2	写记录
88	内部授权	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定义在链路层，但却取决于应用层的具体命令。它们提供各种应用层命令的控制和地址参数。例如，在选择文件命令中，P1用于指出文件如何被访问(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具体命令执行期间传送的字节数量。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读卡器，流入是指数据从读卡器输送到卡上的。

对于从读卡器发送的每一个TDPU命令，智能卡要发送这个TPDU的响应。响应包括三个必需域和一个可选域(所有域都是一个字节)

- ACK：指出卡收到了[CLA，INS]命令。
- NULL：智能卡上I/O通道的流控制信号。智能卡(向读卡器)表示卡仍然在处理命令，因此读卡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)也是传送到读卡器的状态响应。

ACK字节是来自命令TPDU中INS字节的重复。如果响应没有在规定时间到达读卡器，读卡器将会发出一个RST序列来重启在读卡器和卡之间的协议。若读卡器从卡接收到一个以上的NULL字节，就可以避免这种情况。SW1通知读卡器请求的命令结果。在应用层的协议中，定义了SW1允许的取值。某些命令需要卡发送数据给读卡器，此时，卡通过回送SW2，要求读卡器执行GetResponse命令，然后智能卡将会返回上一命令执行生成的数据字节。

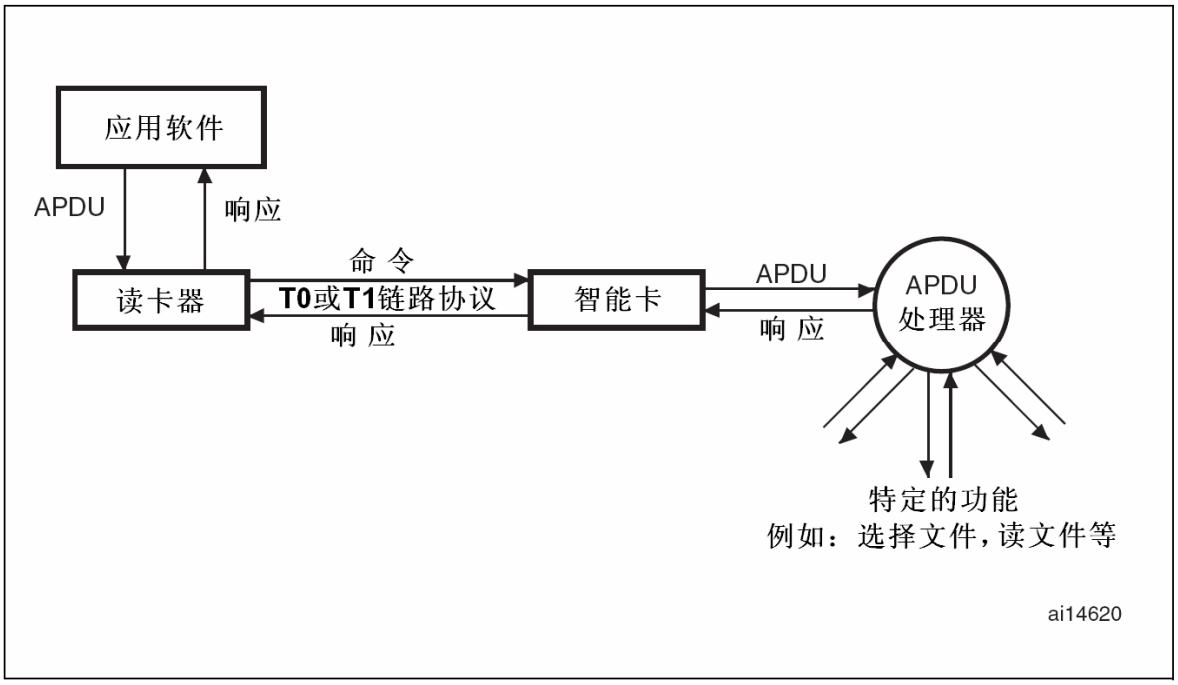
5.2　应用层协议

ISO 7816-4标准涉及应用软件的两部分功能。

- 文件系统：以API形式提供的一系列函数。通过在读卡器这一端使用API，应用软件可以访问文件系统的文件。
- 安全功能：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读卡器应用层之间的应用层协议。这些应用层协议间交换的数据结构被称为协议数据单元(APDU)。下面的图说明了这个构架:

图6　　应用层通信架构

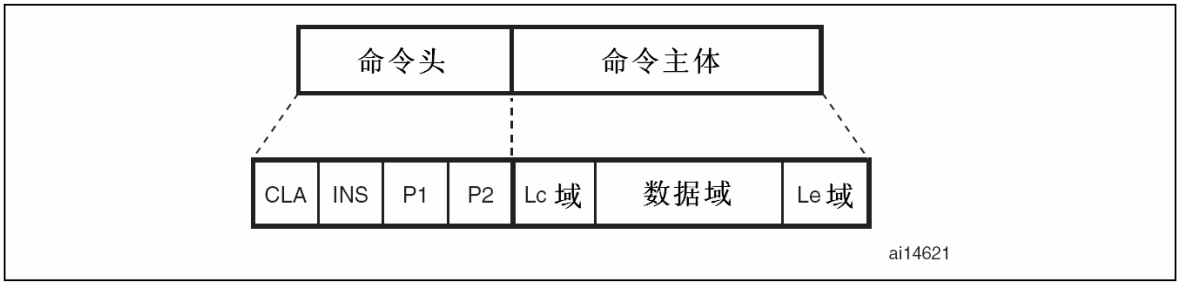


ISO 7816-4中定义的APDU结构与T0协议中使用的TPDU结构非常类似。事实上，当T0协议传输一个APDU时，APDU中的分量直接来自于TPDU中的分量。

5.2.1 ISO 7816-4 APDU

ISO 7816-4应用层协议有两种消息：命令APDU(从读卡器发送到智能卡)和响应APDU(从智能卡发送到读卡器)。

图7　　命令APDU结构

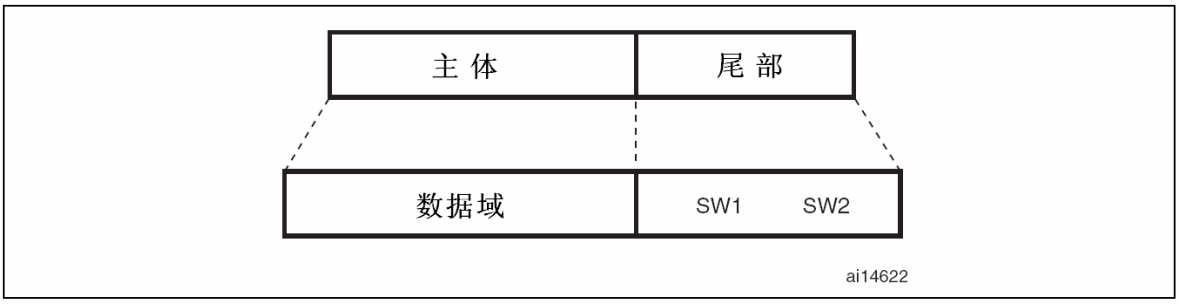


命令APDU包括头和主体(见上图)。命令头包括CLA、INS、P1和P2域。如同T0协议，CLA和INS指出了应用的分类和命令。P1和P2用来详细说明具体命令，并由每一条[CLA，INS]命令给出具体定义。APDU主体的长度是可变的，它可以作为命令的一部分传送数据到卡的APDU处理器上，也可以用于传达一个从卡到读卡器的响应。Lc域指出了作为命令的一部分，传送到卡上的字节数，即数据域的长度。数据域包括必须要传送到卡上的信息，它们是APDU处理器执行APDU的命令所需要的数据。Le域说明了在响应APDU中，需要返回到读卡器的字节数量。

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只包括命令头。
- 情况2：没有数据传送到卡上，但有数据从卡上返回。APDU的主体只包含非空的Le域。
- 情况3：有数据传送到卡上，但没有数据从卡上返回。APDU的主体包括Lc域和数据域。
- 情况4：有数据传送到卡上，同时有数据作为命令的结果从卡上返回。APDU的主体包括Lc域、数据域和Le域。

图8　　响应APDU结构



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。主体可以为空也可以包括一个数据域——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。

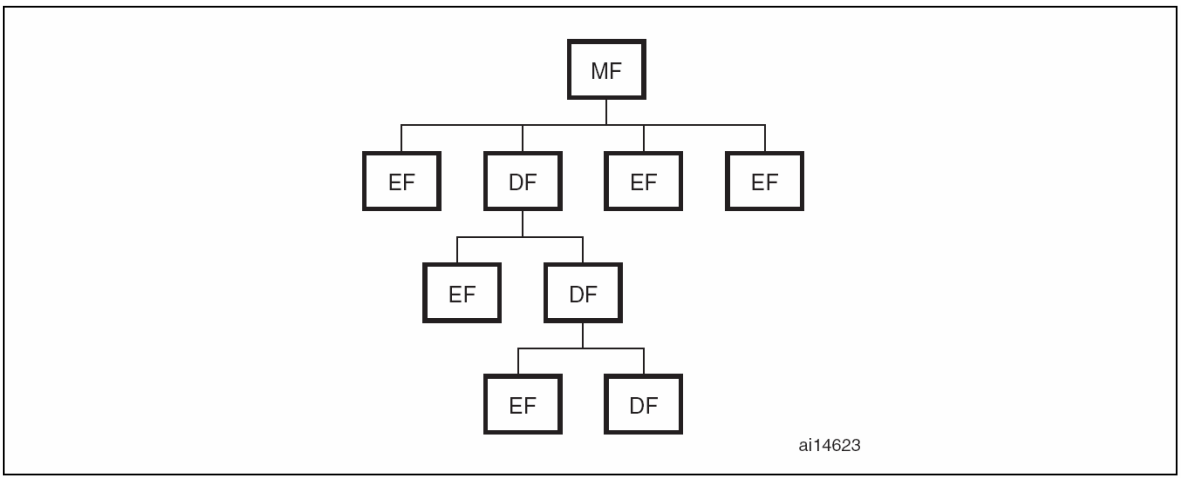
5.2.2 文件系统API

文件系统用在非易挥发性存贮器或EEPROM上。它被定义为简单的层次结构(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有一个主文件，它同时也是文件系统的根。一个主文件可以包括目录文件或基本文件。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。目录文件把智能卡分为基本文件集合的有序结构。必须要为目录文件或主文件包含的目录文件分配一个唯一的文件标识——这样每一个文件都可以有一个唯一的访问路径。目录文件同样可以通过名字来访问(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，包含实际的数据。在目录文件中通过5位的标识符来标识基本文件。文件系统的层次结构见图9

图9　　智能卡文件系统结构(1)



1. MP=主文件，DF=目录文件，EF=基本文件。

基本文件有四种类型：

- 透明文件。
- 带有固定长度记录的线性文件。
- 带有可变长度记录的线性文件。

- 带有固定长度记录的循环文件。

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，要求提供一个相对文件起始的字节偏移量指针。另外，透明文件的读写命令需要包含读出或写入文件的字符串的长度。

固定的或可变长度的文件包含一些由序号数字标识的记录。在固定长度记录的文件中，所有的记录包括相同数目的字节；相反，可变长度记录的文件包含长度可以变化的记录。因此，对可变长度记录文件的读写访问需要更多的时间，在文件系统的管理中也需要更多时间。

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、读取、写入文件的函数。

选择文件

这个命令建立一个指向智能卡文件系统中指定文件的指针。任一文件处理操作都需要这个指针。对智能卡文件系统的访问不是多线程的，但可以在任意时刻同时定义几个文件的指针。这是通过管理通道命令完成的，这个命令在读卡器端的应用层和卡之间建立了多个逻辑通道。这允许处于不同状态的卡上文件同时被读卡器的应用层访问。文件的标识可以有以下几种方式：

- 文件标识符(两个字节的数值)
- DF名字(一个字符串)
- 路径(文件标识符的串联)
- 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

读卡器端的应用层使用该命令读取卡上带有透明结构的EF(不是基于记录的EF)内容的一部分。如果试图使用读二进制命令操作一个基于记录的EF，该命令将被中止，同时智能卡返回错误。

读二进制命令有两个参数：被读数据的起始位置相对于文件开始位置的偏移量指针，以及将要读取的并返回读卡器的字节数目。

写二进制

这个命令用来向卡上的透明EF中写入数据。这个命令的执行可以是对卡内 已存在的位进行或运算或者进行与运算或者将位一次写入EF内。。

更新二进制

在读卡器端可以使用这个命令更新透明EF内已存在的数据。它的实际功能如同一个写命令，在卡上的EF中写入命令中所包含的字节。输入的参数包括相对文件开始位置的偏移指针和写入的字节数目。

删除二进制

删除二进制数命令用于清除卡上透明EF中的字节。和前面的命令一样，输入参数为从EF开始位置至将要删除的字节片段的偏移量和将要删除的字节数目。

读记录

这个命令用来读取并返回在卡上EF中的一个或多个记录的内容。不同于前面的命令，使用读记录命令操作的EF必须是基于记录的文件。如果用这个命令操作透明EF，则该命令将被中止，同时智能卡返回错误。

依据不同的输入参数，这个命令可能返回以下内容：

- 一个指定的记录
- 从文件开始位置至指定记录间的所有记录
- 从指定记录的位置至到文件结尾间的所有记录

写记录

这个命令用来把一条记录写入到基于记录的EF中。类似于写二进制命令，这个命令向EF中写入记录，设置或清除在EF中指定记录中指定的位。

添加记录

添加记录命令用于向基于记录的线性EF结尾增加一个记录，或基于记录的循环EF内写入第一个记录。

更新记录

这个命令用于写一个记录到基于记录的卡上的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定义不同，因此获得数据命令是与具体的卡相关联。

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，这也是一个与具体的卡相关联的命令。

5.2.4 安全API

在智能卡上的文件系统中，每一个部件都有其对应的访问属性列表。这个访问属性保证了只有经授权的主体(程序或人员)才能够访问文件系统中的指定部分。这个授权可以是简单的，例如要求读卡器提供一个预先定义的个人标识号码(PIN)；或者也可以是更加复杂的，例如需要读卡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)。

下面简单介绍一下安全API提供的几个函数。

验证

这个命令是由读卡器端应用层发送到卡上的安全系统，目的是使卡确信读卡器知道保存在卡上的密码，该密码的作用是限制访问保存在卡上的敏感信息。这种密码式的信息，与特定的文件或文件层次结构的一些或全部相关联。如果验证命令失败，即读卡器提供了一个错误的密码，将会向读卡器返回一个错误。

内部鉴别

这个命令允许卡通过证明和读卡器共享密钥来向读卡器鉴别自己。读卡器应用层软件首先生成一个随机数并且用卡和读卡器都知道的一些算法加密。这就形成了一个对卡的查证，卡然后用共享的(存在卡中)密钥解密这个查证数据，并且把解密结果数据发送到读卡器。如果从读卡器中接收到的数据和它生成的随机数匹配，那么读卡器应用层软件确认了卡的身份。

外部鉴别

这个命令和获得询问命令一起使用，让读卡器应用层软件向卡鉴别自己。读卡器收到来自卡上的询问数据(一个随机数)，用密钥对其加密，然后用外部鉴别命令发送到卡上。卡解密这个数据并用它和先前用获得询问命令生成的随机数比较；如果比较匹配，卡就确认了读卡器应用层的身份。

获得询问

这个命令是由读卡器发送到卡上的，用来向读卡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，这个数是用于外部鉴别命令中。

管理通道

管理通道命令是读卡器应用层用来打开和关闭在它与卡之间的逻辑通信通道的。最初，卡通过完成ATR序列和读卡器应用层建立应用层协议，从而打开一个基本通信通道。管理通道命令可以利用这个通道来打开或关闭额外的逻辑通道。

封装

这个命令支持使用T0协议的安全信息。该命令把某个需要加密的命令APDU合并到封装命令APDU的数据段，随后卡上的APDU处理器可以取出并执行这个命令。

获得响应

与前面的命令一样，获得响应命令允许使用T0协议来传送APDU。T0协议不支持APDU的第4种类型，即：不能发送一组数据到卡上，然后返回一组数据。所以在使用T0协议时，较早的命令回送一个响应，指出卡将有更多数据等待发送。获得响应命令用于读取这些数据。

6 智能卡接口库：说明

用户可以使用应用层直接访问一个智能卡。使用下列用户接口可以向智能卡发送ADPU命令，或从智能卡接收ADPU命令。

6.1 文件组织

下表介绍了智能卡库模块：

表7 库文件说明

文件	说明
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

下表列出了智能卡库的各种函数：

表8 智能卡库函数

函数名	说明
SC_Handler	处理所有的智能卡状态及发送接收所有智能卡和读卡器之间的通讯数据。
SC_PowerCmd	开启或关闭智能卡的电源。
SC_Reset	设置或清除智能卡复位引脚。
SC_ParityErrorHandler	重新发送智能卡没有正确接收的字节。
SC_PTSConfig	配置通讯的IO速度(波特率)。

6.2.1 SC_Handler函数

表9 SC_Handler

函数名	说明
函数原型	`void SC_Handler(SC_State *SCState, SC_ADPU_Commands *SC_ADPU, SC_ADPU_Responce *SC_Response)`
功能描述	处理所有的智能卡状态及发送接收所有智能卡和读卡器之间通讯数据。
输入参数1	SCState：指向一个包含智能卡状态的SC_State枚举类型的指针。
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
调用前提条件	无
调用函数	无

SCState

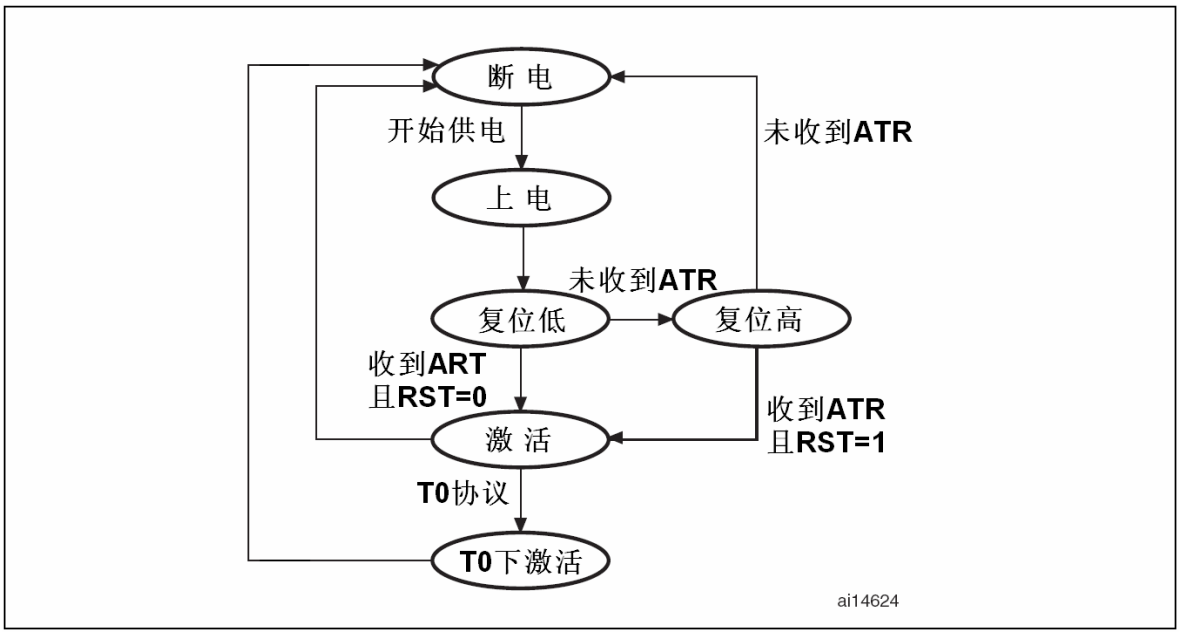
SCState通知用户智能卡的状态，允许用户关闭智能卡。它的取值在下表中定义。

表10 SCState

SCState	含义
SC_POWER_OFF	没有电源提供该给智能卡(V_{CC}=0)；STM32F10xxx智能卡接口关闭。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，智能卡RST引脚(引脚2)被置低(RST=0)。V_{CC}=5V提供给智能卡；时钟CLK提供给智能卡。开始复位响应(ATR)过程。读卡器等待来自智能卡的ATR帧。
SC_RESET_HIGH	如果没有接收到响应，读卡器强制复位引脚RST为高(RST=1)，并且保持它为高直到接收到复位响应。
SC_ACTIVE	如果收到复位响应，读卡器进入激活状态并且解码ATR帧。它返回有关使用的协议信息。
SC_ACTIVE_ON_T0	如果使用的协议是T0，读卡器进入SC_ACTIVE_ON_T0状态，该状态下可以向智能卡发送命令。

图10　智能卡操作的状态机



SC_ADPU_Commands

SC_ADPU_Commands结构体在文件smartcard.h中定义如下：

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

- 命令头

指明ADPU命令头。它是SC_Header类型，在smartcard.h中定义：

```
typedef struct
{
    u8 CLA;   /* 命令类型 */
    u8 INS;   /* 操作代码 */
    u8 P1;    /* 选择模式 */
    u8 P2;    /* 选择选项 */
} SC_Header;
```

- CLA

指定命令集的名称，用于指明命令所在的集合。

- INS

从命令集中指明特定的命令。

- P1

指明[CLA,INS]命令使用的参数。

- P2

指明[CLA,INS]命令使用的参数。

- 命令主体

指明ADPU命令主体，这是SC_Body类型，在smartcard.h中定义：

```
typedef struct
{
    u8 LC;                  /* 数据域长度 */
    u8 Data[LCmax];         /* 命令参数 */
    u8 LE;                  /* 期望返回的数据长度 */
} SC_Body;
```

- LC

指明作为[CLA,INS]命令执行的一部分，传送到卡上的数据字节数目。

- Data

包含传送到卡中的数据缓存区指针。

- LE

指明作为[CLA,INS]命令执行的一部分，从卡上返回的数据字节数目。

- SC_Response

指明ADPU命令响应。它是SC_ADPU_Response类型，在smartcard.h中定义：

```
typedef struct
{
    u8 Data[LCmax]; /* 从卡上返回的数据 */
    u8 SW1;         /* 命令处理状态 */
    u8 SW2;         /* 命令处理参数 */
} SC_ADPU_Responce;
```

- Data

指向包含从卡返回数据的数据缓存区指针。

- SW1

第一个状态代码。该字节指示错误种类。

- SW2

第二个状态代码。该字节指示具体命令状态或错误指示。

例子：

```
/* 选择主(根)文件 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

表11 SC_PowerCmd

函数名	说明
函数原型	`void SC_PowerCmd(FunctionalState NewState);`
功能描述	开启或关闭智能卡的供电。
输入参数	NewState：智能卡电源的新状态。 这个参数可以是ENABLE或DISABLE。
输出参数	无
返回值	无
调用前提条件	无
调用函数	无

例子：

```
/* 打开卡电源 */
SC_PowerCmd(ENABLE);
```

6.2.3 SC_Reset

表12 SC_Reset

函数名	说明
函数原型	`void SC_Reset(BitAction ResetState);`
功能描述	设置或清除智能卡的复位引脚。
输入参数	ResetState：这个参数给出了智能卡复位引脚的状态。 应该是BitAction枚举类型中的数值： Bit_RESET：清除端口引脚='0'。 Bit_SET：设置端口引脚='1'。
输出参数	无
返回值	无
调用前提条件	无
调用函数	无

例子：

```
/* 置智能卡复位引脚为'1' */
SC_Reset(Bit_SET);
```

6.2.4 SC_ParityErrorHandler

表13 SC_ParityErrorHandler

函数名	说明
函数原型	`void SC_ParityErrorHandler(void);`
功能描述	重新发送没有被智能卡正确地接收的字节。
输入参数	无
输出参数	无
返回值	无
调用前提条件	无
调用函数	无

例子：

```
/* 重新向智能卡发送字节 */
SC_ParityErrorHandler();
```

6.2.5 SC_PTSConfig

表14 SC_PTSConfig

函数名	说明
函数原型	void SC_PTSConfig(void);
功能描述	配置I/O通信速度(波特率)。
输入参数	无
输出参数	无
返回值	无
调用前提条件	必须在刚刚完成ATR序列后调用。
调用函数	无

例子：

```
/* 根据智能卡的 TA1 数值，配置波特率 */
SC_PTSConfig();
```

6.3 如何向智能卡发送APDU命令

下面将详细介绍关于如何使用SC_Handler( )函数向智能卡发送ADPU命令，如何得到卡发回的响应将在随后介绍。使用必须根据智能卡规格书修改SC_CLA和智能卡命令代码数值。

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：包含当前智能卡的状态
- SC_ATR_Table：指向一个包含智能卡ATR帧数组的指针，该数组由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：16位的文件标识符。
- SCState：当前的智能卡状态。
- SC_Response->SW1和SC_Response->SW2：返回给SC_SELECT_FILE命令的智能卡响应。

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：当前的智能卡状态。
- SC_Response->SW1和SC_Response->SW2：返回给SC_GET_RESPONSE命令的智能卡响应。
- SC_Response->Data：返回给SC_GET_RESPONSE命令的智能卡响应数据。

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：当前的智能卡状态。
- OFFSET_MSB：数据偏移量的高字节。
- OFFSET_LSB：数据偏移量的低字节。
- LENGTH：指示需要读取区域的字节长度(只适合于基本文件)。
- SC_Response->Data：返回需要读的智能卡数据。
- SC_Response->SW1和SC_Response->SW2：返回给SC_GET_RESPONSE命令的智能卡响应。

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：包含16字节的文件参数(文件标识符、文件访问条件等)。
- SCState：当前的智能卡状态。
- SC_Response->SW1和SC_Response->SW2：返回给SC_CREATE_FILE命令的智能卡响应。

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：当前的智能卡状态。
- OFFSET_MSB：数据偏移量的高字节。
- OFFSET_LSB：数据偏移量的低字节。
- LENGTH：指示需要读取区域的字节长度(只适合于基本文件)。
- SC_Responce->Data：包含需要写的智能卡数据。
- SC_Response->SW1和SC_Response->SW2：返回给SC_UPDATE_BINARY命令的智能卡响应。

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：包含8个字节的CHV1代码。
- SCState：当前的智能卡状态。
- SC_Response->SW1和SC_Response->SW2：返回给SC_VERIFY命令的智能卡响应。

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线在传输一个字节前被设置为高)来实现。当传送端探测到这个信号，它会重发没有被正确接收的字节。

6.4.1 从智能卡向读卡器发送数据

STM32F10xxx能够通过硬件检测出接收数据的奇偶错误，并在停止位期间拉低数据线来示意智能卡发生了错误。

6.4.2 从读卡器向智能卡发送数据

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。智能卡库中的SC_ParityErrorHandler()函数用于检查是否发生了奇偶错误，若有错误发生则对错误进行处理。

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

微控制器向智能卡发送一个字节后，智能卡捕捉到I/O线上的数据；若卡上检测到奇偶错误，它将在停止位期间拉低I/O线。若发生了帧错误，对应的IRQ事件将调用SC_ParityErrorHandler()函数来重发最后的数据。

7 智能卡接口示例

本文附带了一个使用智能卡库的例子，以帮助用户开发自己的应用程序。

这个例子实现了对与ISO 7816-3/4标准兼容的GSM11.11智能卡的简单操作，这些操作有：文件系统的遍历，引脚1使能/关闭，对文件的读/写操作和受保护文件中个人标识码的验证等。

这个例子在STM3210B-EVAL或者STM3210E-EVAL评估板上运行，这些板提供同智能卡通讯所需的所有硬件，用户可以简单地对评估板进行任意裁剪。通过去掉文件platform_config.h中相应行的注释，可以选择需要在上述哪一块板上运行例程。

7.1 固件包描述

STM32F10xxx智能卡应用程序固件包中包括智能卡库和以下各节中说明的例子。

固件包包含下面的子目录：

7.1.1 FWLib目录

FWLib目录包含STM32F10xxx固件库核心的所有子目录和文件:

- *inc子目录* 包含固件库的头文件。
- *src子目录* 包含固件库的源文件。

7.1.2 Smartcard_AN目录

Smartcard_AN目录包含智能卡接口核心的所有子目录和文件:

- *include子目录* 包含例程头文件。
- *source子目录* 包含例程源文件。
- *project子目录* 包含两个例程的工程文件：
  - EWARMv5: EWARM 第五版工具链的工程。
  - HiTOP: HiTOP 工具链的工程
  - RIDE: RIDE 工具链的工程
  - RVMDK: RVMDK 工具链的工程

7.2 固件说明

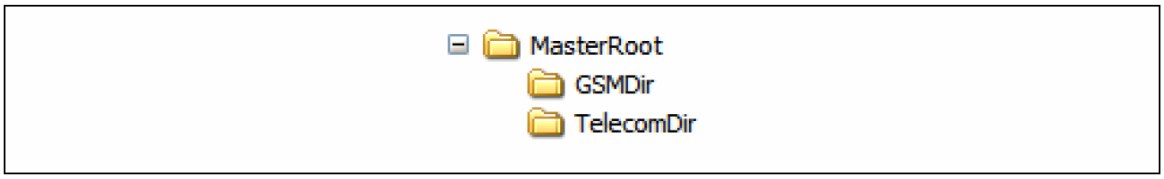
GSM智能卡内的目录树定义了3个目录：

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：

- 读取在主根目录下的ICCID文件。
- 读取在GSMDir目录下的IMSI文件，该文件的安全访问权限是PIN1。
- 开放/关闭PIN1。

7.2.1 智能卡启动：接收复位应答(A2R)

读卡器访问卡的第一个动作是执行复位并接收复位应答的过程，调用SC_Handler可以实现这个过程：

```
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);
```

当检测到一个卡时，将开始执行复位过程，并对复位应答进行接收和解码。如果识别出的协议是T0协议，则读卡器的智能卡状态是活跃的(SC_ACTIVE_ON_T0)，并且可以对智能卡进行文件系统的操作。

在接收ATR后调用SC_PTSConfig()函数，可以进行协议类型选择(PTS)。GSM卡的PTS过程如下：

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

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

7.2.2 按指定路径读一个文件

假设待读取路径是：MasterRoot/GSMDir/IMSI

要进入这个路径，需进行以下操作：

- 使用APDU的SelectFile命令选择GSMDir。
- 使用APDU的SelectFile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，所以在读取文件前必须检查PIN1。必须在包含要读取文件的目录下执行Verify命令，因此在例子中必须选择GSMDir。

获取文件特征：

- 选择需要获取其特性的文件。
- 发送APDU的SelectFile命令后，发送一个GETRESPONSE命令读取返回的数据。

ISMI文件有9个数据字节，需要使用下列的参数运行READ_BINARY命令，：

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

7.2.3 使能/关闭PIN1(CHV1)编码

在应用程序的开始需要检查PIN1(CNV1)的状态。可以在成功选择MasterRoot后，使用GET_RESPONSE命令实现该操作。

本例程中，需要检查PIN1状态，如果它被使能，那就关闭PIN1，这样即可访问所有具有安全访问要求的文件。

检查PIN1状态的流程如下：

- 使用APDU的SelectFile命令选择目录MasterRoot。

- 在发送选择MasterRoot的SelectFile命令后，发送GETRESPONSE命令读取返回的数据。
- 检查14个接收到的字节的第8位：如果它是0：PIN1被使能，否则被关闭。

发送CHV1编码至智能卡可以使能或关闭PIN1。CHV1编码的长度是8个字节。

7.2.4 验证PIN1(CHV1)编码

一些文件有受限的访问条件，例如：IMSI，CHV1文件，CHV2。只有符合访问条件的用户才被允许进行受限制的操作(读，更新，更改CHV1)。

验证PIN1(CHV1)条件的流程如下：

- 进入包含要被访问文件的目录。
- 通过提交CHV1编码来使用APDU的验证(Verify)命令。
- 选择文件，执行选定的操作。

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。