life.augmented

AN4100
应用笔记

使用 STM32F05xx 微控制器设计智能卡接口

前言

本文档介绍基于 STM32F05xx USART 外设的固件和硬件智能卡接口解决方案。此固件和硬件包的主要用途是，为使用处于智能卡模式的 USART 外设（支持 T=0 和 T=1 协议）的应用开发提供资源以加速开发过程。

固件接口包含了为支持 ISO 7816-3/4 规范而开发的库源文件。还提供了一个基于意法半导体的 STM320518-EVAL 评估板（带一些附加硬件）的应用示例。

本文档及其相关的固件可以从意法半导体网站 *http://www.st.com* 下载。

表 1 列出了本应用笔记所涉及的微控制器和开发工具。

表 1. 适用的产品和工具

类型	适用的产品
微控制器	STM32F05xx 器件
开发工具	STM320518-EVAL 板

词汇表

STM32F05xx 器件为：

- Flash 容量高达 32 KB 的 STM32F050xx 微控制器。
- Flash 容量高达 64 KB 的 STM32F051xx 微控制器。

目录

1 智能卡接口说明 6

1.1 前言 6

1.2 外部接口 6

1.3 协议 7

1.4 智能卡时钟发生器 7

2 智能卡读卡器硬件连接 8

3 ISO 7816：协议概述 10

3.1 简介 10

3.2 ISO 7816-2——引脚分布 10

4 ISO 7816-3——电子信号和传输协议 11

4.1 智能卡的上电与复位 11

4.2 数据传输 12

4.3 复位应答 (ATR) 15

5 ISO 7816-4——智能卡命令 17

5.1 T0 协议 17

5.2 T1 协议 18

5.3 应用层协议 20

5.3.1 ISO 7816-4 APDU 21

5.3.2 文件系统 API 22

5.3.3 ISO 7816-4 功能 23

5.3.4 安全 API 24

6 智能卡 (T=0) 接口库：说明 26

6.1 文件组织 26

6.2 智能卡接口库函数 26

6.2.1 SC_Handler 函数 26

6.2.2 SC_PowerCmd 29

6.2.3 SC_Reset 30

6.2.4 SC_ParityErrorHandler 30

6.2.5 SC_PTSConfig 31
6.3 如何向智能卡发送 APDU 命令 31
6.3.1 SC_GET_A2R 31
6.3.2 SELECT_FILE 32
6.3.3 SC_GET_RESPONSE 32
6.3.4 SC_READ_BINARY 33
6.3.5 SC_CREATE_FILE 33
6.3.6 SC_UPDATE_BINARY 34
6.3.7 SC_VERIFY 34
6.4 奇偶校验错误管理 35
6.4.1 从智能卡向读卡器发送数据 35
6.4.2 从读卡器向智能卡发送数据 35

7 智能卡 (T=0) 接口示例 36
7.1 固件说明 37
7.1.1 智能卡启动：复位应答 (A2R) 38
7.1.2 读取指定路径的文件 38
7.1.3 使能/禁止 PIN1 (CHV1) 代码 40
7.1.4 验证 PIN1 (CHV1) 代码 40

8 智能卡 (T=1) 接口库：说明 41
8.1 文件组织 41
8.2 智能卡接口库函数 41
8.2.1 T1_Protocol_Init 函数 42
8.2.2 T1_SetParameter 函数 42
8.2.3 T1_Negotiate_IFSD 函数 42
8.2.4 T1_APDU 函数 43

9 智能卡 (T=1) 接口示例 44
9.1 固件说明 44

10 结论 46

11 版本历史 47

表格索引

表 1. 适用的产品和工具 1
表 2. 智能卡引脚 6
表 3. STM32F05xx 和智能卡连接 8
表 4. 时钟速率转换因子 F 14
表 5. 位速率调节因子 D 14
表 6. 最大编程电流因子 I 14
表 7. 复位应答结构 16
表 8. CLA 指令集定义 17
表 9. ISO 7816-4 INS 代码 18
表 10. 块帧 19
表 11. T0 文件库说明 26
表 12. 智能卡库函数 26
表 13. SC_Handler 26
表 14. SCState 27
表 15. SC_PowerCmd 29
表 16. SC_Reset 30
表 17. SC_ParityErrorHandler 30
表 18. SC_PTSConfig 31
表 19. T1 文件库说明 41
表 20. 智能卡 T =1 库函数 41
表 21. T1_Protocol_Init 42
表 22. T1_SetParameter 42
表 23. T1_SetParameter 42
表 24. T1_APDU 43
表 25. 文档版本历史 47

图片索引

图 1. ISO 7816-3 异步协议 7
图 2. 智能卡接口硬件连接 9
图 3. 智能卡的触点定义 10
图 4. 读卡器和智能卡 FSM 11
图 5. 复位应答 12
图 6. 字符等待时间 19
图 7. 块等待时间 20
图 8. 应用程序通信架构 20
图 9. APDU 命令结构 21
图 10. APDU 响应结构 21
图 11. 智能卡文件系统架构 22
图 12. 智能卡操作的状态机 27
图 13. 智能卡软件示例流程图 37
图 14. 智能卡示例：文件系统说明 38
图 15. 获得指定路径的文件的响应（示例） 39
图 16. 智能卡软件（T1 示例）流程图 45

1 智能卡接口说明

1.1 前言

智能卡接口在 USART 智能卡模式下进行开发。有关 USART 寄存器的说明，请参见 STM32F05xx 参考手册 (RM0091)。USART 智能卡模式支持 ISO 7816-3（A 类）标准中定义的异步协议智能卡。更多详细信息，请参见此标准。

使能智能卡模式后，必须将 USART 配置为：

- 8 位数据位加上奇偶校验
- 0.5 或 1.5 位停止位

智能卡的时钟源为一个 5 位预分频器和智能卡时钟发生器。将 GPIO 引脚与软件配合使用来提供智能卡接口所需的其它功能。

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的情况。

智能卡有三种不同的工作电压：

- 5 V（ISO7816-3 A 类）
- 3 V（ISO7816-3 B 类）
- 1.8 V（ISO7816-3 C 类）

1.2 外部接口

表 2. 智能卡引脚

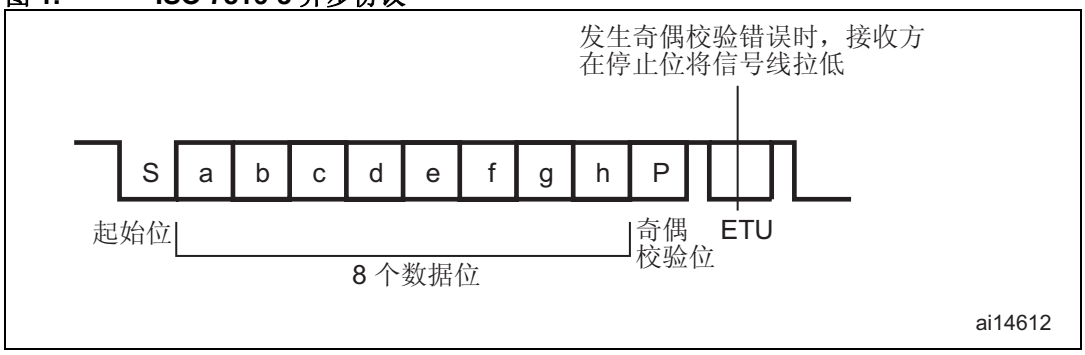
STM32F05xx 引脚	智能卡引脚	功能
USART CK	CLK	智能卡时钟
USART_TX	IO	IO 串行数据：漏极开路输出
任意 GPIO	RST	复位智能卡
任意 GPIO	V_{CC}	电源电压
任意 GPIO	V_{PP}	编程电压

Smartcard_RST（智能卡复位）、Smartcard_3/5V（3 V 或 5 V）、Smartcard_CMDVCC（用于 V_{CC} 的命令）和 Smartcard_OFF 信号（用于智能卡检测的信号）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 STM32F05xx 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连接的智能卡提供时钟信号。智能卡使用此时钟获得智能卡与另一 USART 之间的串行 I/O 的波特率时钟。智能卡中的 CPU（如果有的话）也使用该时钟。

要操作智能卡接口，需要在智能卡中的 CPU 运行代码时调节智能卡的时钟速率，以便更改波特率或提高智能卡的性能。有关控制这些时钟速率的协商以及时钟速率的变更的协议的详细说明，请参见 ISO 7816-3 标准。

该时钟用作智能卡的 CPU 时钟，因此微控制器时钟速率的更新必须与智能卡时钟同步，即，时钟的高或低脉冲的宽度不得短于旧的或新的编程值。

2 智能卡读卡器硬件连接

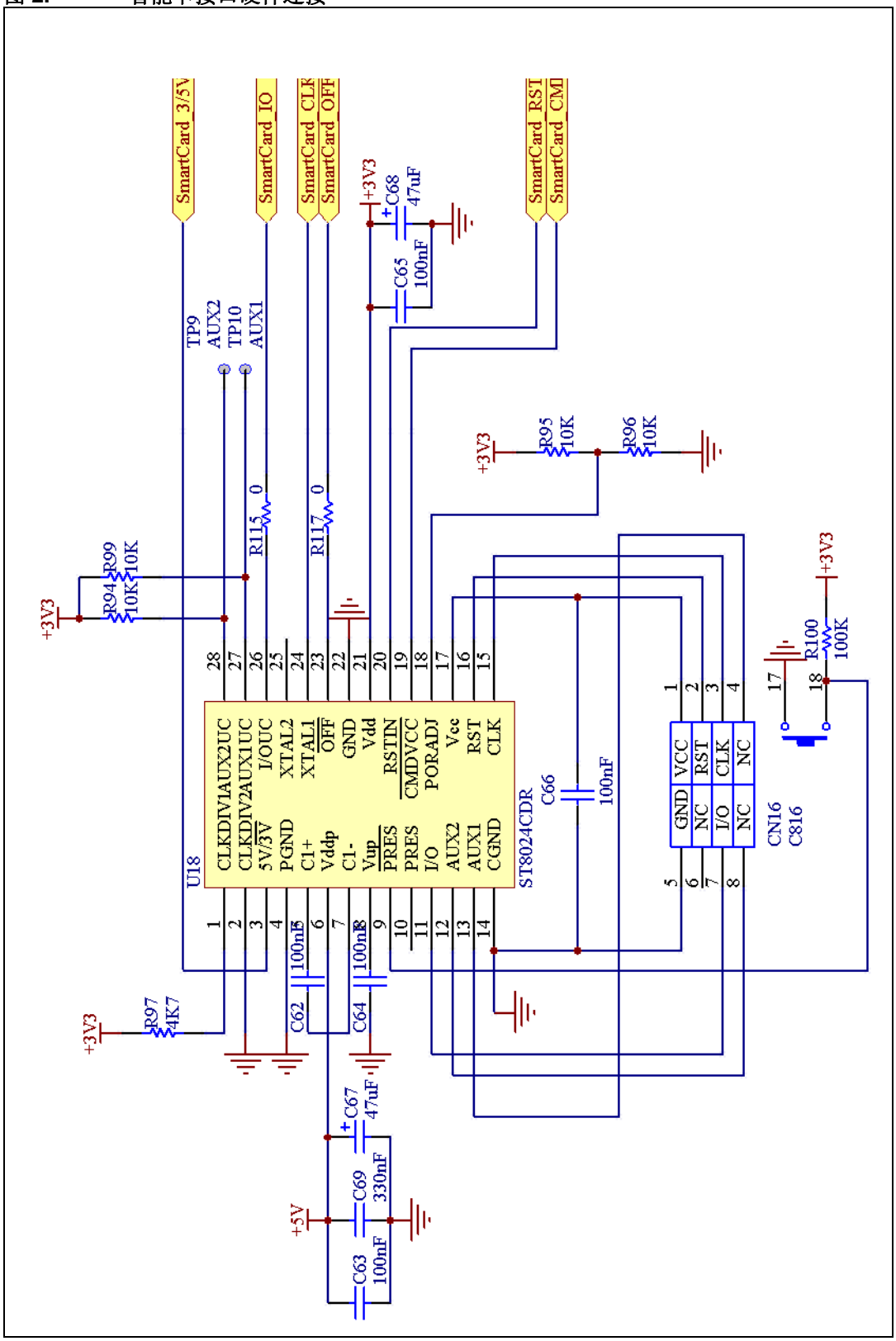
请记住，STM320518-EVAL 板没有智能卡读卡器。为运行此示例，使用了外部智能卡读卡器，如*表 3* 中所述。

ST8024 接口芯片用于与智能卡相连。ST8024 是一个针对异步 3 V、5 V 智能卡的，完善的低成本模拟接口，它位于智能卡和 STM32F05xx 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 3. STM32F05xx 和智能卡连接

外部智能卡引脚	STM320518-EVAL	功能
CLK	USART1_CK：PA08	智能卡时钟：复用推挽输出
IO	USART1_TX：PA09	IO 串行数据：复用开漏输出
RST	PB.15	复位智能卡：推挽输出
V_{CC}	PC.13	供电电压：推挽输出
OFF	PA.11	智能卡检测：输入悬空
3/5V	PA.12	3 V 或 5 V：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

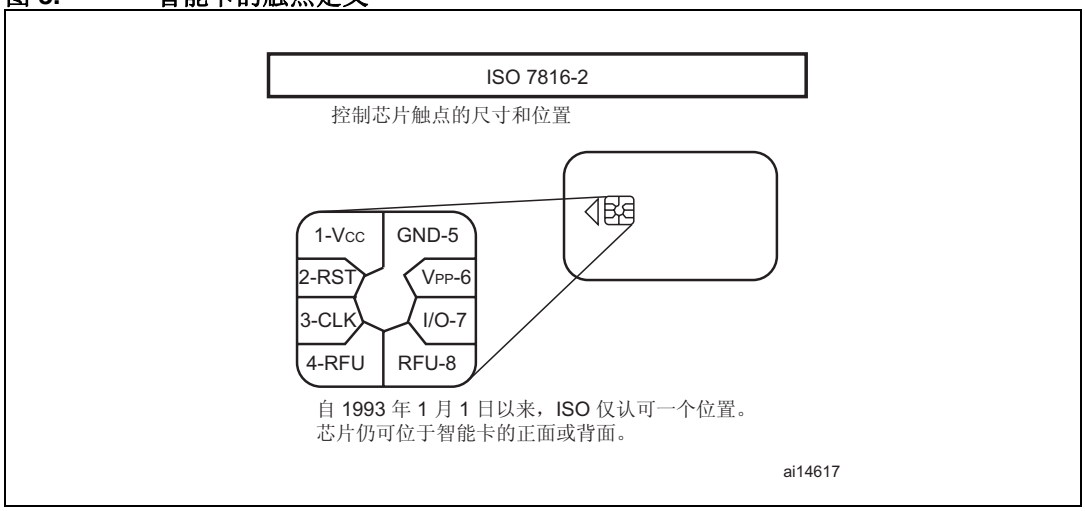
“ISO 7816：识别卡——带有触点的集成电路卡”提供了把相对简单的，容易被伪造、偷盗、丢失的普通识别卡转变成一个防篡改的智能集成电路卡（Intergrated Circuit Card，ICC）的规范，人们一般称之为智能卡。ISO 7816 包括至少 6 个经审核的部分和一些有待审核的新增部分，如下：

- 第 1 部分：物理特性
- 第 2 部分：触点的尺寸和位置
- 第 3 部分：电气接口和传输协议
- 第 3 部分：协议类型选择的修正版本 2
- 第 4 部分：交互的结构、安全和命令
- 第 5 部分：应用提供商的注册

3.2 ISO 7816-2——引脚分布

ISO 7816-2 详细说明了具有八个电气触点的 ICC，这些触点位于智能卡正面的标准化位置。其中一些触点以电气方式连接到智能卡内嵌的微处理器芯片；另一些触点目前没有使用，他们保留作为今后扩展。*图 3* 显示了触点位置。

图 3. 智能卡的触点定义

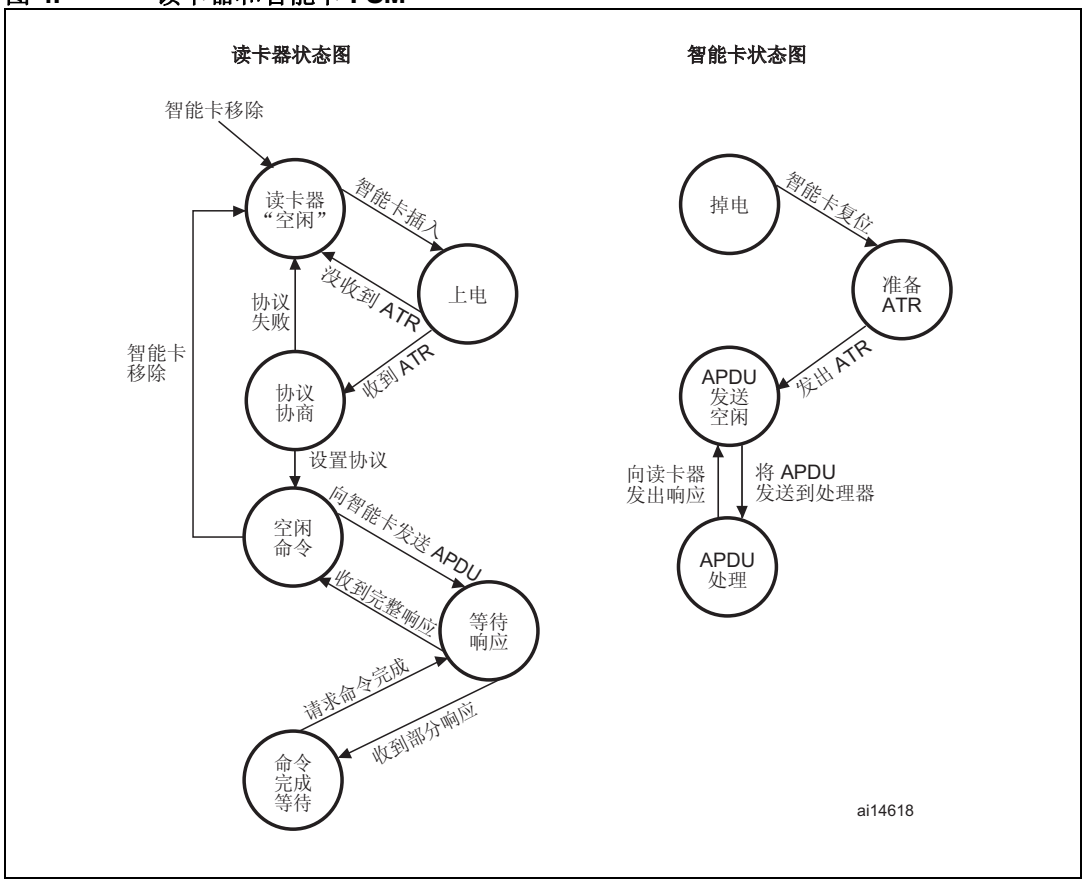


4 ISO 7816-3——电子信号和传输协议

ISO 7816-3 开始深入探讨智能卡“智能”方面的规范。此标准将智能卡与读卡器之间的关系描述为“主”（读卡器）和“从”（智能卡）关系。建立通信的方式为：读卡器通过之前所述的触点向智能卡发信号，然后智能卡相应地做出响应。

智能卡与读卡器之间的通信将根据不同的状态转换来继续，如*图 4* 所示。

图 4. 读卡器和智能卡 FSM



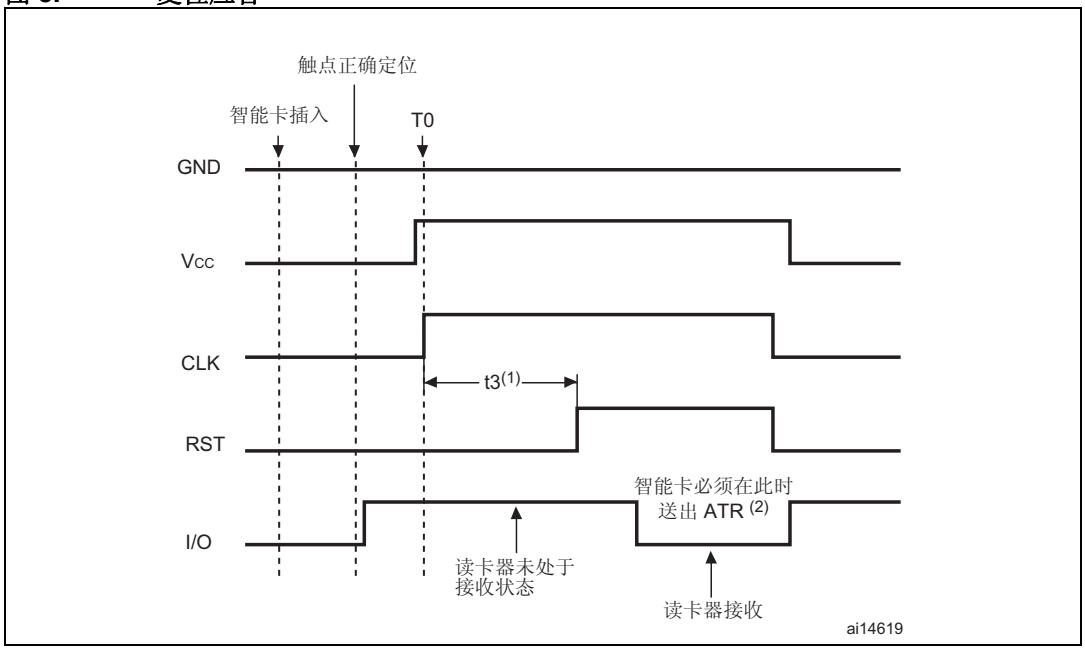
通信通道为单线程通道；读卡器向智能卡发送命令后将阻塞，直到收到响应为止。

4.1 智能卡的上电与复位

将智能卡插入读卡器时，不会对任何触点施加电源。对错误的触点施加电源可能会严重损坏智能卡上的芯片，如果在已上电的触点之间插入智能卡，很容易出现这种情况。触点将保持断电状态，直到边沿检测器确定智能卡与触点正确对齐并处于某个可接受（对于读卡器）的机械公差范围内。

当读卡器检测到已正确插入智能卡后，对卡施加电源。智能卡首先进入的是空闲状态，然后读卡器通过 RST 线向智能卡发送一个复位信号。当电源 (V_{CC}) 触点处于正常稳定的 5 V 工作电压时，智能卡将进入空闲状态。即使在 I/O 状态下引入一些在 3 V 下工作的微处理器芯片，也始终先应用 5 V 的初始电源设置。在读卡器一侧将 I/O 触点设置为接收模式，并应用提供稳定时钟 (CLK)。复位线处于低电平状态。它必须在低电平状态下保持至少 40 000 个时钟周期，读卡器才能启动有效的复位序列，从而将复位线提升到高电平状态。

图 5. 复位应答



1. t3 = 40 000 个时钟周期
2. 在 RST 变为高电平后，智能卡必须在 400 个时钟周期与 40 000 个时钟周期之间发出 ATR。

4.2 数据传输

读卡器与智能卡之间的数据传输通过两条接触线的协同操作来进行：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 +5 V 或 0 V 电压来表示逻辑的 '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'。

此外，正向约定和反向约定还控制着在智能卡与读卡器之间传输的每个字节的位序。在正向约定中，起始位后面的第一个位是字节的低序位。后面的位序依次增高。在反向约定中，起始位后面的第一个位是字节的高序位。后面的位序依次降低。传输的每个字节的奇偶校验都应为偶校验；这意味着，字节中比特为 1 的总数（包括奇偶校验位）必须为偶数。

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 信号线上的采样，去读取一个比特以及每个后序比特时的时间间隔。此时间定义为基本时间单位 (ETU) 并通过多个因素之间的线性关系进行建立。

注： *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 1 MHz 和 5 MHz 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。*

在复位应答期间使用的初始 ETU 将在后续传输期间替换为工作 ETU。F 是时钟速率转换因子，D 是位速率调节因子，用于在后续传输中确定工作 ETU。

- 对于内部时钟卡：
  - 初始 ETU = 1/9600 s
  - 工作 ETU = (1/D)*(1/9600) s
- 对于外部时钟卡：
  - 初始 ETU = 372/fi s
  - 工作 ETU = (1/D)*(F/fs) s

fs 最小值必须为 1 MHz。

fs 最大值在*表 4* 中给出。

I 和 P 定义了 VPP 上的有效状态。

- 最大编程电流：Ipp = 1 mA
- 编程电压：Vpp = P.V

N 是智能卡请求的额外保护时间。在接收下一个字符前，智能卡需要与上一个字符的前沿有至少 (12+N) 个 ETU 的延迟。没有额外的保护时间用来将字符从智能卡发送到接口器件。

这些参数的默认值为：

F = 372； D = 1； I = 50； P = 5； N = 0

整数值表示全局接口字节数。

全局接口字节 TA1、TB1、TC1、TB2 含有整数值 FI、DI、II、PI1、N 和 PI2，其中直接包含上述参数 F、D、I、P、N 的值或用于计算这些参数值。

TA1 在最高有效半字节（b8 到 b5）对 FI 编码，在最低有效半字节（b4 到 b1）对 DI 编码。

TB1 在 b7 和 b6 位对 II 编码，在 5 个最低有效位（b5 到 b1）对 PI1 编码。最高有效位 b8 的值为 0。

注： *接口器件可能忽略 TB1 的 b8 位。*

TC1 在八个位（b8 到 b1）对 N 编码。

TB2 在八个位（b8 到 b1）对 PI2 编码。

表 4. 时钟速率转换因子 F

FI	0000	0001	0010	0011	0100	0101	0110	0111
F	内部时钟	372	558	774	1116	1488	1860	RFU
最大频率	-	5	6	8	12	16	20	-

FI	1000	1001	1010	1011	1100	1101	1110	1111
F	RFU	512	768	1024	1536	2048	RFU	RFU
最大频率	-	5	7.5	10	15	20	-	-

RFU：预留给将来使用

表 5. 位速率调节因子 D

DI	0000	0001	0010	0011	0100	0101	0110	0111
D	RFU	1	2	4	8	16	RFU	RFU

DI	1000	1001	1010	1011	1100	1101	1110	1111
D	RFU	RFU	1/2	1/4	1/8	1/16	1/32	RFU

RFU：预留给将来使用

编程电压因子 P

从 5 到 25 的 PI1 提供以伏特为单位的 P 值。PI1=0 表示 VPP 已连接到智能卡，可通过 VCC 生成内部编程电压。PI1 的其它值预留给将来使用。

如果存在 PI2，则应忽略 PI1 的指示。从 50 到 250 的 PI2 提供以 0.1 V 为单位的 P 值。PI2 的其它值预留给将来使用。

表 6. 最大编程电流因子 I

II	00	01	10	11
I	25	50	100	RFU

RFU：预留给将来使用

额外保护时间 N

N 直接对额外保护时间（0 到 254 ETU）进行编码。N=255 指示两个连续字符的起始沿之间的最小延迟减少至 11 ETU。

4.3 复位应答 (ATR)

将 RST 信号从读卡器发送到智能卡后，智能卡必须在 40 000 个时钟周期内以 ATR 的第一个字符进行响应。智能卡可能会因为某些原因不回复 ATR，其原因有很多种，最常见的原因是智能卡被错误地插入到读卡器中（可能上下颠倒）。在某些情况下，智能卡可能由于损坏或折断而无法工作。无论哪种情况，如果未在规定的时间内返回 ATR，读卡器应开始一个序列使智能卡掉电。在此序列期间，读卡器将 RST、CLK 和 I/O 线设置为低电平，并将 V_{CC} 线上的电压降至标称值 0（即，小于 0.4 V）。

ATR 是在成功完成上电序列后从智能卡返回到读卡器的字符串。如 ISO/IEC 7816-3 中定义，ATR 包括 33 个或更少的字符，其中包含以下元素：

- TS——强制的初始字符
- T0——强制的格式字符
- TAi TBi TCi TDi——可选的接口字符
- T1、T2、TK——可选的历史字符
- TCK——条件校测字符

历史相关字符用来指明智能卡制造商或提供商。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）。检查字符提供了可用于测量 ATR 完整性的机制；即，检查当字符从智能卡发送到读卡器时是否出现传输错误。

ATR 的结构如*表 7* 所示。如上所述，初始 TS 字符用于在读卡器与智能卡之间建立位信令和位序约定。T0 字符用于指示后续接口字符或历史字符是否存在。接口字符用于定制 I/O 通道的特性，包括智能卡和读卡器在后续交换命令（从读卡器到智能卡）和响应（从智能卡到读卡器）期间使用的特定协议。历史字符（如果存在）用于将智能卡制造商特定的信息从智能卡传送到读卡器，并因此传送到正在使用读卡器的应用系统。对于历史位中提供的信息，实际并没有建立标准。

ATR 序列的总长度限制为 33 个字节，并且必须遵循以下格式：

表 7. 复位应答结构

	字符 ID	定义
初始字符部分	TS	强制的初始字符
格式字符部分	T0	表示是否存在接口字符的指示符
接口字符部分	TA1	全局，对 F1 和 D1 编码
	TB1	全局，对 11 和 PI1 编码
	TC1	全局，对 N 编码
	TD1	对 Y2 和 T 编码
	TA2	特定
	TB2	全局，对 PI2 编码
	TC2	特定
	TD2	对 Y3 和 T 编码
	TA3	TAi、TBi 和 TCi 为特定
	..TDi	对 Yi+1 和 T 编码
	T1	智能卡特定信息
历史字符部分	...TKi	（最多 15 个字符）
检测字符部分	TCK	可选的检测字符

5 ISO 7816-4——智能卡命令

上一章讨论了复位应答 (ATR) 机制，该机制用于在智能卡与读卡器之间建立基本通信通道。此通道是半双工物理通道。本章将研究此物理通道上的更复杂协议的使用。

链路层通信协议直接位于物理通道之上，在读卡器与智能卡之间提供无差错通信。建立此链路层协议后，可定义应用程序层协议。ISO 7816-4 定义了两个此类应用程序层协议：

- 文件系统 API，提供一组用于处理文件（例如读取、写入、选择等）的函数。
- 安全服务 API，允许智能卡和读卡器互相验证身份，并对智能卡和读卡器之间交换的数据加密。

ISO 7816-4 定义了一个协议消息结构来支持应用程序协议 API。此消息结构包含应用程序协议数据单元 (APDU)，在读卡器应用程序与智能卡应用程序之间通过链路级层议交换这些数据单元。本章将概述文件访问和安全 API。

5.1 T0 协议

T0 协议是一种面向字节的协议，用于在读卡器与智能卡之间通过通道传输字符。此外，通过查看奇偶校验位对每个字节进行错误处理。如果实际的奇偶校验位与所传输数据的奇偶校验不符，表示传输发生了错误。在 T0 协议中，接收方在检测到奇偶校验错误时会发信号指示要求重新传输字节。这通过将 I/O 线保持在低电平状态（通常在传输字节前将 I/O 线设置为高电平）来完成。当发送方检测到此情况时，会重新发送未被正确接收的字节。

读卡器与智能卡之间交换的数据结构称为传输协议数据单元 (TPDU)。它包括两种不同的结构：

- 命令（从读卡器发送到智能卡）
- 响应（从智能卡发送到读卡器）

命令头包括以下五个字段，每个字段的长度为一个字节：

- CLA：指明所有使用命令的命令集名称。
- INS：指明来自命令集内某个具体的命令。
- P1：用于指定 [CLA, INS] 指令使用的寻址方式
- P2：也用于指定 [CLA, INS] 指令使用的寻址方式
- P3：指定在 [CLA, INS] 指令执行过程中传输到或传输自智能卡的数据字节数。

CLA 的每个值都定义一个应用程序特定的指令集。下面的*表 8* 列出了一些指令集的值。

表 8. CLA 指令集定义

CLA 字节	指令集
0x	ISO 7816-4 指令（文件和安全）
10 到 7F	预留给将来使用
8x 或 9x	ISO 7816-4 指令
Ax	应用程序/供应商定义的指令
B0 到 CF	ISO 7816-4 指令
D0 到 FE	应用程序/供应商定义的指令
FF	预留供协议类型选择之用

CLA 指定了命令所在的类，INS 字节指出该类中的具体命令。*表 9* 列出了 ISO 7816-4 标准中用于访问文件系统和安全功能的指令。

表 9. ISO 7816-4 INS 代码

INS 值	命令名称	INS 值	命令名称
0E	擦除二进制	C0	获取响应
20	验证	C2	封装
70	管理通道	CA	获取数据
82	外部验证	D0	写入二进制
84	获取查询码	D2	写入记录
88	内部验证	D6	更新二进制
A4	选择文件	DA	放置数据
B0	读取二进制	DC	更新记录
B2	读取记录	E2	附加记录

P1 和 P2 参数在链路层定义，但实际上取决于应用层的具体命令。它们为各种应用程序特定的指令提供控制或寻址参数。例如，在“选择文件”指令中，P1 用于指示文件的引用方式（通过标识符、名称、路径等），P2 提供关于选择哪个文件的进一步细节。P3 定义要在 INS 指定的指令的执行期间传输的字节数。用于指示数据移动的约定以智能卡为中心；即，传出指数据从智能卡移动到读卡器，传入指数据从读卡器移动到智能卡。

对于每个从读卡器发送的 TDPU 命令，智能卡都将发送 TPDU 响应。响应包括三个必需字段和一个可选字段（长度均为一个字节）：

- ACK：指示智能卡已收到 [CLA, INS] 命令
- NULL：智能卡使用该命令对 I/O 通道进行流控制。它发信号（到读卡器）指示智能卡仍在处理命令，因此读卡器在发送另一命令前必须等待。
- SW1：当前命令的状态响应
- SW2：（可选）也将状态响应传送到读卡器

ACK 字节是 TPDU 命令的 INS 字节的重复。如果响应未在指定的时间内到达读卡器，读卡器可能会启动 RST 序列以重新启动读卡器与智能卡之间的协议。如果读卡器从智能卡收到至少一个 NULL 字节，则可以阻止此操作。SW1 将所请求指令的结果通知读卡器。允许用于 SW1 的值将定义为应用程序协议的一部分。某些指令要求智能卡将数据发送到读卡器。在这种情况下，SW2 会返回到读卡器，并触发读卡器执行 GetResponse 命令。然后，智能卡将返回执行上一命令时生成的数据字节。

5.2 T1 协议

T=1 协议是一种异步半双工块传输协议，这意味着一个块是可在智能卡与终端之间传输的最小数据单元。

此数据块可能包含为特定应用程序定义的应用程序协议数据 (APDU)。将信息以单个块的形式移动要求无差错传输，而且 T=1 协议的错误检测和修正要比 T=0 协议复杂得多。

通过 LRC（纵向冗余校验）或 CRC（循环冗余校验）管理错误检测。

块帧包括三个字段（序言字段、信息字段、尾声字段），如*表 10* 中所述。

表 10. 块帧

序言字段			信息字段	尾声字段
节点地址 NAD	协议控制字节 PCB	长度 LEN	可选 INF	错误检测 LRC 或 CRC EDC
1 字节	1 字节	1 字节	0-254 字节	1/2 字节

序言字段包含三个字节：

- NAD（节点地址）
- PCB（协议控制字节）
- LEN（数据长度）

NAD 字节使用位 3 - 位 1 标识源地址，使用位 7 - 位 5 标识目标地址。位 4 和位 8 用于 Vpp 控制（更多详细信息，请参见 ISO_IEC_7816-3）。

PCB 字节允许标识三种类型的块帧：

- 信息块（I 块）
- 接收就绪块（R 块）
- 监视块（S 块）

信息块是用于在智能卡与读卡器之间传输应用程序命令和数据的帧。当协议发送链接块序列形式的数据时，将接收就绪块用作确认。监视块用于建立控制参数以及对由于某种错误条件而产生的重新同步或中止状态进行影响。

LEN 字节指示帧的信息字段中的字节数（如果有的话）。其允许的值范围为 0x00 - 0xFE。信息字段最大为 254 字节。

信息字段用于传送应用程序命令和数据。

尾声字段包含可能是 LRC（纵向冗余校验）或 CRC（循环冗余校验）的块错误检测代码。LRC 占用 1 个字节，而 CRC 占用 2 个字节。

T=1 协议使用两个等待时间参数：

- 字符等待时间 (CWT)
- 块等待时间 (BWT)

字符等待时间（*图 6*）的定义是一个块中的两个连续字符之间的最大时间，而块等待时间（*图 7*）的定义是：智能卡收到的块的最后一个字符的前沿与智能卡发送的下一个块的第一个字符的前沿之间的最大延迟。

图 6. 字符等待时间

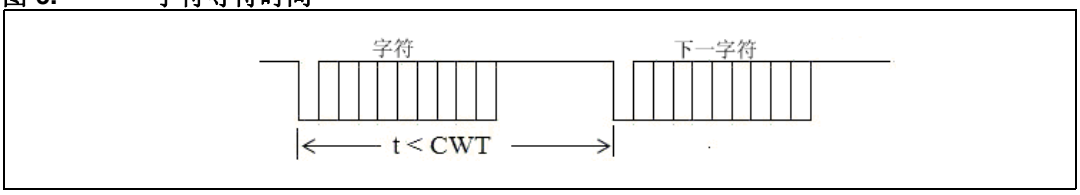
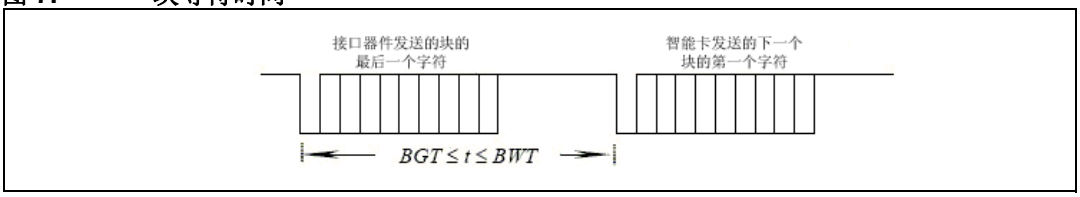


图 7. 块等待时间



还有一个块保护时间 (BGT)，它的定义是：一个块的最后一个字符的前沿与要在交替方向发送的新块的第一个字符的前沿之间的最小时间。

使用 ATR 中的 CWI 和 BWI 数据元素通过以下公式计算 CWT 和 BWT：

CWT 和 BWT 公式：

$$CWT = (11 + 2^{CWI})ETU$$

$$BWT = 11ETU + 2^{BWI} \times 960 \times \frac{372}{f}s$$

其中 f 是时钟频率。

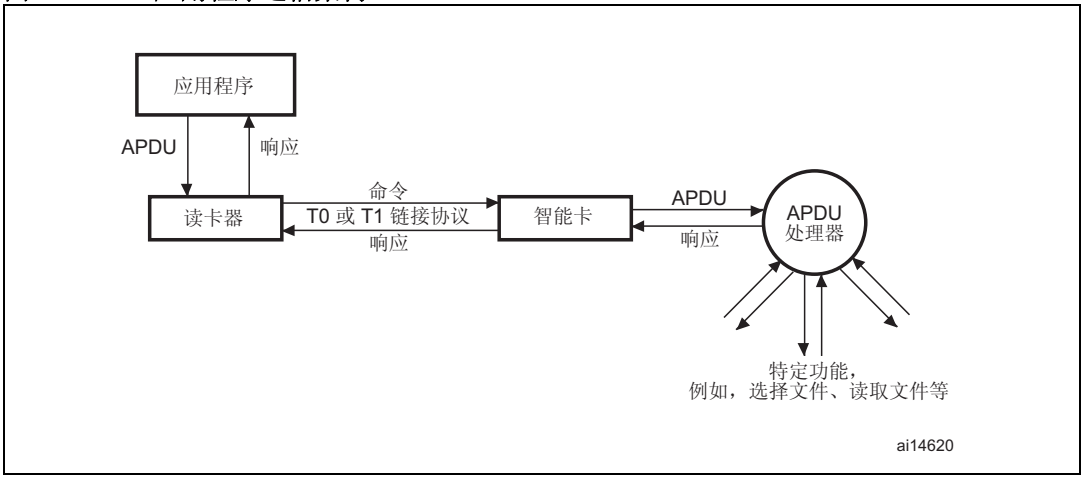
5.3 应用层协议

ISO 7816-4 标准适用于应用程序软件的两部分功能：

- 文件系统：以 API 的形式提供一组函数。通过使用此 API，读卡器一侧的应用软件可访问文件系统中的文件。
- 安全功能：这些功能可限制对应用程序软件或智能卡上的文件的访问。

T0/T1 协议用于支持智能卡应用程序与读卡器应用程序之间的应用程序层协议。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)。下图说明了此架构：

图 8. 应用程序通信架构

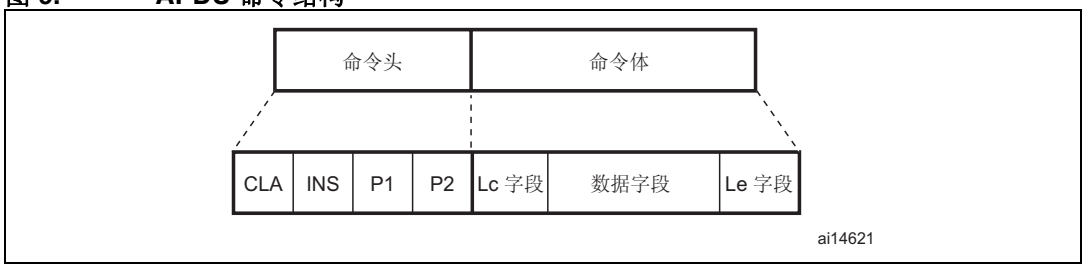


ISO 7816-4 定义的 APDU 结构与 T0 协议使用的 TPDU 结构非常相似。事实上，通过 T0 协议传输 APDU 时，APDU 的分量会直接覆盖 TPDU 的分量。

5.3.1 ISO 7816-4 APDU

有两种类型的消息可支持 ISO 7816-4 应用程序协议：APDU 命令（从读卡器发送到智能卡）和 APDU 响应（从智能卡发送到读卡器）。

图 9. APDU 命令结构



APDU 命令包含**命令头**和**命令体**（可在上面的*图 9* 中看到）。

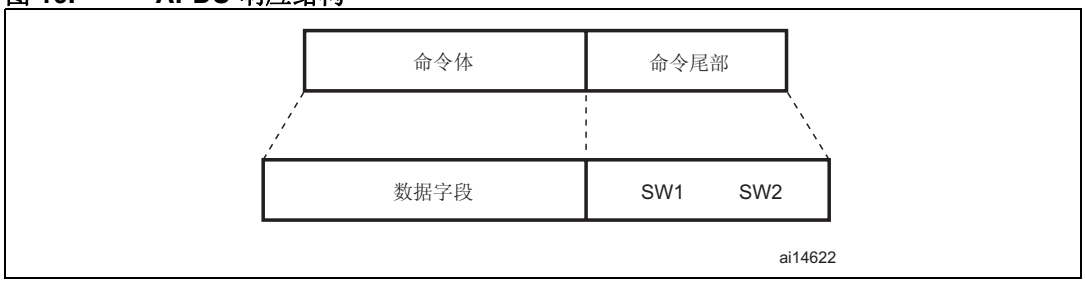
命令头包括 CLA、INS、P1 和 P2 字段。与 T0 协议相同，**CLA** 和 **INS** 指定应用程序类和指令。**P1** 和 **P2** 用来详细说明具体命令，并由每一条 [CLA, INS] 命令给出具体定义。

APDU 命令体的大小不同，用于将数据作为命令的一部分传输到智能卡的 APDU 处理器，或将响应从智能卡传送到读卡器。**Lc 字段**指定要作为指令的一部分传输到智能卡的字节数，即，数据字段的长度。**数据字段**包含必须发送到智能卡的信息，以使其 APDU 处理器执行在 APDU 中指定的命令。**Le 字段**指定 APDU 响应中将返回到读卡器的字节数。

APDU 命令体可采用四种不同的格式：

1. 没有数据传输到或传输自智能卡，因此 APDU 仅包含命令头。
2. 没有数据传输到智能卡，但从智能卡返回数据。APDU 命令体仅包含非空 Le 字段。
3. 数据传输到智能卡，但不从智能卡返回数据。APDU 命令体包括 Lc 和数据字段。
4. 数据传输到智能卡，并且从智能卡返回数据作为命令的结果。APDU 命令体包括 Lc、数据和 Le 字段。

图 10. APDU 响应结构



APDU 响应的结构比 APDU 命令的结构简单得多。它包括命令体和命令尾部。命令体为空或者包括数据字段 —— 取决于具体命令。数据字段的长度由相应 APDU 命令中的 Le 字段决定。命令尾部最多包括两个名为 SW1 和 SW2 的状态信息字段。这两个字段返回一个状态代码，其中一个字节用于指定错误类别，另一个字节用于指定命令特定的状态或错误指示。

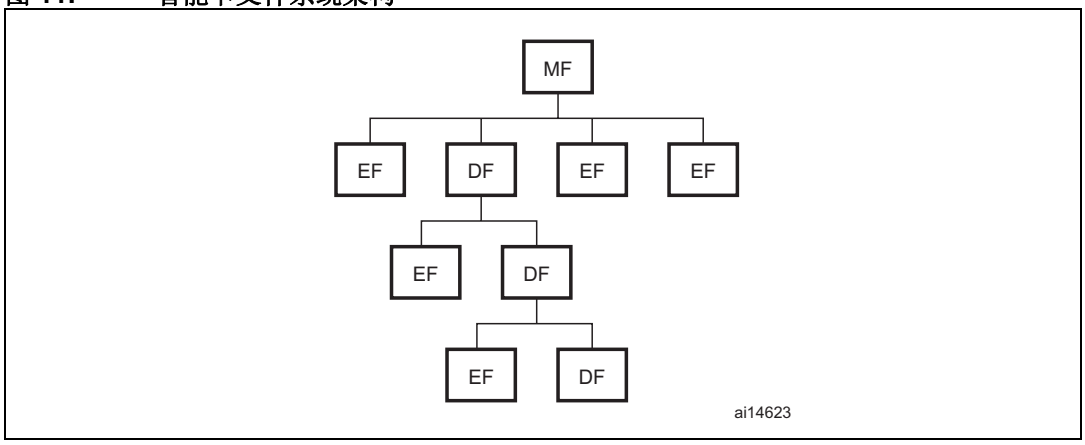
5.3.2 文件系统 API

在非易失性存储器或 EEPROM 上使用文件系统。它定义为简单的层级结构（与传统的文件系统相似）。文件系统可包含三种类型的文件（用一个双字节标识符进行标识）：

- 主文件 (MF)
- 专用文件 (DF)
- 基本文件 (EF)

每个智能卡上都有一个主文件，该文件是文件系统的根。主文件可能包含专用文件或基本文件。为主文件预留的文件标识符为 3F00。专用文件实质上是基本文件的容器（或目录）——DF 可能包含零个或更多 EF。专用文件将智能卡分割成多个具有有序结构的基本文件。必须为专用文件指定一个文件标识符，该文件标识符在专用文件以及包含该专用文件的主文件内是唯一的，从而确保每个文件的路径也是唯一的。也可以通过名称（长度为 1-16 个字节）引用专用文件。命名约定可在 ISO 7816-5 中找到。基本文件是层级中的叶节点，包含实际数据。基本文件可通过专用文件内的 5 位标识符进行标识。*图 11* 给出了文件系统层级。

图 11. 智能卡文件系统架构(1)



1. MF = 主文件，DF = 专用文件，EF = 基本文件。

有四种类型的基本文件：

- 透明文件
- 长度固定的线性记录文件
- 长度可变的线性记录文件
- 长度固定的循环记录文件

透明文件实质上是一串字节，即非结构化的二进制文件。因此，在读取数据或将数据写入到此类型的文件时，需要在文件的开始处偏移一个字节。此外，用于读取或写入透明文件的命令将包含要读取或写入文件的字节串的长度。

长度固定和可变的文件所包含的记录通过序号标识。在长度固定的记录文件中，所有记录都包含相同数量的字节。相反，长度可变的记录文件所包含的记录在长度上可能有所不同。因此，长度可变的记录文件在读/写访问时间方面的开销较高，并且文件系统所需的管理开销也较高。

循环文件允许应用程序以一致的透明方式访问记录。可将其视为文件环。对环中的下一个物理记录执行写操作。

5.3.3 ISO 7816-4 功能

下面将简要介绍 ISO 7816-4 中针对选择、读取和写入文件而定义的几个功能。

选择文件

此命令可建立一个指向智能卡文件系统中某个特定文件的逻辑指针。任何文件处理操作都需要此指针。对智能卡文件系统的访问并非多线程，但任何时候都可以定义多个文件指针。这通过“管理通道”命令来实现，该命令可在读卡器侧应用程序和智能卡之间建立多个逻辑通道。这样，卡上的不同文件可同时被读卡器的应用层访问。文件的标识可通过以下方式提供：

- 文件标识符（双字节值）
- DF 名称（字符串）
- 路径（文件标识符串联）
- 短 ID

请注意，并非所有智能卡都支持全部四种命名机制。

读取二进制

读卡器侧的应用程序使用此命令检索智能卡上 EF 的一部分。但是，EF 必须是透明文件（而不是面向记录的文件）。如果试图对面向记录的 EF 执行“读取二进制”命令，该命令将中止并显示从智能卡返回的错误指示符。

“读取二进制”命令有两个参数：从文件开始处到要读取的初始字节的偏移量指针，以及要读取并返回到读卡器的字节数。

写入二进制

此命令用于将数据插入到卡上的透明 EF 中。此命令可用于设置 EF 中的一系列字节（将指定字节内的所选位设置为值 1）、清除一系列字节或在 EF 中写入一系列字节。

更新二进制

读卡器侧应用程序可使用此命令直接在智能卡上的透明 EF 中擦除和存储一系列连续字节。它可以有效用作写命令：将命令中提供的字节串写入到智能卡上的 EF 中。输入参数包括与文件开始处的偏移量指针以及要写入的字节数。

擦除二进制

“擦除二进制”命令用于清除智能卡上透明 EF 内的字节。与先前的命令相似，输入参数包括从 EF 开始处到要擦除的字节段的偏移量，以及将要擦除的字节数。

读取记录

此命令用于读取和返回智能卡上 EF 内的一个或多个记录的内容。与上一命令不同，“读取记录”命令的 EF 必须是面向记录的文件。如果将其应用到透明 EF，命令将中止并将向读卡器返回一个错误。

可能从此命令返回以下内容，具体取决于输入参数：

- 指定的记录
- 从文件开始处到某个指定记录的所有记录
- 从某个指定记录到文件末尾处的所有记录

写入记录

此命令用于将记录写入到面向记录的 EF 中。与“写入二进制”命令一样，此命令可用于将记录写入 EF，将 EF 中某个特定记录内的特定位置 1 或清零。

附加记录

“附加记录”命令用于将记录添加到面向记录的线性 EF 末尾，或将第一个记录写入到智能卡上面向记录的循环 EF 中。

更新记录

此命令可将特定记录写入到智能卡上面向记录的 EF 中。与“更新二进制”命令一样，将擦除旧记录，并将新记录写入到 EF 中。

获取数据

此命令可读取并返回在智能卡上的文件系统内存储的数据对象的内容。“获取数据”命令特定于智能卡，因为对于不同的智能卡，数据对象的定义也不同。

放置数据

此命令（顾名思义）将信息放置到智能卡上的数据对象中。与上一命令一样，此命令特定于智能卡。

5.3.4 安全 API

智能卡上的文件系统的每个组件都具有一个相关的访问属性列表。访问属性确保只允许被授权方访问文件系统的特定组件。身份验证过程可以很简单，例如要求读卡器提供预定义的个人标识号 (PIN)。也可以更复杂，例如要求读卡器通过对智能卡提供的字节串进行加密或解密，来证明其拥有与智能卡共享的机密（例如密钥）。

下面将简要介绍安全 API 提供的几个功能。

验证

读卡器侧的应用程序将此命令发送到智能卡上的安全系统。其目的是使智能卡确信读卡器知道智能卡保存的密码，以限制对智能卡上存储的敏感信息的访问。密码型信息可能与特定文件相关联，也可能与部分或全部文件层级相关联。如果“验证”命令失败，即，读卡器提供的密码不正确，将向读卡器返回一个错误。

内部验证

此命令允许智能卡证明其拥有一个与读卡器共享的密钥，从而向读卡器验证自身。读卡器应用程序软件首先生成一个随机数字，然后使用智能卡和读卡器都知道的某个算法对其加密。这会构成一个对智能卡的质询。然后，智能卡使用密钥（存储在智能卡上）将此质询解密，并将生成的数据发送回读卡器。如果读卡器收到的数据与其生成的随机数字相匹配，则读卡器应用程序软件将确信智能卡的身份。

外部验证

此命令与“获取质询”命令配合使用，旨在使读卡器应用程序软件向智能卡验证自身。读卡器接收来自智能卡的质询数据（一个随机数字），并使用密钥对其加密。然后使用“外部验证”将其发送到智能卡。智能卡对数据解密，然后将其与上一“获取质询”命令中生成的随机数字进行比较。如果匹配，则智能卡将确信读卡器应用程序的身份。

获取质询

读卡器将此命令发送到智能卡。其目的是为读卡器应用程序提供由智能卡生成的随机数字。如上文所述，此数字用于“外部验证”命令。

管理通道

读卡器应用程序使用“管理通道”命令打开和关闭其与智能卡之间的逻辑通信通道。首先，智能卡通过完成一个 ATR 序列来建立与读卡器应用程序的应用程序层协议，从而打开基本通信通道。然后，使用此通道通过“管理通道”命令打开或关闭附加逻辑通道。

封装

此命令支持使用 T0 协议进行安全消息传送。它可以对 APDU 加密，然后将其合并到“封装”命令的数据部分（属于其 APDU）中。随后，智能卡上的 APDU 处理器可以提取和执行该命令。

获取响应

与上一命令一样，“获取响应”命令允许使用 T0 协议来传输 APDU。T0 协议无法支持 APDU 的类型 4，即，不能将数据块发送到智能卡，然后收到返回的数据块。因此，在使用 T0 协议时，初始命令产生的响应将指示更多数据正在等待智能卡发送。然后，使用“获取响应”检索此数据。

6 智能卡 (T=0) 接口库：说明

用户可直接通过应用层访问智能卡 (T=0)。它允许使用以下用户接口将 ADPU 命令发送到智能卡或从智能卡接收 ADPU 命令：

6.1 文件组织

表 11 显示了智能卡库模块：

表 11. T0 文件库说明

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

表 12 列出了智能卡库的各个函数。

表 12. 智能卡库函数

函数名称	说明
SC_Handler	处理所有智能卡状态并用于发送和接收智能卡与读卡器之间的所有通信数据。
SC_PowerCmd	接通或断开智能卡的电源。
SC_Reset	将智能卡复位引脚置 1 或清零。
SC_ParityErrorHandler	重新发送（智能卡）未能正确接收的字节。
SC_PTSConfig	配置 IO 速度（波特率）通信。

6.2.1 SC_Handler 函数

相关内容在*表 13* 中介绍。

表 13. SC_Handler

函数名称	SC_Handler
函数原型	`void SC_Handler(SC_State *SCState, SC_ADPU_Commands *SC_ADPU, SC_ADPU_Response *SC_Response)`
行为说明	处理所有智能卡状态并用于发送和接收智能卡与读卡器之间的所有通信数据。
输入参数 1	SCState：指向包含智能卡状态的 SC_State 枚举的指针。
输入参数 2	SC_ADPU：指向将要初始化的 SC_ADPU_Commands 结构的指针。
输入参数 3	SC_Response：指向将要初始化的 SC_ADPU_Response 结构的指针。

表 13. SC_Handler （续）

函数名称	SC_Handler
输出参数	无
返回参数	无
必需的先决条件	无
调用的函数	无

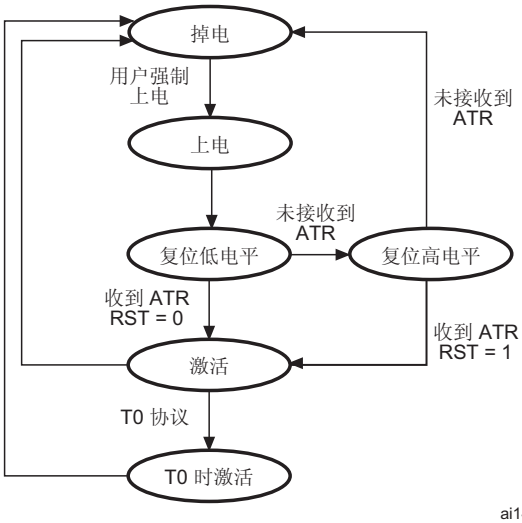
SCState

SCState 向用户通知智能卡状态，并允许用户切断智能卡的电源。它可以是下面的*表 14* 中定义的值之一。

表 14. SCState

SCState	含义
SC_POWER_OFF	未向智能卡提供电源 (V_{CC} = 0)。STM32F05xx 智能卡接口已禁止。未向智能卡提供时钟。
SC_POWER_ON	智能卡外设已使能并初始化。未向智能卡提供电源。未向智能卡提供时钟。
SC_RESET_LOW	在此状态下，RST 智能卡引脚（引脚 2）驱动为低电平 (RST = 0)。向智能卡提供 V_{CC} = 5 V。向智能卡提供时钟 CLK。复位应答 (ATR) 过程开始。读卡器等待来自智能卡的 ATR 帧。
SC_RESET_HIGH	如果没有获得应答，读卡器会将复位引脚 RST 强制设置为高电平 (RST = 1)，并使其保持在高电平，直到其获得复位应答为止。
SC_ACTIVE	如果获得复位应答，读卡器将进入活动状态并对 ATR 帧进行解码。它将返回有关所使用的协议的信息。
SC_ACTIVE_ON_T0	如果使用的协议是 T0，读卡器将进入 SC_ACTIVE_ON_T0 状态，然后可向智能卡发送命令。

图 12. 智能卡操作的状态机



SC_ADPU_Commands

smartcard.h 文件中定义了 SC_APDU_Commands 结构：

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

- **命令头**

  指定 APDU 命令头。它属于 SC_Header 类型，该类型在 *smartcard.h* 文件中定义：

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

- **CLA**

  指定用于建立指令集合的命令集的类名称。

- **INS**

  指定指令集中的特定指令。

- **P1**

  指定 [CLA, INS] 指令使用的参数。

- **P2**

  指定 [CLA, INS] 指令使用的参数。

- **主体**

  指定 APDU 命令体。它属于 SC_Body 类型，该类型在 *smartcard.h* 文件中定义：

```
typedef struct
{
  u8 LC;           /* 数据字段长度 */
  u8 Data[LCmax];  /* 命令参数 */
  u8 LE;           /* 预期的返回数据长度 */
} SC_Body;
```

- **LC**

  指定在 [CLA, INS] 指令执行过程中传输到智能卡的数据字节数。

- **Data**

  指定指向传输到智能卡的数据缓冲区的指针。

- **LE**

  指定在 [CLA, INS] 指令执行过程中传输自智能卡的数据字节数。

- **SC_Response**

  指定 APDU 命令响应。它属于 SC_ADPU_Response 类型，该类型在 *smartcard.h* 文件中定义：

```
typedef struct
{
  u8 Data[LCmax];  /* 从智能卡返回的数据 */
  u8 SW1;          /* 命令处理状态 */
  u8 SW2;          /* 命令处理限定 */
} SC_ADPU_Response;
```

- **Data**

  指定指向将包含返回的智能卡数据的数据缓冲区的指针。

- **SW1**

  指定第一个状态代码字节。此字节存储错误类别。

- **SW2**

  指定第二个状态代码字节。此字节存储命令特定的状态或错误指示。

示例：

```
/* 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_Response);
```

6.2.2 SC_PowerCmd

相关内容在*表 15* 中介绍。

表 15. SC_PowerCmd

函数名称	SC_PowerCmd
函数原型	`void SC_PowerCmd(FunctionalState NewState);`
行为说明	接通或断开智能卡的电源。
输入参数	NewState：智能卡电源的新状态。 此参数可以是：ENABLE 或 DISABLE。
输出参数	无
返回参数	无
必需的先决条件	无
调用的函数	无

示例：

```
/* 将智能卡上电 */
  SC_PowerCmd(ENABLE);
```

6.2.3 SC_Reset

相关内容在*表 16* 中介绍。

表 16. SC_Reset

函数名称	SC_Reset
函数原型	`void SC_Reset(BitAction ResetState);`
行为说明	将智能卡复位引脚置 1 或清零。
输入参数	ResetState：此参数指定智能卡复位引脚的状态。 BitVal 必须是 BitAction 枚举值之一： - Bit_RESET：将端口引脚清零。 - Bit_SET：将端口引脚置 1。
输出参数	无
返回参数	无
必需的先决条件	无
调用的函数	无

示例：

```
/* 将智能卡复位引脚置 1 */
  SC_Reset(Bit_SET);
```

6.2.4 SC_ParityErrorHandler

相关内容在*表 17* 中介绍。

表 17. SC_ParityErrorHandler

函数名称	SC_ParityErrorHandler
函数原型	`void SC_ParityErrorHandler(void);`
行为说明	重新发送（智能卡）未能正确接收的字节。
输入参数	无
输出参数	无
返回参数	无
必需的先决条件	无
调用的函数	无

示例：

```
/* 将字节重新发送到智能卡 */
SC_ParityErrorHandler();
```

6.2.5 SC_PTSConfig

相关内容在*表 18* 中介绍。

表 18. SC_PTSConfig

函数名称	SC_PTSConfig
函数原型	`void SC_PTSConfig(void);`
行为说明	配置 IO 速度（波特率）通信。
输入参数	无
输出参数	无
返回参数	无
必需的先决条件	必须在 ATR 序列完成后立即调用。
调用的函数	无

示例：

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

6.3 如何向智能卡发送 APDU 命令

有关如何使用 `SC_Handler( )` 函数向智能卡发送 APDU 命令并获取智能卡响应的详细说明，请参见下文。用户必须根据智能卡规范更新 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_Response);
}
```

- **SCState**：存储当前的智能卡状态。
- **SC_ATR_Table**：指向包含智能卡 ATR 帧的数组（通过 `SC_Handler` 函数填充）的指针。

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_Response);
```

- **FileName**：包含 16 位文件标识符。
- **SCState**：存储当前的智能卡状态。
- **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_Response.SW2;
SC_Handler(&SCState, &SC_ADPU, &SC_Response);
```

- **SCState**：存储当前的智能卡状态。
- **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_Response);
```

- **SCState**：存储当前的智能卡状态。
- **OFFSET_MSB**：用于读取数据的偏移量的最高有效字节。
- **OFFSET_LSB**：用于读取数据的偏移量的最低有效字节。
- **LENGTH**：包含要读取的区域的大小（以字节为单位，仅对基本文件有效）。
- **SC_Response->Data**：返回要读取的智能卡数据。
- **SC_Response->SW1** 和 **SC_Response->SW2**：返回给 SC_READ_BINARY 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_Response);
```

- **FileParameters**：包含 16 字节文件参数（文件 ID、文件访问条件...）。
- **SCState**：存储当前的智能卡状态。
- **SC_Response->SW1** 和 **SC_Response->SW2**：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_Response);
```

- **SCState**：存储当前的智能卡状态。
- **OFFSET_MSB**：用于读取数据的偏移量的最高有效字节。
- **OFFSET_LSB**：用于读取数据的偏移量的最低有效字节。
- **LENGTH**：包含要写入的区域的大小（以字节为单位，仅对基本文件有效）。
- **SC_Response->Data**：包含要写入的智能卡数据。
- **SC_Response->SW1** 和 **SC_Response->SW2**：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_Response);
```

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

6.4 奇偶校验错误管理

在 T0 协议中，通过查看奇偶校验位对每个字节进行错误处理。如果实际的奇偶校验位与所传输数据的奇偶校验不符，则必定发生了错误；接收方在检测到奇偶校验错误时会发信号指示要求重新传输字节。这通过将 I/O 线保持在低电平状态（通常在传输字节前将 I/O 线设置为高电平）来完成。当发送方检测到此情况时，会重新发送未被正确接收的字节。

6.4.1 从智能卡向读卡器发送数据

STM32F05xx 能够通过硬件检测已接收数据的奇偶校验错误，方法是在停止位期间下拉数据线。

6.4.2 从读卡器向智能卡发送数据

反过来也一样，如果智能卡下拉 I/O 线来发信号指示出现奇偶校验错误，STM32F05xx 将能够通过硬件检测帧错误。智能卡库使用 `SC_ParityErrorHandler()` 函数检查是否出现奇偶校验错误并管理该错误（如果有的话）。

每次调用 SC_ParityErrorHandler 函数时，该函数都会重新发送检测到错误的字节。

将一个字节从微控制器发送到智能卡后，智能卡会捕捉在 I/O 线上发送的数据。如果在智能卡上检测到奇偶校验错误，则会在停止位期间下拉 I/O 线。将出现帧错误事件，相关 IRQ 事件会调用重新发送最后数据的 `SC_ParityErrorHandler()` 函数。

7 智能卡 (T=0) 接口示例

随智能卡库提供了一个示例，以帮助用户开发自定义应用程序。

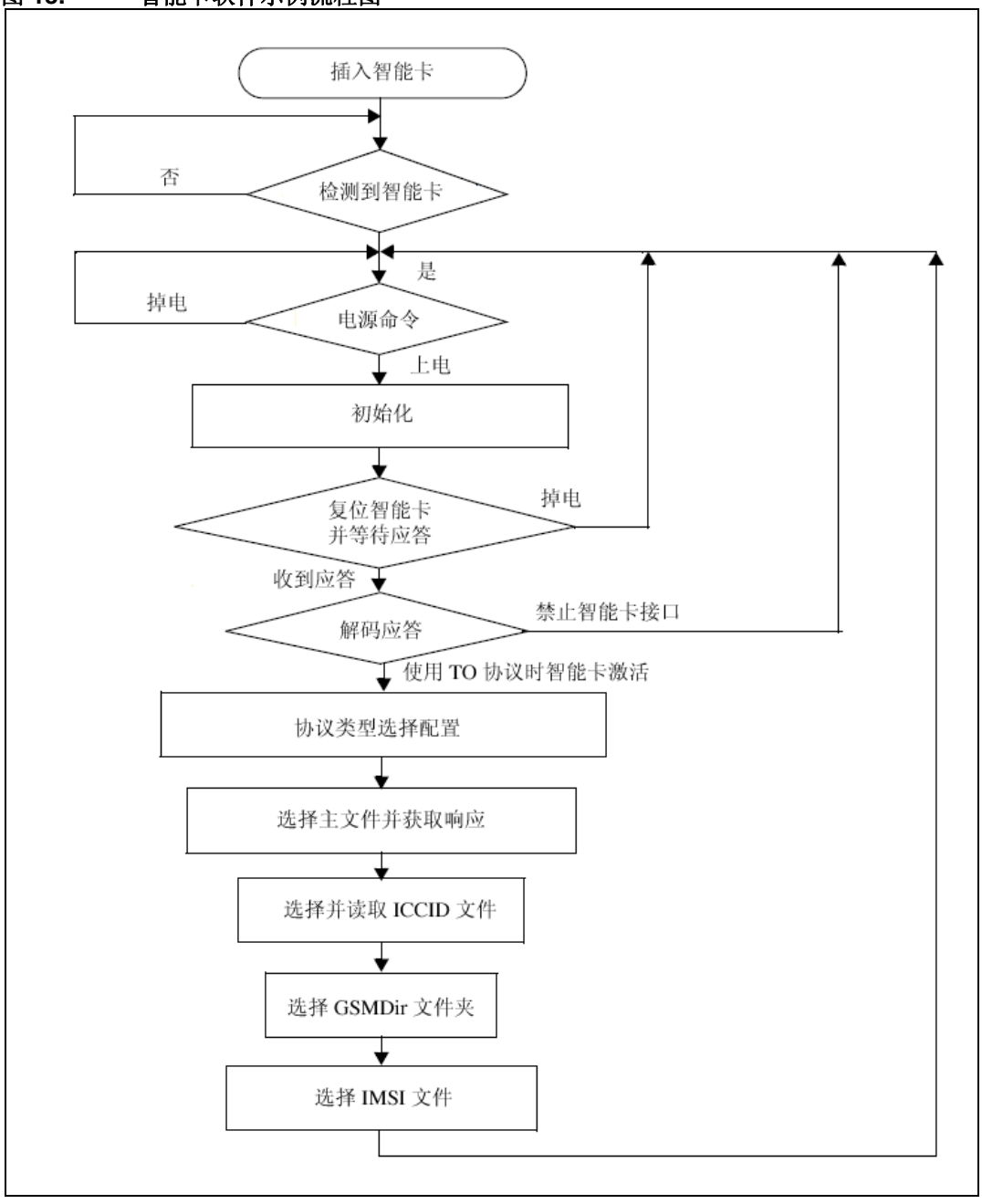
该示例提供了对 ISO 7816-3/4 兼容的 GSM11.11 智能卡的简单操作，例如文件系统浏览、引脚 1 使能/禁止、对文件的读/写操作和对受保护文件的引脚验证。

此示例在意法半导体 STM320518-EVAL 评估板上运行，用户可轻松对其进行定制以适合任何其它硬件。

注： *请记住，STM320518-EVAL 板没有智能卡读卡器。为运行此示例，使用了外部智能卡读卡器，如表 3 中所述。*

7.1 固件说明

图 13. 智能卡软件示例流程图



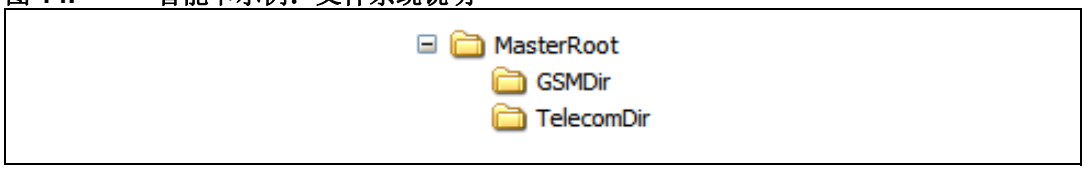
已为 GSM 智能卡目录树定义了三个目录：

MasterRoot[3]={0x3F, 0x00};

GSMDir[3]={0x7F, 0x20};

TelecomDir[3]={0x7F, 0x10};

图 14. 智能卡示例：文件系统说明



在示例结束时：

- 读取位于 MasterRoot 目录下的 ICCID 文件
- 读取 GSMDir 下的 IMSI 文件，该文件将通过 PIN1 获得安全访问权限
- 使能/禁止 PIN1

7.1.1 智能卡启动：复位应答 (A2R)

当读卡器想要访问智能卡时，要执行的第一个操作是复位应答过程。为此，将调用 SC_Handler，如下所示：

```
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_Response);
}
```

在检测到智能卡时，将生成、接收和解码 A2R 序列。如果识别的协议是 T0 协议，读卡器的智能卡状态将变为活动状态 (SC_ACTIVE_ON_T0)，并且智能卡可用于文件系统上的操作。

在 ATR 使用 *SC_PTSConfig()* 函数后，将应用过程类型选择 (PTS)。对于使用的 GSM 智能卡，PTS 过程如下：

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

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

7.1.2 读取指定路径的文件

假设指定的读取路径为：MasterRoot/GSMDir/IMSI

要访问该路径，请执行以下操作：

- 使用 SelectFile APDU 命令选择 GSMDir
- 使用 SelectFile APDU 命令选择 IMSI 文件

用户必须获得文件特性才能检查其访问条件。IMSI 文件具有 CHV1(PIN1) 读取条件，因此必须先检查 PIN1，然后才能读取。必须在包含要读取的文件的目录下执行验证命令，必须在示例中选择 GSMDir。

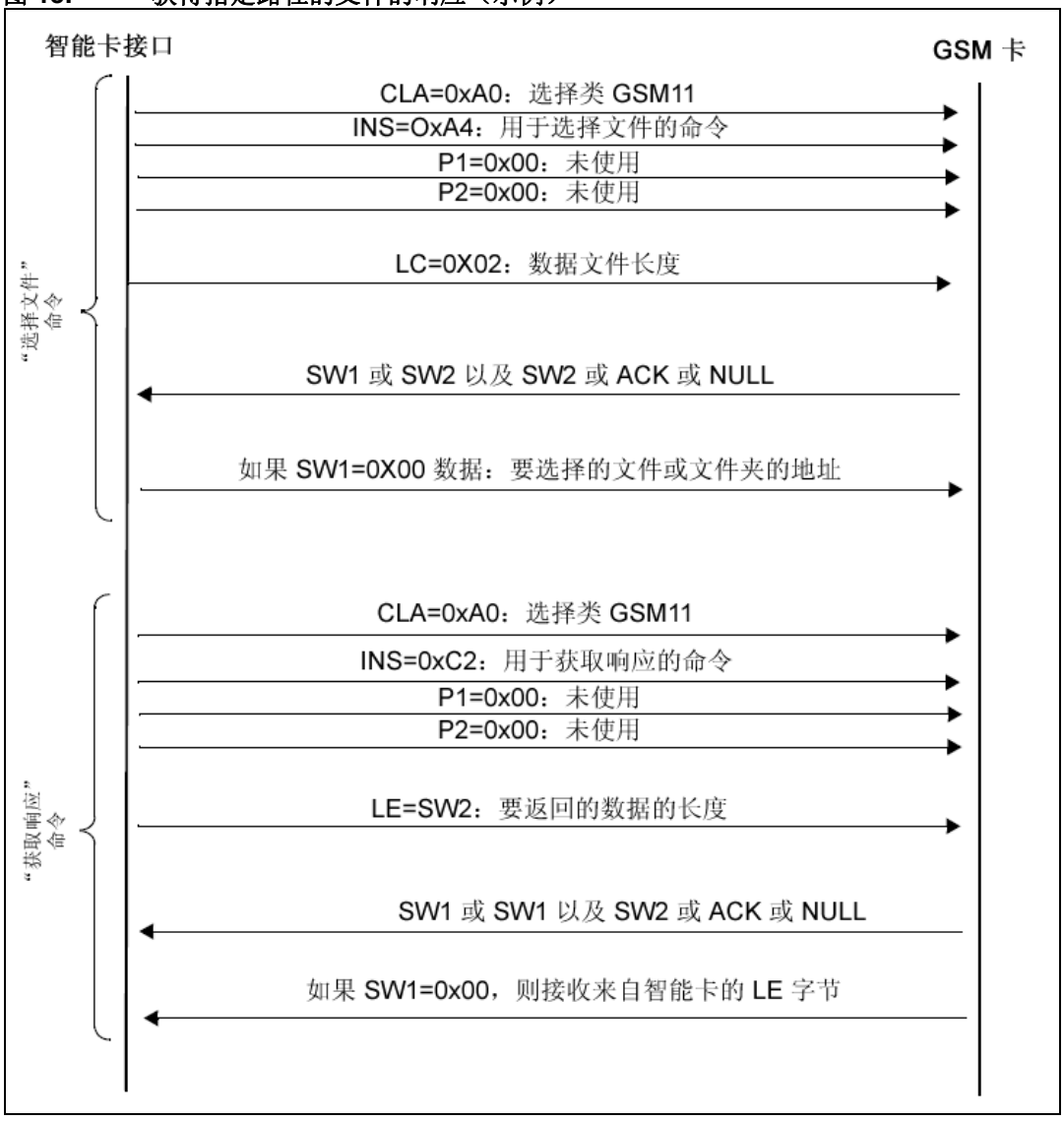
要获得文件特性：

- 选择需要其特性的文件
- 在发出 SelectFile APDU 命令后，通过发送 GETRESPONSE 命令获得返回的数据

IMSI 文件具有 9 个数据字节；要正确运行 READ_BINARY 命令，请使用以下参数：

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

图 15. 获得指定路径的文件的响应（示例）



7.1.3 使能/禁止 PIN1 (CHV1) 代码

在应用程序开始时，检查 PIN1 (CHV1) 状态。这可以通过在成功选择 MasterRoot 后发出 GET_RESPONSE 命令完成。

在此示例中，检查 PIN1 状态。如果将其使能，则会禁止 PIN1 来为所有具有安全访问条件的文件提供访问权限。

要检查 PIN1 状态，请执行以下过程：

- 使用 SelectFile APDU 命令选择 MasterRoot 目录。
- 在发出 MasterRoot SelectFile 命令后，通过发送 GETRESPONSE 命令获得返回的数据。
- 检查 14 个已接收字节的位 8。如果位 8 为 0，将使能 PIN1；否则，将禁止 PIN1。

要使能或禁止 PIN1，必须将 CHV1 代码发送到智能卡。CHV1 代码为 8 字节长。

7.1.4 验证 PIN1 (CHV1) 代码

某些文件具有受限制的访问条件，如 IMSI、CHV1 文件、CHV2。用户必须符合这些文件的访问条件，才能执行受限制的操作（读取、更新、更改 CHV1）。

要验证 PIN1 (CHV1) 条件，请执行以下过程：

- 转到包含要访问的文件的目录。
- 通过提供 CHV1 代码来使用“验证 APDU”命令。
- 选择文件并执行所选操作。

8 智能卡 (T=1) 接口库：说明

STM32F05xx 的智能卡 T1 库是一个 USART 驱动程序，允许用户按照 ISO 7816-3 标准与智能卡进行通信。

8.1 文件组织

表 19 列出了智能卡 T=1 库模块。

表 19. T1 文件库说明

文件	说明
t1_hal.h、t1_hal.c	– 智能卡固件函数 – 物理层
t1_protocol.h、t1_protocol.c	T=1 协议函数
t1_protocol_param.h、t1_protocol_param.c	T=1 参数配置函数
pps.h、pps.c	PPS 函数
atr.h、atr.c	ATR 函数

8.2 智能卡接口库函数

智能卡 T=1 库包含以下用于与支持 T=1 协议的智能卡进行通信的函数（*表 20*）。

表 20. 智能卡 T =1 库函数

函数名称	说明
T1_Protocol_Init	使用默认值初始化 T=1 协议。
T1_SetParameter	设置 T=1 协议参数。
T1_Negotiate_IFSD	发送 IFS 请求以指示可以支持的新 IFSD（读卡器侧）。
T1_APDU	通过 T=1 协议发送 APDU 命令并接收 APDU 响应。

8.2.1 T1_Protocol_Init 函数

相关内容在*表 21* 中介绍。

表 21. T1_Protocol_Init

函数名称	T1_Protocol_Init
函数原型	void T1_Protocol_Init (t1_TypeDef * t1, uint32_t sc_freq)
说明	使用默认值初始化 T=1 协议的结构。
参数 1	t1：指向 T=1 协议的 t1 结构的指针
参数 2	sc_freq：智能卡时钟频率
返回参数	无
必需的先决条件	无

8.2.2 T1_SetParameter 函数

相关内容在*表 22* 中介绍。

表 22. T1_SetParameter

函数名称	T1_SetParameter
函数原型	int32_t T1_SetParameter (t1_TypeDef * t1, uint8_t param_type, uint32_t value)
说明	将 T=1 协议参数设置为 t1 结构。
参数 1	t1：指向 T=1 协议的 t1 结构的指针
参数 2	param_type：要建立的参数
参数 3	value：参数的新值
返回参数	状态：支持或不支持的参数
必需的先决条件	无

8.2.3 T1_Negotiate_IFSD 函数

相关内容在*表 23* 中介绍。

表 23. T1_SetParameter

函数名称	T1_SetParameter
函数原型	int32_t T1_Negotiate_IFSD (t1_TypeDef * t1, uint8_t nad, uint8_t ifsd)
说明	发送 IFS 请求以指示可以支持的新 IFSD（读卡器侧）。
参数 1	t1：指向 T=1 协议的 t1 结构的指针
参数 2	nad：节点地址的值
参数 3	ifsd：要协商的读卡器 IFS 的新值
返回参数	IFSD 请求通信的状态
必需的先决条件	T1 协议已经初始化。

8.2.4 T1_APDU 函数

相关内容在*表 24* 中介绍。

表 24. T1_APDU

函数名称	T1_APDU
函数原型	int32_t T1_APDU (t1_TypeDef * t1, uint8_t nad, void *apdu_c, uint32_t apdu_c_len, void *apdu_r, uint32_t apdu_r_len)
说明	通过 T=1 协议发送 APDU 命令并接收 APDU 响应。
参数 1	t1：指向 T=1 协议的 t1 结构的指针
参数 2	nad：nad 值（节点地址）
参数 3	apdu_c：指向 ADPU 命令缓冲区的指针
参数 4	apdu_c_len：ADPU 命令缓冲区的长度
参数 5	apdu_r：指向 ADPU 命令缓冲区的指针
参数 6	apdu_r_len：指向 ADPU 命令缓冲区的指针
返回参数	APDU 处理的状态
必需的先决条件	T1 协议已经初始化。

9 智能卡 (T=1) 接口示例

随智能卡库提供了一个示例，以帮助用户将 USART 外设与 STM32F05xx 微控制器配合使用来开发智能卡 (T=1) 应用程序。

该示例介绍了如何使用智能卡 T=1 协议驱动程序以及如何与 Java Card 进行通信。其中包括发送某些 APDU 命令，例如“选择文件”、“读取文件”和“写入文件”。

此示例在意法半导体 STM320518-EVAL 评估板上运行，用户可轻松对其进行定制以适合任何其它硬件。

注： *请记住，STM320518-EVAL 板没有智能卡读卡器。为运行此示例，使用了外部智能卡读卡器，如表 3 中所述。*

9.1 固件说明

示例首先应用“复位应答”过程，然后根据收到的 ATR 配置智能卡接口和 T=1 协议。

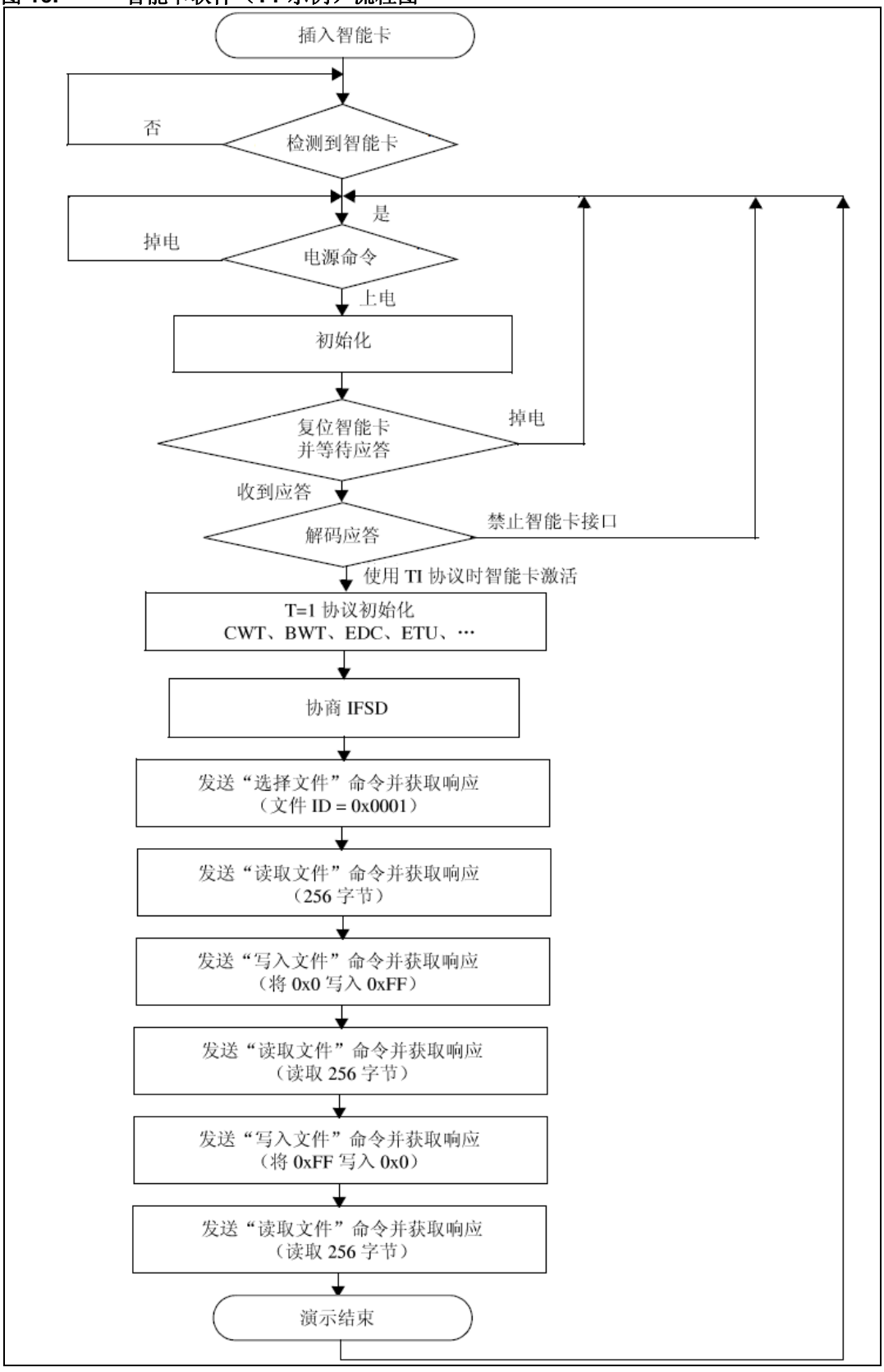
之后，演示将发送 IFSD 请求，指示读卡器将块的信息字段的长度从 32 延长到 254。

随后，按照以下顺序开始发送/接收 APDU 命令/响应：

1. 选择 ID = 0x0001 的文件。
2. 读取文件：从同一文件读取 256 字节（以查看文件中保存的初始数据）
3. 写入文件：向此文件写入 0x0 到 0xFF。
4. 读取文件：从文件读取 256 字节（以验证写入的数据）。
5. 写入文件：向此文件写入 0xFF 到 0x0。
6. 读取文件：从文件读取 256 字节（以验证写入的数据）。
7. 演示结束。

请参见图 16 中的流程图。

图 16. 智能卡软件（T1 示例）流程图



10 结论

由于 STM32F05xx USART 的智能卡模式支持 ISO 7816-3/4 规范，用户可使用较少的固件和硬件资源来开发基于智能卡的应用程序。

11 版本历史

表 25. 文档版本历史

日期	版本	变更
2012 年 08 月 03 日	1	初始版本。

请仔细阅读：

中文翻译仅为方便阅读之目的。该翻译也许不是对本文档最新版本的翻译，如有任何不同，以最新版本的英文原版文档为准。

本文档中信息的提供仅与 ST 产品有关。意法半导体公司及其子公司 （“ST”）保留随时对本文档及本文所述产品与服务进行变更、更正、修改或改进的权利，恕不另行通知。

所有 ST 产品均根据 ST 的销售条款出售。

买方自行负责对本文所述 ST 产品和服务的选择和使用， ST 概不承担与选择或使用本文所述 ST 产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为 ST 授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在 ST 的销售条款中另有说明，否则，ST 对 ST 产品的使用和 / 或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

意法半导体的产品不得应用于武器。此外，意法半导体产品也不是为下列用途而设计并不得应用于下列用途：（A）对安全性有特别要求的应用，例如，生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）汽车应用或汽车环境，且 / 或（D）航天应用或航天环境。如果意法半导体产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向意法半导体发出了书面通知，采购商仍将独自承担因此而导致的任何风险，意法半导体的产品设计规格明确指定的汽车、汽车安全或医疗工业领域专用产品除外。根据相关政府主管部门的规定，ESCC、QML 或 JAN 正式认证产品适用于航天应用。

经销的 ST 产品如有不同于本文档中提出的声明和 / 或技术特点的规定，将立即导致 ST 针对本文所述 ST 产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大 ST 的任何责任。

ST 和 ST 徽标是 ST 在各个国家或地区的商标或注册商标。

本文档中的信息取代之前提供的所有信息。

ST 徽标是意法半导体公司的注册商标。其他所有名称是其各自所有者的财产。

意法半导体集团公司

澳大利亚 - 比利时 - 巴西 - 加拿大 - 中国 - 捷克共和国 - 芬兰 - 法国 - 德国 - 中国香港 - 印度 - 以色列 - 意大利 - 日本 - 马来西亚 - 马耳他 - 摩洛哥 – 菲律宾 – 新加坡 - 西班牙 - 瑞典 - 瑞士 - 英国 - 美国

www.st.com