# 作业答案

## 1 第一章

目的：了解 RFID 以及物联网；

1．选择一种常见天线，结合应用场景分析其特性

农村或者城中村中，常见的八木天线：由一个有源振子（一般用折合振子）、一个无源反射器和若干个无源引向器平行排列而成的端射式天线。方向性好，价格便宜，”适合自制”；

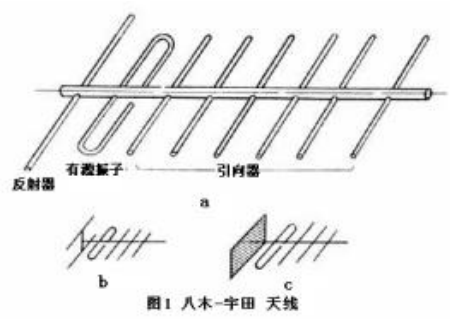


图1 八木-宇田 天线

卫星（通信）天线、卫星电视天线（锅），是一个金属抛物面，负责将卫星信号反射到位于焦点处的馈源和高频头内。

电台/电视台/转播站的全向天线；



无线通信基站（塔）的全向或者定向天线；


开放性题目，希望大家看到天线，研究探索其特性；

2．请谈谈你所理解的 RFID 及其在物联网中的作用；

建议答案：分两部分，a：RFID 是啥？b：rfid 在物联网中的作用；

对于 a：RFID 是一种器件，当它处于 Reader 生成的射频场时，会以无线通信方式，自动返回预先配置的数据，该数据作为该器件的 ID，这种方式被称为为 TTF：Tag Talk First；或者等待 Reader 发送的命令后，再做出应答，返回比 ID 更多的数据信息。

RFID 器件可以是无源，半无源，有源，其中无源 RFID 因为不需要配备电源（从 Reader 生成的射频场中耦合），因此应用更方便，应用范围更大；

随着 IC 出现，RFID 内逐渐使用 IC 存贮 ID 信息，而随着 IC 技术的发展，IC 的集成度更高，RFID 内存贮的信息也更多，应用的场景也更加丰富，在安全性要求更高的金融领域，芯片已经是一个符合 GP(GlobalPlatform)规范的计算机了。

RFID 进行无线通信时，使用的频谱包括 LF，HF，UHF，SHF。LF 和 HF 采用电感耦合方式通信，从射频场中耦合电源，通信距离较短，速率低，但一般不需要备电；UHF 和 SHF 采用雷达散射耦合方式，通信距离远，速率快，但一般需要备电；

为了 Transponder 省电，RFID 作了不少努力，比如 Reader 和 Transponder 采用相同的工作频谱，Transponder 采用负载调制技术，尽可能简化 Transponder 的复杂性等；

传输速率快，且应用于非可视场景（相比于条码）

期待/理想：更远的通信距离（UHF/SHF）同时不备电的 RFID 技术；

总之：是对第一章全部内容的总结融合。

对于 b：“自动”标注物体，从而实现把“物”接入网络；随着技术发展，RFID 返回的数据增加，不止可以实现“标注”物体的功能（“我是谁”），还可以承载物体的特性（“我是啥样子的”）。前者支持实现传统物联网意义上的功能，比如物流行业中的商品跟踪（获取 ID 信息后，与 ID 相关的信息 online 获取），后者支持更宽广的应用：比如校园卡的电子钱包 / 电子支付，支持 online，也支持 offline：因为 RFID 中实现“存贮 ID”的器件发展到 IC 后，存贮的数据增加 / 甚至是 CPU.

大部分同学了描述 RFID 在物联网中应用后带来的益处。比如应用于物流行业，实现物品自动跟踪；应用于智能家具，实现。。。

但题目本意是：“RFID 在物联网中的作用”：自动标识物体，之后的益处，都是物体被自动标识后带来的。

对于应用场景，希望深入研究/调研和分析，结合已经介绍的 RFID 知识，说明用法：比如智能家具，“可被用于智能家具设备的自动控制。。。”过于粗线条，比如说明：RFID 置于哪里，如何使用…

另外：也有把 RFID 作为实现通信的手段：自动传输传感器信息，这似乎要备电了吧？

3．你觉得 RFID 系统中的关键技术是哪些？请说明原因；

关键技术：1.自动识别，芯片发展，存贮空间扩大（可存贮更多信息），安全性增强；2. 通信方式：射频通信；3. 耦合方式，获得能量；

大家做了研究工作，但超出意外，比如答案涉及：制造技术、系统集成、中间件，安全等，还是建议专注于课内：

其中：都谈到了安全。。。的确是，但不属于 RFID 系统的“关键”，至少不是课程到目前重点关注的。

典型答案：天线/中间件/防碰撞协议/安全

似乎有 copy 的嫌疑。

4．请介绍一个基于 RFID 技术的应用系统。尽可能描述全些： 比如 card/tag/label 的形状，系统组成、系统功能等；

略。

5. 请谈谈你对 RFID 未来的技术发展方向和应用场景的看法

高频：更远的通信距离；

芯片：更大的存贮能力，更小的功耗；

有同学提的更多，比如有

无感

更新的通信技术，比如 WIFI6 + 5G，

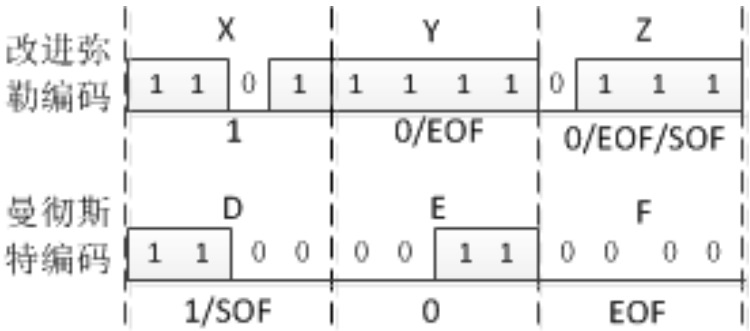
大胆设想，万一可以呢？“虽然 5G 很耗电的，而且 Transponder 不适合做的太复杂！”

# 2 第三章 编码与调制

目的：熟悉编码；

1．关于 ISO/IEC 14443A 协议：

1.0.0，ISO/IEC14443A 协议规定，PCD 采用改进的弥勒码，PICC 采用曼彻斯特码，PCD 和 PICC 的编码如下图示。




1.0.1，上述两种码型可采用 4 位（bit）表示。对于改进的弥勒编码，X 信号（逻辑

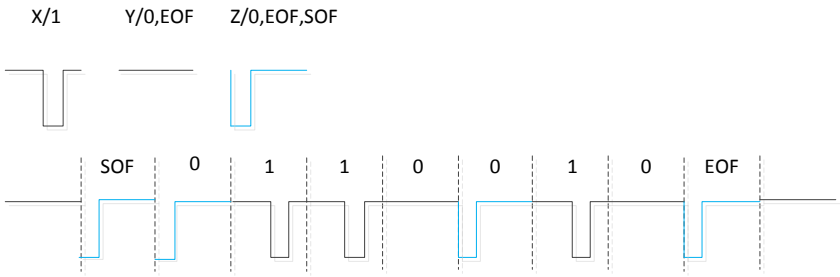
1)用 1101 表示，Y 信号(逻辑 0 和 EOF)用 1111 表示，Z 信号 (逻辑 0 和 EOF、SOF) 用 0111 表示；对于曼彻斯特编码，D 信号（逻辑 1 和 SOF）用 1100 表示，E 信号（逻辑 0）用 0011 表示，F 信号（EOF）0000 表示。

1.1 对于 RFID HF ISO/IEC 14443A 协议，若 PCD 向 PICC 发送长度为 7 个比特的数据，值为 0x26，PICC 响应数据长度为 16 比特，值为：0x084E；双方发送时先发送低比特数据，请画出如上两个方向帧的编码图（数据前后要增加 SOF 和 EOF，ATQA 不比不含 CRC）；

答：PCD->PICC，发送：0x26：0x0010,0110，发送比特为：0110,010，发送波形为：

| SOF | 0 | 1 | 1 | 0 | 0 | 1 | 0 | EOF | |
|---|---|---|---|---|---|---|---|---|---|
| Z | Z | X | X | Y | Z | X | Y | Z | Y |

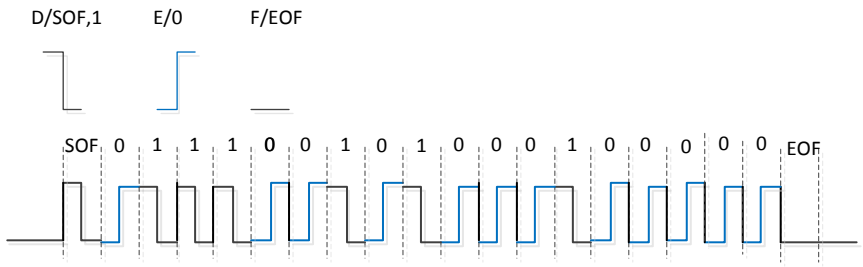
波形如下：




PICC->PCD 方向，发送 0x084E，0000,1000,0100,1110，发送比特为：0111,0010,0001,0000.发送波形为(增加了奇校验)：

| SOF | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | EOF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D | E | D | D | D | E | E | D | E | D | E | E | E | D | E | E | E | E | E | F |

波形如下：

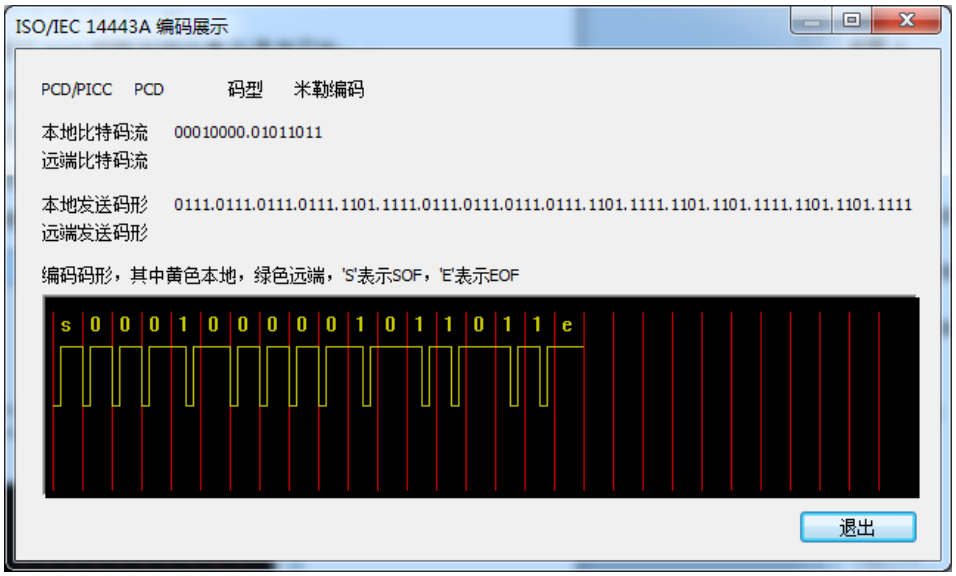



在 PICC->PCD 方向，L 表示没有副载波调制；所以 SOF 的前半部分表示通信开始；

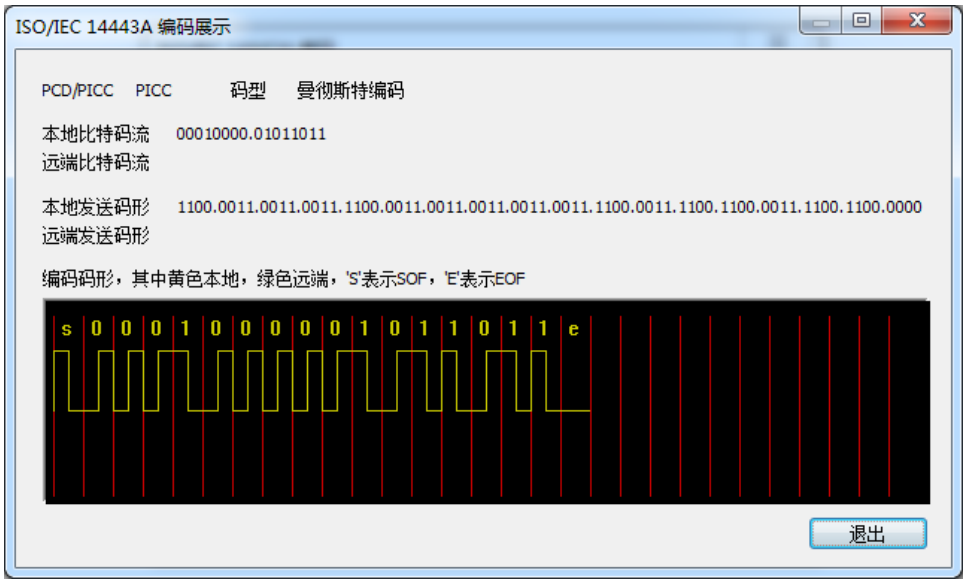
1.2 如 1.0.1 所示，IOS/IEC 14443A 协议 PCD->PICC 和 PICC->PCD 两个方向每比特的编码都可以用 4 位（bit）表示；请编写程序，输出用比特表示的帧编码，包括 SOF 和 EOF，每个 4 个比特用逗号分隔；输入为 16 进制表示的数据；

比如若要输入的数据为：0x08D5，[按先发低比特，数据流为：0x105B]

PCD->PICC：




PICC->PCD：




解：略.

给过学生程序，相当于要读懂程序.

文件：rfid_iso14443A_codec.c/h.

函数说明如下：

1. 单比特编码；

```
int RFID_Code_Miller( BYTE prebit, BYTE bitDat );
int RFID_Code_Manchester( BYTE bitDat ); // bitDat=1 or 0
```

2. PCD->PICC 编码和 PICC->PCD 编码：

```
int ISO14443A_PCD_Code(
            BYTE aInBuf[], UINT16 u16InBitLen,
            BYTE aOutBuf[], UINT16 *pu16OutBitLen
        );

int ISO14443A_PICC_Code(
        BYTE aInBuf[], UINT16 u16InBitLen,
        BYTE aoutBuf[], UINT16 *pu16OutBitLen
    );
```

文件：rfid_iso14443A_frm.cpp/h 中，实现了 framing，其中输入的是命令帧，函数：ISO14443A_Framing()实现：计算 CRC 校验，增加奇校验，按最低比特优先顺序；其输出为：字节 0,1,…，每个字节内，bit7,6,…,1,0.

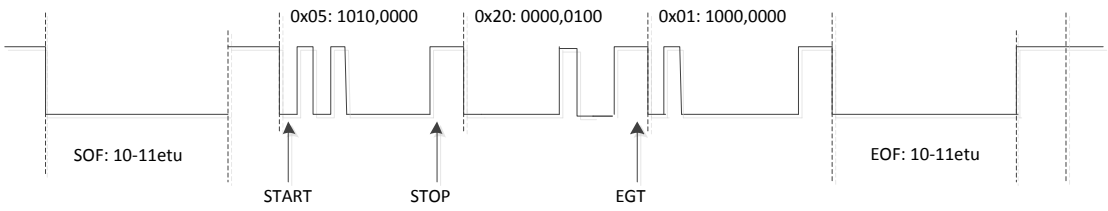
```
int ISO14443A_Framing(
    BYTE        byBitLenofFirstByte,
    BYTE        byFirstByte,
    BYTE        ainBuf[], UINT16 u16inBitLen,
    BYTE        aoutBuf[], UINT16 *pu16OutBitLen
     );
```

2．关于 ISO/IEC 14443B 协议：

- 若 PCD 向 PICC 发送的字节为：0x052001，请画出该帧的编码示意图；
- 若 PICC 向 PCD 发送的字节为：0x2B，请画出该帧的编码示意图；

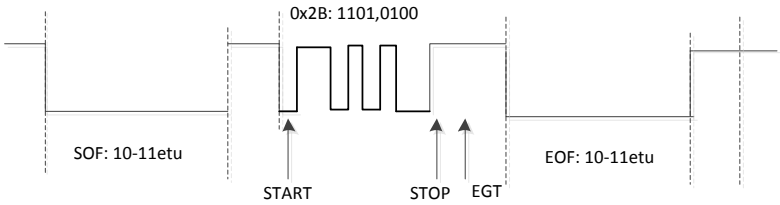
解：PCD->PICC 方向，采用 NRZ-L 编码，1 用 H 表示，0 用低表示，调制方式为 ASK,调制指数为 10%，发送时低比特优先。

0x052001 帧的编码示意图为：




PICC->PCD 方向，NRZ-L 编码，1 用 H 表示，0 用低表示，（副载波）调制方式为：BPSK.

0x2B 帧的编码示意图为：




3．以 ISO/IEC 14443B 为例，谈谈你理解的数字通信系统中的信道编码与脉冲调制（副载波调制）的作用；

解：信道编码是将数字数据编码成适合于在数字信道上传输的数字信号，并具有所需的抵抗差错的能力，即通过相应的编码方法使接收端能具有检错或纠错能力。

对于 ISO/IEC 14443B,PCD->PICC 方向采用 NRZ-L 编码,调制方式为调制指数 10%的 ASK，其优点是 PCD 实现简单，尤其是 PICC 更简单。采用 10% ASK 信号，确保了射频场中一直存在载波，保证了为 PICC 供电；

NRZ-L 编码方式虽然不含丰富的同步信号，但 14443B 在帧构成方面做了补偿。14443B 采用异步通信方式，每次传输由 10 个 etu 构成，包括 1 个 START 比特，8bit 数据，1 个 STOP 比特，以及额外保护时间 EGT，通过 START 信号达到收方同步到发方的目的。

在 PICC->PCD 方向，采用副载波调制，有用信息的频谱分布在副载波附近而不是在载波附近，便于阅读器对传送数据信息的提取；

在大部分场景下（比如 14443A），采用副载波调制，还可以减少 PICC 的能量损耗，这对于无源 PICC 尤其重要，无源 PICC 靠电感耦合从 PCD 获得能量。在采用副载波调制信号进行负载调制时，调制管每次导通时间较短，对应答器电源影响较小；调制管的总导通时间减少，总功率损耗下降；

但对于 14443B 而言，并没有上述效用。由于采用的副载波调制方式为 BPSK，调制管的总导通时间并未减少(逻辑 1 时一半导通，逻辑 0 时一半导通)。

4．ISO/IEC15693 协议：

- VCD->VICC 方向采用脉冲位置编码，用图示方法说明协议如何定义帧的开始和结束。
- 说明 VICC->VCD 方向采用的编码方式，用图示方法说明协议如何定义帧的开始和结束。

解：

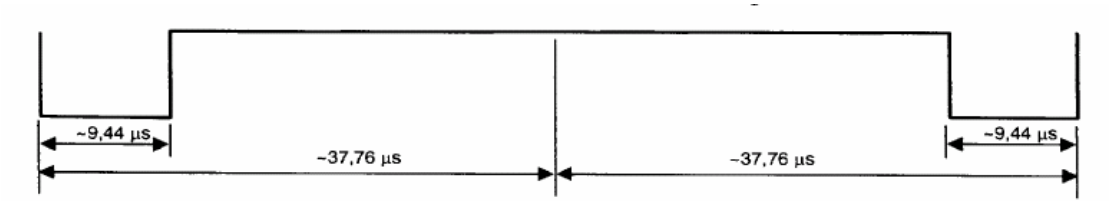
VCD->VICC:

IEC15693 采用的脉冲位置编码方式为 256 取 1，或者 4 取 1.在无发送数据帧时保持高电平。协议采用”编码违例”表示 SOF 和 EOF。

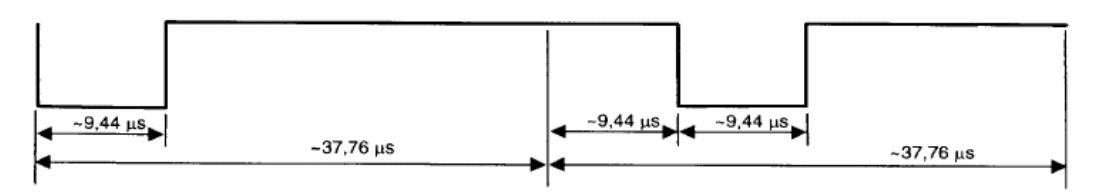
对于 SOF，无论 256 取 1，或者 4 取 1，在 4 个连续的 ETU 时间内，都不可能出现两个

PAUSE，由此定义 SOF。

256 取 1 的 SOF 定义如下：

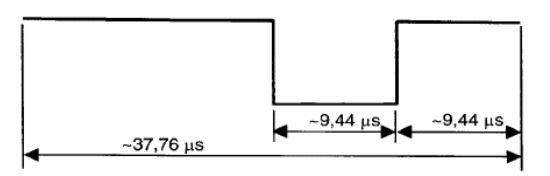



4 取 1 的 SOF 定义如下：




其中的编码违例有两个：1. 4 个 ETU 时间内，有两个 PAUSE；2. 第一个 PAUSE 在前半个位置；可用于接收端同步。

PAUSE 位于前半个位置“编码违例”表示 EOF，如下：

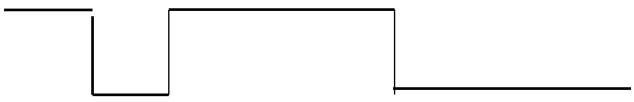



VICC->VCD 方向:采用反相曼彻斯特编码方式,H-L 表示逻辑”0”，L-H 表示逻辑”1”。

正常情况下，无论传输”1”或者”0”，H/L 电平的保持时间都不会超过 1 个 ETU 时间（一个 ETU 时间内，H/L 电平各保持 0.5ETU，1 和 0 连续，出现 1 个 ETU 的高电平，0 和 1 连续,出现 1 个 ETU 的低电平),因此定义出现超过 1 个 ETU 的 H/L 电平的信号表示 SOF/EOF。实际为：1.5 个 ETU。

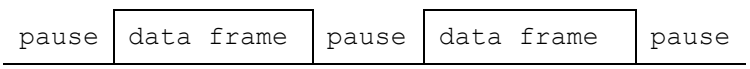
SOF：

EOF：



L 表示没有副载波；

5、某种 RFID HF 应答器芯片，片内有一个 32 比特只读存贮器，其进入阅读器射频场并上电复位成功后，周期性地送出存贮器内的数据，输出帧如下图.



其中 pause 阶段输出低电平，数据帧格式如下

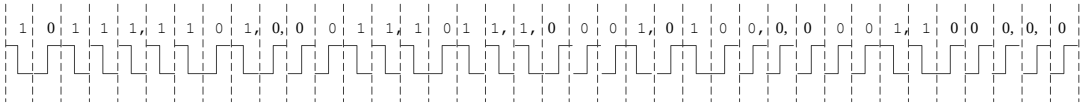
| 输出顺序 | 1 | 2~9 | 10 | 11~18 | 19 | 20~27 | 28 | 29~36 | 37 | 38 |
|---|---|---|---|---|---|---|---|---|---|---|
| 输出信息 | SOF | b0~b7 | 1 | b8~b15 | 1 | b16~b23 | 0 | b24~b31 | 1 | EOF |

其中 SOF 为逻辑 1，EOF 为逻辑 0；若只读存贮器中 32 位数据为：0x1828DCBE，线路编码为曼彻斯特编码，请写出该数据桢(精确到比特)，并画出数据帧的线路码波形。

答：帧如下：

| 输出顺序 | 1 | 2~9 | 10 | 11~18 | 19 | 20~27 | 28 | 29~36 | 37 | 38 |
|---|---|---|---|---|---|---|---|---|---|---|
| 输出信息 | SOF | b0~b7 | P | b8~b15 | P | b16~b23 | P | b24~b31 | P | EOF |
| 逻辑值 | 1 | 0111,1101 | 0 | 0011,1011 | 1 | 0001,0100, | 0 | 0001,1000, | 0 | 0 |

波形图：



# 3 第四章 数据校验和防碰撞算法

目的：掌握防碰撞算法：

一.假设一个读卡器的射频场中存在如下三张卡，请针对 ISO/IEC14443A 写出读卡器识别三张卡的过程（遇到比特碰撞，取 1 继续）：

- 0x12345678; 0x123456789ABCDE; 0x123456789ABCDEF01234;

解：

PICC1 的等效 ID：0x12345678

PICC2 的等效 ID：0x12345678,9ABCDE88

PICC3 的等效 ID：0x12345678,9ABCDE88,F0123488

0. PCD 发送 REQA，三张卡应答 ATQA;

1. PCD 发出：ANTICOLLISION，0x93，0x20，三个 PICC 应答：

PICC1：发送：0x12345678，　　最先 8 比特：0001,1110

PICC2：发送：0x9ABCDE88，　　最先 8 比特：0001,0001

PICC3：发送：0xF0123488，　　最先 8 比特：0001,0001

比特 5 位置发生冲突；

2. PCD 发出：ANTICOLLISION，0x93，0x25，bit0-3 为：0001，指定 bit4=1,只有 PICC1 应答：PICC1：发送除最低 5 比特外的其余 27 比特，不发生冲突；
3. PCD 发送 SELECT 命令，ID 为 0x12345678，PICC1 以 SAK 应答。PCD 找出了第一个 PICC.
4. PCD 访问 PICC1，结束后向 PICC1 发送 HALT 命令。

重新开始寻找 PICC2 和 PICC3.

1. PCD 发出：ANTICOLLISION，0x93，0x25，bit0-3 为 0001，bit4 = 0，2 个 PICC 应答：

PICC2：发送：0x9ABCDE88，　　最先 7 比特：　001,0111

PICC3：发送：0xF0123488，　　最先 7 比特：　001,0010

第 8 个位置，比特 9 发生冲突

2. PCD 发出：ANTICOLLISION，0x93，0x32，0x88,(0001,0001),bit8=0，指定 bit9=1，1 个 PICC 应答：

PICC2：发送除最低 10 比特外的其余 22 比特，不发生冲突；

3. PCD 发送 SELECT 命令(0x93)，ID 为 0x9ABCDE88，PICC2 以 SAK 应答，通知 PCD 还有更多 ID.
4. PCD 发出：ANTICOLLISION,SEL=0x95.0x20，1 个 PCD 应答：

PICC2：发送 0x12345678，不发送冲突；

5. PCD 发送 SELECT 命令(0x95)，ID 为 0x12345678，PICC2 以 SAK 应答，PCD 找出了第一个 PICC.
6. PCD 访问 PICC2，结束后向 PICC2 发送 HALT 命令。
7. 找出了第 2 个 PICC.

继续：

2. 1.PCD 发出:ANTICOLLISION,0x93,0x32，0x88,(0001,0001),bit8=0,指定 bit9=0，1 个 PICC 应答：1 个 PICC 应答：

PICC3：发送：0xF0123488，不发生冲突；

3. PCD 发送 SELECT 命令，ID 为 0xF0123488，PICC3 以 SAK 应答，通知 PCD 还有更多 ID.
4. PCD 发出：ANTICOLLISION，0x95. 0x20，1 个 PCD 应答：

PICC3：发送 0x9ABCDE88，不发送冲突；

5. PCD 发送 SELECT 命令(0x95)，ID 为 0x9ABCDE88，PICC3 以 SAK 应答，通知 PCD 还

有更多 ID.

6. PCD 发出：ANTICOLLISION, 0x97.0x20 1 个 PCD 应答：
   PICC3：发送 0x12345678，不发送冲突；
7. PCD 发送 SELECT 命令(0x97)，ID 为 0x12345678，PICC3 以 SAK 应答。PCD 找出了 PICC3.
8. PCD 访问 PICC3，结束后向 PICC3 发送 HALT 命令。
9. 找出了第 3 个 PICC.

注意：以上过程：

1. 没有包含标准帧和防碰撞帧内每个字节的奇校验；
2. 没有描述帧结构，其中标准帧不包含 CRC，防碰撞帧没有包含 UID 的 BCC 校验；

二.假设一个读卡器的射频场中存在如下三张卡,请针对 ISO/IEC14443B 写出读卡器识别三张卡的过程（AFI=0x10，卡选择 SLOT 号取自 PUPI 对 SLOT 数求余）：

- PICC1：PUPI=0x12345678,AFI=0x10;
- PICC2：PUPI=0x3456789A,AFI=0x10;
- PICC3：PUPI=0x56789ABC,AFI=0x20;

解：

1. PCD 发送 REQB 命令,制定 AFI=0x10,M=0, N(SLOT)=1;PICC3 不应答;PICC1 和 PICC2 应答：
2. PCD 发现冲突，设置：M=2, N(SLOT)=4:
   PICC1，对 4 求余，为 0，应答；
   PICC2，对 4 求余，为 2，不应答；
3. PCD 收到 PICC1 的应答，识别出 PICC1.
4. PCD 发送 SLOT-MARKER, SLOT=2,没有 PICC 应答；
5. PCD 发送 SLOT-MARKER, SLOT=3,PICC2 应答，识别出 PICC2.
6. PCD 发送 SLOT-MARKER, SLOT=4,没有 PICC 应答；

三．假设一个读卡器的射频场中存在如下三张卡，请针对 ISO/IEC15693 写出读卡器识别三张卡的过程：

- VICC1：0xE01234567890ABCD; VICC2：E0567890ABCDEF1D;
- VICC3：0xE090ABCDEF123456;

解：

1. VCD 发送：INVENTORY 命令，MASK LENGTH=4,MASK VALUE=0，没有应答；
2. VCD 发送：EOF，没有应答(1)；
3. …
4. VCD 发送：EOF，没有应答(5)；
5. VCD 发送：EOF，VICC3 应答；
6. VCD 发送：EOF，没有应答(7)；
7. …
8. VCD 发送：EOF，没有应答(C)；
9. VCD 发送：EOF，VICC1 和 VICC2 应答；
10. VCD 发送：EOF，没有应答(E)；
11. VCD 发送：EOF，没有应答(F)；
12. VCDs 识别出 VICC3，访问 VICC3，结束后，发送 STAYQUITE 命令；

13. VCD 发送：INVENTORY 命令，MASK LENGTH=8,MASK VALUE=0x0D，没有应答；
14. VCD 发送：EOF，VICC2 应答；
15. VCD 发送：EOF，没有应答(2)；
16. …
17. VCD 发送：EOF，没有应答(B)；
18. VCD 发送：EOF，VICC1 应答；
19. VCD 发送：EOF，没有应答(D)；
20. …
21. VCD 发送：EOF，没有应答(F)；
22. VCDs 识别出 VICC2，访问 VICC2，结束后，发送 STAYQUITE 命令；
23. VCDs 识别出 VICC1，访问 VICC1，结束后，发送 STAYQUITE 命令；

## 增加如下：

写出下列字符串“Hello RFID”中每个字符的奇校验位。

| 字符 | H | e | l | l | o | | R | F | I | D |
|---|---|---|---|---|---|---|---|---|---|---|
| ASCII | 0x48 | 0x65 | 0x6C | 0x6C | 0x6F | 0x20 | 0x52 | 0x46 | 0x49 | 0x44 |
| 奇校验 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

2．写出 RFID HF 15693,14443A/B 协议采用的 CRC16 校验的生成多项式，编写程序实现该 CRC 校验算法（输入：协议类型 和 字符串，输出 CRC 校验）给出三种协议下，字符串”Hello RFID”的 CRC 校验码；

代码：

Crc 文件：crc16.c

```
#include "rfid_crc.h"
UINT16 UpdateCrc(BYTE ch, UINT16 *lpwCrc)
{
    ch = (ch^(unsigned char)((*lpwCrc) & 0x00FF));
    ch = (ch^(ch<<4));

    *lpwCrc       =       (*lpwCrc       >>       8)^((UINT16)ch       <<
8)^((UINT16)ch<<3)^((UINT16)ch>>4);
    return(*lpwCrc);
}

/**
 *  CRC 校验
 *
 *  \input  BYTE    aBuf[],缓冲区
 *          UINT16  u16Len 缓冲区长度
 *          UINT16  u16InitPrm  CRC16 初值.
 *  \ouput  BYTE    *pCRCMSB    CRC 高 8bit
 *          BYTE    *pCRCLSB    CRC 低 8bit
 *  \return 1 成功，<0 失败.
 */
int CRC16( UINT16 u16InitValue, BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE
*pCRCLSB )
{
    UINT16 wCrc;
    unsigned char chBlock;
    int i;

    wCrc = u16InitValue;
    for( i = 0; i < u16Len; i++ )
    {
        chBlock = aBuf[i];
        UpdateCrc(chBlock, &wCrc);
    }
    if( u16InitValue == 0xFFFF )
    {
        wCrc = ~wCrc; // ISO 3309
    }
    *pCRCLSB = (BYTE) (wCrc & 0xFF);
    *pCRCMSB = (BYTE) ((wCrc >> 8) & 0xFF);

    return 1;
}
```

```
int ISO14443A_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB )
{
    return CRC16( 0x6363, aBuf, u16Len, pCRCMSB, pCRCLSB );
}
int ISO14443B_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB )
{
    return CRC16( 0xFFFF, aBuf, u16Len, pCRCMSB, pCRCLSB );
}
int ISO15693_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB )
{
    return CRC16( 0xFFFF, aBuf, u16Len, pCRCMSB, pCRCLSB );
}
```

头文件：rfid_crc.h

```
#ifndef _RFID_CRC_H_
#define _RFID_CRC_H_
typedef unsigned char BYTE;
typedef unsigned short UINT16;
int ISO14443A_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB );
int ISO14443B_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB );
int ISO15693_CRC( BYTE aBuf[], UINT16 u16Len,BYTE *pCRCMSB, BYTE *pCRCLSB );
#endif
```

主程序：main.c

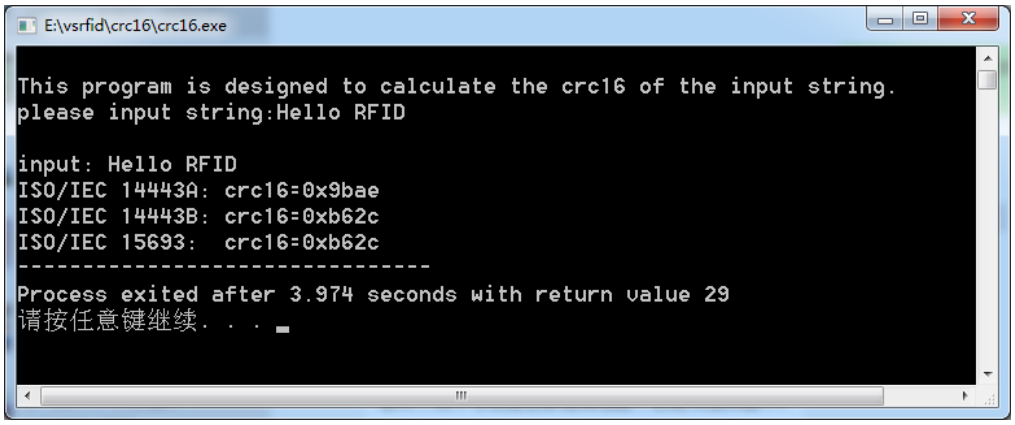
```
#include <stdio.h>
#include <string.h>
#include "rfid_crc.h"

int main( int argc, char *argv[])
{
    char str[200];
    int protocol;
    unsigned char crc_msb, crc_lsb;

    printf("\nThis program is designed to calculate the crc16 of the input
string.");
    printf("\nplease input string:");
    gets( str );
    printf( "\ninput: %s", str );
    ISO14443A_CRC( str, strlen(str), &crc_msb, &crc_lsb);
    printf("\nISO/IEC 14443A: crc16=0x%02x%02x", crc_msb, crc_lsb);
    ISO14443B_CRC( str, strlen(str), &crc_msb, &crc_lsb);
    printf("\nISO/IEC 14443B: crc16=0x%02x%02x", crc_msb, crc_lsb);
```

```
    ISO15693_CRC( str, strlen(str), &crc_msb, &crc_lsb);
    printf("\nISO/IEC 15693:  crc16=0x%02x%02x", crc_msb, crc_lsb);

}
```




3．对于 14443A 协议，若 PICC 输出的 ATQA 为：0x0046，请写出 REQA 和 ATQA 的帧结构，并画出这两个帧的编码。

REQA：0x26，帧结构如下：

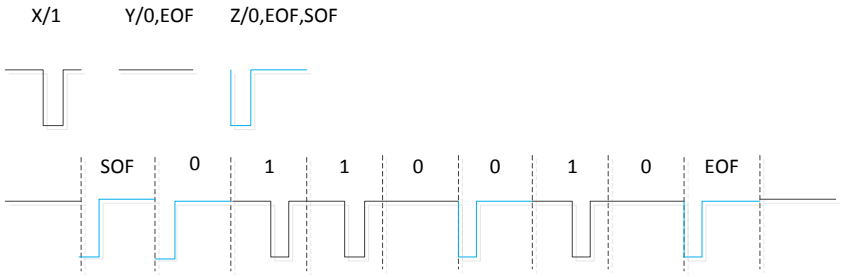
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| SOF | 0x26 | | | | | | EOF |
| SOF | 0 | 1 | 1 | 0 | 0 | 1 | 0 | EOF |

REQA 编码采用改进的弥勒编码，编码如下：

发送波形为：

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| SOF | 0 | 1 | 1 | 0 | 0 | 1 | 0 | EOF | |
| Z | Z | X | X | Y | Z | X | Y | Z | Y |

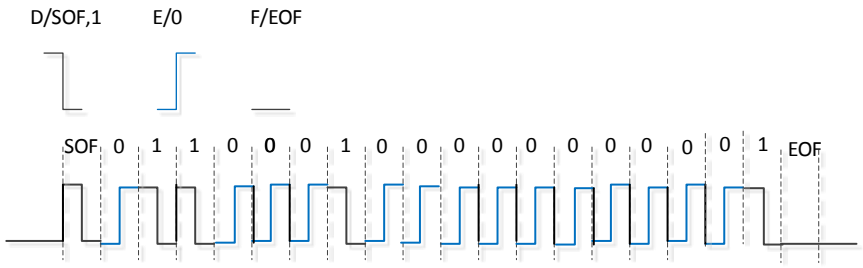
波形如下：




PICC->PCD 方向，发送 0x0046，0000，0000，0100，0110，发送比特为：0110，0010，0000，0000．发送波形为（增加了奇校验）：

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SOF | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | EOF |
| D | E | D | D | E | E | E | D | E | E | E | E | E | E | E | E | E | D | F |

波形如下：




在 PICC->PCD 方向，L 表示没有副载波调制；所以 SOF 的前半部分表示通信开始；

4.一张符合 14443A 协议的 PICC，其 UID = 0x12345678；请画出其防碰撞帧（设 PCD 已经接收到了 13 比特，新的命令中，第 1 4 比特取 1 ）

该 PICC 的 UID 的 BCC=0x08，完整的防碰撞帧为：

| S | SEL | | NVB | | UID0 | | UID1 | | UID2 | | UID3 | | BCC | | E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | 0x93 | 1 | 0x36 | 1 | 0x78 | 1 | 0x56 | 1 | 0x34 | 0 | 0x12 | 1 | 0x08 | 1 | E |
| S | 1100 1001 | 1 | 0110 1100 | 1 | 0001 1110 | 1 | 0110 1010 | 1 | 0010 1100 | 0 | 0100 1000 | 1 | 0001 0000 | 1 | E |

其中：NVB=0x36，是因为 PCD 发送 UID 的 14 比特时，PCD 发送 3 字节（SEL+NVB+UID0）以及 UID1 的低 6 比特；

PCD 方向发送的帧结构为：

| S | SEL | | NVB | | UID0 | | UID1 (低 6 比特) |
|---|---|---|---|---|---|---|---|
| S | 0x93 | 1 | 0x36 | 1 | 0x78 | 1 | 低 6 比特 |
| E | 11001001 | 1 | 0x11000110 | 1 | 00011110 | 1 | 0110,11 |

由于 UID1 的 bit5=1，但 PICC 的 UID1 的 bit5=0，因此该 PICC 不做应答；

若新的命令中，第 14 比特取 0，此时 PCD 发送的帧结构为：

| S | SEL | | NVB | | UID0 | | UID1 (低 6 比特) |
|---|---|---|---|---|---|---|---|
| S | 0x93 | 1 | 0x36 | 1 | 0x78 | 1 | 低 6 比特 |
| E | 11001001 | 1 | 0x11000110 | 1 | 00011110 | 1 | 0110,10 |

此时，PICC 应答，PICC 方向发送的帧结构为：

| UID1 (高 2 比特) | | UID2 | | UID3 | | BCC | | E |
|---|---|---|---|---|---|---|---|---|
| 0x56 (高 2 比特) | | 0x34 | 0 | 0x12 | 1 | 0x08 | 1 | E |
| 10 | x | 00101100 | 0 | 01001000 | 1 | 00010000 | 1 | E |

5. 运行 PCDSim 和 PICCSim，分析防碰撞协议执行的过程：

4 字节 UID：以学号作为 14443A 协议的 ID 号，

7 字节 UID：高字节为：0x3E4D5C；低 4 字节为学号；

6．研读 15693 协议：命令及其操作；

答案 略．

# 4 第八章 13.56MHz RFID 技术

目的：了解如何使用 RFID，开发系统。

1．如果请你为学校构建一套基于 RFID 技术的一卡通系统， 芯片采用 MF S50，请规划 S50 的存贮区，并设计存贮区的安全特性。系统具有如下功能：

1） 门禁功能：

- 记录进出宿舍、图书馆信息；
- 离线查看最近 10 条记录；

2） 钱包功能：

- 当前总金额；
- 最近 10 笔明细数据：充值、消费（金额、Reader ID 等）；

3） 图书借阅：

- 当前借阅的图书信息：书名、借出日期、归还日期等；
- 最多 5 本图书；

4） 其他：

- 有想法就提；

解：(例如)：

S50 芯片共 16 个 sectors，每个 sector 3 个可用 blocks，每个 block 16 字节，实际可用空间 768 个字节。

下述有关时间的定义：4 字节，单位秒，表示：自 1970 年 1 月 1 日 0 时 0 分 0 秒到现在的时间；

空间分配如下：

1）门禁：每条记录 16 字节，4 字节：时间秒，2 字节：读卡器 ID，其中包含了楼栋信息（宿舍、图书馆）；1 字节：in/out；1 字节保留；采用 4 个 sectors，可以存贮 12 条记录；

2）钱包功能：每笔交易占用一个 block/字节，如下：

表6 消费交易记录区数据结构（区8、区9）

| 名称 | 大小 (Byte) | 编码 | 说明 |
|---|---|---|---|
| 交易时间 | 5 | BCD | 年月日时分（YYMMDDHHNN）。 |
| 原额 | 3 | HEX | |
| 交易金额 | 2 | HEX | |
| 交易类型 | 1 | HEX | |
| 系统代码 | 1 | HEX | |
| 复用字段 | 4 | | 根据系统不同，有不同定义。见表 7 |

分配 4 个 sectors，共 12 个 blocks，实际支持 12 条记录。

钱包占用一个 scrotrs.

表 4 钱包及充值记录区（区 2）

| 块号 | 名称 | 大小 (Byte) | 编码 | 说明 |
|---|---|---|---|---|
| 0 | 原额 | 3 | HEX | 充值前金额，钱包单位为人民币分。 |
| | 余额 | 3 | HEX | 充值后金额，钱包单位为人民币分。 |
| | 充值员 ID | 2 | BCD | 表示本充值点充值员的 ID 码 |
| | 充值日期 | 3 | BCD | YYMMDD |
| | 充值机代码 | 3 | BCD | |
| | （预留） | 2 | | |
| 1 | 公共钱包 | 4 | HEX | |
| | 公共钱包 | 4 | HEX | 含阴影的字段为原值取反 |
| | 公共钱包 | 4 | HEX | |
| | ADDRESS | 4 | HEX | |
| 2 | 公共钱包(备份) | 4 | HEX | |
| | 公共钱包(备份) | 4 | HEX | 含阴影的字段为原值取反 |
| | 公共钱包(备份) | 4 | HEX | |
| | ADDRESS | 4 | HEX | |
| 3 | Key A2 | 6 | | |
| | Access | 4 | | |
| | Key B2 | 6 | | |

3）图书：分配 5 个 sectors，每个 sectors 存贮一条图书信息。每条图书信息定义如下：24 字节：

借出日期、归还日期，各 4 字节

20 字节：书名；

信息如下：

| | sector | 存贮方式 | 安全 |
|---|---|---|---|
| 发行 | 0 | | |

| | | | |
|---|---|---|---|
| 门禁 | 1-4 | 每条记录占用 1 个 block，共 12 条记录 | C1C2C3=100，<br>Read: KEYA or KEYB<br>write: KEYB<br>门禁端掌握 KEYB，可读可写；<br>用户掌握 KEYA，可读； |
| 图书 | 10-14 | 每条记录占用 1 个 sector，共条记录。 | |
| 钱包 | 5 | 钱包 | C1C2C3=001，<br>KEYA/KEYB，可读可消费<br>C1C2C3=110，<br>通过修改 KEYB，执行充值操作 |
| | 6-9 | 每条记录占用 1 个 block，共 12 条记录 | C1C2C3=010,不可写；<br>Read: KEYA or KEYB<br>用户掌握 KEYA，可读；<br>充值/消费终端具有 KEYB，可读； |
| 备用 | 15 | | |

2．某人 L 和你一起组成一个开发小组，参与某个基于 RFID 的系统开发工作。你们小组负责 Reader 端软件开发，RFID 芯片采用 SL2 ICS20，L 担任项目经理，负责 ICS20 空间分配，并承担和上位机（计算机）之间的接口，你承担面向 RFID Card 的软件开发工程师工作，要求实现一组对 ICS20 访问的 APIs。请给出如下 API 的接口函数（用 C 语言描述），并参考 ICS20 Datasheet 和 ISO15693 协议，详细画/写出 VCD 到 VICC 和 VICC 到 VCD 之间数据传输的数据格式（写出各个字节的内容，其中 CRC 不必计算）。

- Inventory: inventory
- Stay quiet:
- 读一块存贮器(Read Single Block);
- 写一块存贮器(Write Single Block);
- 读多块存贮器(Read multiple Blocks);
- 写多块存贮器(Write multiple Blocks);

解：所有函数返回：int，表示函数是否执行成功。<0 失败；

- Inventory: inventory

request

| SOF | Flags | Inventory | Optional AFI | Mask length | Mask value | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 8 bits | 8 bits | 0 – 64 bits | 16 bits | |

**Figure 12 — Inventory request format**

response

| SOF | Flags | DSFID | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 13 — Inventory response format**

函数名:

```
int inventory(
    uint8 u8Flag,
    uint8 u8AFI,
    uint8 u8MaskLen,            // bitlen
    uint8 au8MaskValue[8],      // .
    uint8 *pu8RxFlags,
    uint8 *pu8RxDSFID,
    uint8 au8RxUID[8]
    );
```

返回: 1, 表示清晰收到一个 VICC 的应答;

0, 表示没有 VICC 应答;

<0, 表示有多个 VICC 应答;

如果返回 < 0, 或者: VCD 发送 EOF, 继续点名, 或者: 修改 masklen & maskvalue, 重新发送 inventory()

■ Stay quiet:

| SOF | Flags | Stay quiet | UID | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 16 bits | |

**Figure 14 — Stay quiet request format**

The Stay quiet command shall always be executed in Addressed mode (Select_flag is set to 0 and Adddress_flag is set to 1).

没有 response.

函数名:

```
int stayQuiet(
    uint8 u8Flags,
    uint8 au8UID[8]
);
```

■ 读一块存贮器(Read Single Block);

request:

| SOF | Flags | Read single block | UID | Block number | CRC16 | EOF |
|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 16 bits | |

**Figure 15 — Read single block request format**

If the Option_flag is set in the request, the VICC shall return the block security status, followed by the block value.

If it is not set, the VICC shall return only the block value.

request 中, 是否存在 UID, 由 flags 决定 (address mode or select mode)。

response:

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 16 — Read single block response format when Error_flag is set**

| SOF | Flags | Block security status | Data | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | Block length | 16 bits | |

**Figure 17 — Read single block response format when Error_flag is NOT set**

函数名:

```
int readSingleBlk(
```

```
        uint8 u8Flags,
        uint8 au8UID[8],
        uint8 u8BlkNo,
        uint8 *pu8RxFlags,
        uint8 *pu8ErrCode
        uint8 *pu8BlkSecurityStatus,
        uint8 au8Dat[],
    );
```

返回值：flags.

是否返回：pu8BlkSecurityStatus，要看 u8Flags 中 optional_flag.

或者：

```
// 返回：0，正确，否则 Flags.
Uint8 readSingleBlk(
        uint8 u8VisitMode,      // 1 for address mode, 0 for select mode
        uint8 u8Optionalflag;   // 1 return security status;
        uint8 au8UID[8],
        uint8 u8BlkNo,
        uint8 *pu8BlkSecurityStatus,    // optionalflag = 1 时，有效;
        uint8 au8Dat[],
    );
```

■ 写一块存贮器(Write Single Block)；

| SOF | Flags | Write single block | UID | Block number | Data | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | Block length | 16bits | |

**Figure 18 — Write single block request format**

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 19 — Write single block response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 20 — Write single block response format when Error_flag is NOT set**

```
int writeSingleBlk(
     uint8 u8Flags,
     uint8 au8UID[8],
     uint8 u8BlkNo,
     uint8 au8Dat[],
     uint8 *pu8RxFlags,
     uint8 *pu8ErrCode
 );
```

■ 读多块存贮器(Read multiple Blocks);

Request:

| SOF | Flags | Read multiple block | UID | First block number | Number of blocks | CRC16 | EOF |
|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | 16 bits | |

**Figure 24 — Read multiple blocks request format**

response

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 25 — Read multiple blocks response format when Error_flag is set**

| SOF | Flags | Block security status | Data | CRC16 | EOF |
|---|---|---|---|---|---|
| | 8 bits | 8 bits | Block length | 16 bits | |
| | | Repeated as needed | | | |

**Figure 26 — Read multiple block response format when Error_flag is NOT set**

```
int readMultipleBlks(
      uint8 u8Flags,
      uint8 au8UID[8],
      uint8 u8BlkFirstNo,
      uint8 u8BlkNums,
      uint8 *pu8RxFlags,
      uint8 *pu8ErrCode
      uint8 au8BlkSecurityStatus[],
      uint8 au8Dat[]
);
```

其中：au8Dat[]：所有 block 的数据连续保存；

■ 写多块存贮器(Write multiple Blocks)；

Request:

| SOF | Flags | Write multiple block | UID | First block number | Number of blocks | Data | CRC16 | EOF |
|---|---|---|---|---|---|---|---|---|
| | 8 bits | 8 bits | 64 bits | 8 bits | 8 bits | Block length | 16 bits | |
| | | | | | | Repeated as needed | | |

**Figure 27 — Write multiple blocks request format**

Response:

| SOF | Flags | Error code | CRC16 | EOF |
|---|---|---|---|---|
| | 8 bits | 8 bits | 16 bits | |

**Figure 28 — Write multiple blocks response format when Error_flag is set**

| SOF | Flags | CRC16 | EOF |
|---|---|---|---|
| | 8 bits | 16 bits | |

**Figure 29 — Write multiple block response format when Error_flag is NOT set**

```
int writeMultipleBlks (
     uint8 u8Flags,
     uint8 au8UID[8],
     uint8 u8BlkFirstNo,
     uint8 u8BlkNums,
     uint8 au8Dat[],
     uint8 *pu8RxFlags,
     uint8 *pu8ErrCode
);
```