



[12] 发明专利申请公布说明书

[21] 申请号 200680003352.6

[43] 公开日 2008 年 5 月 7 日

[11] 公开号 CN 101176355A

[22] 申请日 2006.1.10

[21] 申请号 200680003352.6

[30] 优先权

[32] 2005.2.4 [33] US [31] 11/051,015

[86] 国际申请 PCT/US2006/000636 2006.1.10

[87] 国际公布 WO2006/083493 英 2006.8.10

[85] 进入国家阶段日期 2007.7.27

[71] 申请人 爱特梅尔公司

地址 美国加利福尼亚州

[72] 发明人 菲利普·D·德雷森

[74] 专利代理机构 北京律盟知识产权代理有限责任公司
代理人 王允方 刘国伟

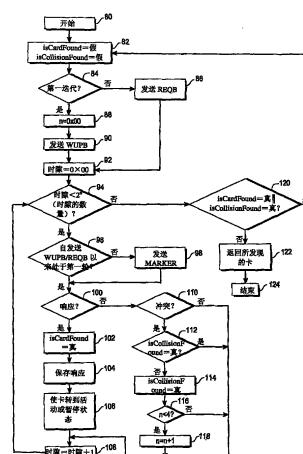
权利要求书 2 页 说明书 9 页 附图 3 页

[54] 发明名称

用于在非接触系统中进行 RF 卡检测的方法

[57] 摘要

本发明提供一种识别读卡器场中的 RF 卡的方法和用于执行所述方法的处理器可读存储媒体。 将轮询请求(90)发送到读卡器场中的卡。 当从正在响应的卡接收到来自所发送命令的清楚响应时(100)，保存所述响应(104)，所述正在响应的卡转到活动状态(106)，且如有可能，时隙数递增(108)，并重复上述步骤直到接收不到清楚的响应为止。 当接收不到清楚响应时(100)，确定是否已发生冲突(110)。 如果没有发生冲突，则时隙数递增(108)，如果确实发生冲突，则发送另一轮询请求(86)。 如果在轮询每一时隙之后没有检测到任何卡或冲突，则返回所发现的卡(122)。



1、一种用于识别处于 RF 读卡器场中的任何 RF 卡的方法，所述方法包括：

a) 从一组命令中发送命令，所述一组命令包括：

i) 第一轮询命令，其用于请求来自所述场中处于防冲突状态中的任何卡的响应，所述第一轮询命令还使任何处于暂停状态中的卡转到所述防冲突状态；

ii) 第二轮询命令，其用于请求来自所述场中处于所述防冲突状态中的任何卡的响应；及

iii) 第三轮询命令，其用于请求来自所述场中处于所述防冲突状态中的任何卡的响应，所述防冲突状态已选择要作出响应的特定时隙；

b) 当从正在响应的卡接收到来自所述所发送命令的清楚响应时：

i) 保存所述响应；

ii) 使所述正在响应的卡转到活动状态；

iii) 递增时隙数，且如果所述递增的时隙数超过预定的最大时隙数，则转到步骤 d)，否则重复步骤 a)和 b)，直到接收不到清楚的响应为止；

c) 当未从正在响应的卡接收到清楚的响应时，确定是否发生冲突，且如果是，则重复步骤 a)和 b)，直到接收不到清楚的响应为止，且如果没有发生冲突，则转到步骤 biii)；

d) 返回一值，指示正在响应的卡的数量。

2、如权利要求 1 所述的方法，其进一步包括当发送所述第一或第二轮询命令时，指示可用的时隙数。

3、如权利要求 1 所述的方法，其进一步包括清除所述活动状态。

4、如权利要求 1 所述的方法，其进一步包括当超过可用的卡识别号的数量时，使所述正在响应的卡转到暂停状态。

5、如权利要求 1 所述的方法，其进一步包括当发现在所述场中不再有卡时返回一值，以指示所述正在响应的卡的数量。

6、如权利要求 1 所述的方法，其进一步包括当填满用于活动状态卡的 CID 时隙的所述数量时返回一值，以指示所述正在响应的卡的数量。

7、如权利要求 1 所述的方法，其进一步包括规定将要作为目标的卡的类型。

8、如权利要求 1 所述的方法，其中所述第一轮询命令是将要发送的第一命令。

9、如权利要求 1 所述的方法，其进一步包括增大时隙的数量。

10、一种处理器可读存储媒体，在所述处理器可读存储媒体上包含处理器可读代码，所述处理器可读代码用于对处理器进行编程以执行用于识别 RF 读卡器场中的任何 RF 卡的方法，所述方法包括：

a) 从列表发送命令，所述列表包括：

-
- i) 第一轮询命令,其用于请求来自所述场中处于防冲突状态中的任何卡的响应,所述第一轮询命令还使处于暂停状态中的任何卡转到所述防冲突状态;
 - ii) 第二轮询命令,其用于请求来自所述场中处于所述防冲突状态中的任何卡的响应;及
 - iii) 第三轮询命令,其用于请求来自所述场中处于所述防冲突状态中的任何卡的响应,所述防冲突状态已选择要在其中作出响应的特定时隙;
 - b) 当从正在响应的卡接收到来自所述所发送命令的清楚响应时:
 - i) 保存所述响应;
 - ii) 使所述正在响应的卡转到活动状态;
 - iii) 递增时隙数,且如果所述递增的时隙数超过预定的时隙数量,则转到步骤 d),否则重复步骤 a)和 b),直到接收不到清楚响应为止;
 - c) 当未从正在响应的卡接收到清楚的响应时,确定是否发生冲突,且如果是,则重复步骤 a)和 b),直到接收不到清楚的响应为止,且如果没有发生冲突,则转到步骤 biii);
 - d) 返回一值,以指示正在响应的卡的数量。

11、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括当发送所述第一或第二轮询命令时指示可用的时隙的数量。

12、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括清除所述活动状态。

13、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括当超过可用的卡识别号的数量时使所述正在响应的卡转到暂停状态。

14、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括当发现在所述场中不再有卡时返回一值,以指示所述正在响应的卡的数量。

15、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括当填满所述用于活动状态的 CID 时隙的数量时返回一值,以指示所述正在响应的卡的数量。

16、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括规定将要作为目标的卡的类型。

17、如权利要求 10 所述的处理器可读存储媒体,其中所述第一轮询命令是将要发送的第一命令。

18、如权利要求 10 所述的处理器可读存储媒体,所述方法进一步包括增大时隙的数量。

用于在非接触系统中进行 RF 卡检测的方法

技术领域

本发明涉及检测 RF 读卡器场中的 RF 卡。

背景技术

射频识别（“RFID”）正变得越来越普遍。RFID 使用无线射频通信来识别并采集关于对象的信息。在对象被识别之后，可通过无线方式创建或更新关于所述对象的信息。RFID 可用于广泛的应用范围，包括身份卡。

RFID 系统由一个或多个读卡器及转发器—与读卡器通信的非接触式存储器标签—组成。转发器可安装在身份卡上。转发器包含使卡能够被识别的数据，并由与射频通信块相关联的存储器构成。读卡器连接至天线，所述天线自转发器接收和向转发器发送射频信号。在图 1 中，现有技术的读卡器 68 可与其 RF 场 66 中的所有卡 70、72、74 和 76 通信。读卡器 68 不能与其场 66 之外的卡 78 进行通信。

当读卡器与其场中的转换器或卡进行通信时，如果有不止一个转换器或卡同时尝试答复所述读卡器，则偶尔会发生冲突。当发生冲突时，读卡器不能从任意一卡接收到清楚的响应。

可采用防冲突机制来防止有不止一个卡同时发送信息。概率性防冲突机制可重复提示卡在单个时隙内答复。然而，使用这种方法，直到正在响应的卡移开所述场，或者离开其可响应来自读卡器请求的状态，才能检测到多个卡的存在。当读卡器将发出轮询命令来提示来自己经选择响应时隙的卡的响应时，可采用时隙防冲突方法。

将需要一种方法，其能够记录出现在读卡器场中的卡的列表，以在处理一卡之前知晓存在哪些卡。将进一步需要知晓卡所处的状态，并能够改变卡的状态。

发明内容

本发明提供一种识别 RF 读卡器场中的 RF 卡的方法和用于执行所述方法的处理器可读媒体。所述方法包括从一组命令中发送一命令，所述一组命令包括：第一轮询命令，其用于请求来自场中处于防冲突状态的任何卡的响应，所述第一轮询命令还使任何处于暂停状态中的卡转到防冲突状态；第二轮询命令，其用于请求来自场中处于防冲突状态的任何卡的响应；及第三轮询命令，其用于请求来自场中处于防冲突状态的任何卡的响应，所述防冲突状态已经选择了响应的具体时隙。当从正在响应的卡接收到来自所发送命令的清楚响应时，保存响应，且所述正在响应的卡转到活动状态，

且如有可能，递增时隙数，并重复上述步骤直到接收不到清楚的响应为止。如果不能递增时隙数，则返回一个值，以指示正在响应的卡的数量。当接收不到来自正在响应的卡的清楚响应时，确定是否发生冲突，且如果已发生冲突，则重复上述步骤，直到接收不到清楚的响应为止。如果未发生冲突，则返回一个值，以指示正在响应的卡的数量。

附图说明

图 1 是 RF 读卡器的方块图，其中如现有技术中所已知，在读卡器的 RF 场的内外均有 RF 卡。

图 2 是如现有技术中所已知的 RF 卡的状态图。

图 3 是一实例性流程图，其显示如何根据本发明来识别 RF 卡。

图 4 是一实例性流程图，其显示根据本发明如何使 RF 卡转到活动状态。

具体实施方式

用于下文所述实施例中的命令是用于 ISO/IEC 14443 标准中所述 B 型信令的命令。然而，在其他实施例中可采用来自其他标准及/或其他信令方案的命令。相关的命令包括：Request B ("REQB"); Wake-Up B ("WUPB"); Slot Marker ("MARKER"); Proximity Card Selection Command ("ATTRIB"); 及 Halt B ("HTLB")。

REQB 向所有防冲突状态邻近卡（“PICC”或“卡”）发出轮询请求。当发送该命令时，将时隙数“N”分配给 PICC，PICC 然后从 1 至“N”中选择随机值作为其时隙分配。在 PICC 选择时隙值“1”时，将立即用 Answer to Request of Type B ("ATQB") 作出响应。重新分配“N”的值，并用每一 REQB 和 WUPB（下文所述）命令选择新的随机时隙。

WUPB 类似于 REQB，但具有一个区别。当发送 WUPB 命令时，使处于暂停状态下的卡转到防冲突状态。

MARKER 或 SLOT MARKER 是另一轮询命令，其为已经选择一特定时隙的卡发出轮询请求。已经选择用于发送该命令的特定时隙的卡将用 ATQB 响应来响应该请求。

ATTRIB 用于使 PICC 转到活动状态。选择一接收 ATTRIB 命令的卡并将其分配至专用信道。需要将未使用的卡识别码（“CID”）值分配给卡，以便卡处于活动状态，因为活动状态命令使用 CID 来引用所述卡。

HTLB 命令使卡转到暂停状态。卡将保持处于暂停状态中，直到被复位或接收 WUPB 命令。

在图 2 中，使现有技术的卡（未示出）复位，或进入 RF 场(10)。卡可以处于下述三个状态之一：防冲突(12)、活动(14)、或暂停(16)。如果卡处于暂停状态(16)中，则可由 WUPB 命令(24)来将其“唤醒”，所述 WUPB 命令(24)会使卡转到防冲突状态

(12)。在防冲突状态(12)中，卡将响应上文所述的以下命令：WUPB、REQB、SLOT MARKER、ATTRIB、及 HLTB。从防冲突状态(12)，可通过 ATTRIB 命令(18)使卡转到活动状态(14)。在活动状态(14)中，命令因卡的类型而不同，不过在处于活动状态中的卡上不会执行任何防冲突命令。可通过卡专用的反选命令(20)使卡从活动状态(14)转到暂停状态(16)。还可通过 HLTB 命令(22)使卡从防冲突状态(12)转到暂停状态(16)。

可通过软件、硬件、或软件和硬件的组合来实施用于识别读卡器场中的 RF 卡的本发明。当本发明的全部或一部分实施在软件中时，该软件可驻留在处理器可读存储媒体（其实例包括但不限于磁光盘、光盘、或半导体存储媒体，例如软磁盘、硬磁盘、CD ROM、存储器 IC 等）上。处理器可读存储媒体存储能够将处理器（其与处理器可读存储媒体通信）编程的代码，以执行实施本发明的步骤。

在图 3 中，当开始了用于识别读卡器场中的所有卡的算法时（方块 80），isCardFound（是否在算法的上一迭代中发现卡的真或假指示符；当调用时，该值应为真）和 isCollisionFound（是否在算法的上一迭代中发现冲突的真或假指示符；当调用时，该值应为真）均设定成“假”（方块 82）。在算法的第一迭代中（方块 84），首先将值“n”（其确定防冲突时隙“N”的数量（其中 $N = 2^n$ ））设定成 0x00（方块 88）。发送 WUPB 命令——使处于暂停状态中的卡转到防冲突状态并请求所有处于防冲突状态中的卡作出响应的轮询请求（方块 90）。WUPB 命令还将“n”的值通知读卡器场中的卡，所述“n”的值告知卡可用的时隙数（从 0 至 $2^n - 1$ 的数值）。当发送 WUPB 命令时，N 的值分配至场中的卡，每一卡从 1 至 N 中选择一随机数值作为卡的时隙分配。在每次发送 WUPB 或 REQB 命令时，N 的值和时隙分配发生变化。还设定期限（“隙”）的初始值（此时，初始值设定成 0x00，不过在其他实施例中可改变该值）（方块 92）。

如果算法不是处于其第一迭代中（方块 84）（下文对其进行更详细的论述），则发送 REQB 命令——请求处于防冲突状态中的所有卡作出响应的轮询命令（方块 86）。如上文所述，在每次发送 REQB 命令时，n 和 N 的值发生改变，且根据 n 的当前值来计算防冲突时隙 N 的数量，且如上文所述设定期限（“隙”）的初始值（方块 92）。（在该实施例中， $0 < n \leq 4$ ，因此 N 的最大值是 16。在其他实施例中可使用 n 的不同的范围和起始值。）

如果时隙的值小于时隙的数量（方块 94），且自从发出 WUPB 或 REQB 命令以来，算法处于其第一轮上（换句话说，在该实施例中，时隙=0x00）（方块 96），并接收到有效响应（方块 100），则将 isCardFound 设定成真（方块 102）且保存所述响应（方块 104）。如果自从发出 WUPB 或 REQB 命令以来算法不是处于其第一轮上（方块 96），则发送 MARKER 命令——发出请求使具有对应于当前时隙的时隙分配的卡作出响应的轮询命令（方块 98）。如果接收到有效响应（方块 100），则将 isCardfound 设定成真（方块 102）并保存响应（方块 104）。

一旦响应得到保存（方块 104），则使正在响应的卡转至活动或暂停状态（方块

106)。在图 4 中, 使卡转到活动或暂停状态的过程首先确定是否可获得一卡识别码 (“CID”) 或是否已经超过卡识别码的数量, 所述卡识别码 (“CID”) 分配给处于活动状态中的每一卡 (方块 58)。(在该实施例中, CID 的初始值设定成 “1”, 不过在其他实施例中可使用不同的值。在算法的开头还规定了可用的 CID 的最大数量。在该实施例中, 可用的 CID 的数量是 14; 在其他实施例中这一数量可发生改变。) 如果一 CID 仍然可用 (方块 58), 则将卡置于活动状态中 (通过发送 ATTRIB 命令) (方块 60), 并递增所用的 CID 数量 (方块 62)。如果不再有可用的 CID (方块 58), 则使卡转到暂停状态 (通过 HLTB 命令) (方块 64)。再次参考图 3, 一旦使卡转到活动或暂停状态 (方块 106), 则递增防冲突时隙的数量 (方块 108), 且如上文的方块 94 及下列等等中所述轮询下一时隙。

如果读卡器没有检测到任何清楚的响应 (方块 100), 则确定是否因冲突而未做出响应 (方块 110)。如果没有冲突 (方块 110), 且在算法的该迭代中此前没有检测到任何冲突 (换句话说, isCollisionFound 未设定成 “真”) (方块 112), 则将 isCollisionFound 设定成 “真” (方块 114)。如果 n 小于 4 (方块 116), 则递增 n (方块 118), 并递增时隙 (方块 108), 且按照上文的方块 94 以及下列等等中所述轮询下一时隙。

如果在算法的该迭代中之前已检测到冲突 (方块 112), 则递增时隙 (方块 108), 并按照上文的方块 94 以及下列等等中所述来轮询下一时隙。

如果 n 大于 4 (方块 116), 则递增时隙 (方块 108), 且按照方块 94 以及下列等等中所述来轮询下一时隙。

如果没有冲突 (方块 110), 则递增时隙 (方块 108), 且按照方块 94 以及下列等等中所述来轮询下一时隙。

如果时隙不小于 2^n (方块 94), 且 isCardFound 与 isCollisionFound 中的任一者为 “真” (方块 120) —— 表明检测到一卡或冲突, 则跟踪这一信息的变量 (即 isCardFound 和 isCollisionFound) 被设定为 “假” (方块 82), 且如上文及下文在方块 84 以及下列等等中所述来运行算法的另一迭代。

如果时隙不小于 2^n (方块 94), 则 isCardFound 和 isCollisionFound 均不是 “真” (方块 120), 返回所发现的卡的列表 (方块 122), 且算法结束 (方块 124)。当在使用 WUPB、REQB 和 MARKER 命令检查每一时隙之后没有发现卡或冲突的情况下, 该算法完成。

下文列出了算法的一实施例的伪代码实施方式。下文是函数说明, 为五个主要的防冲突命令解释了输入和输出:

```
(byte[], int) sendReqbCmd( byte n, byte afi );
(byte[], int) sendWupbCmd( byte n, byte afi );
(byte[], int) sendSlotMarkerCmd( byte slot );
(byte[], int) sendAttribCmd( byte[] pupi, byte cid );
(byte[], int) sendHltbCmd( byte[] pupi )
```

上文所列示的命令具有以下输入和输出。

输入:

byte n—n 指定 “N” 的值（防冲突时隙的数量）

$N = 2^n$ 且 $0 < n \leq 4$

byte afi—afi 指定用于将 PICC 的子集定为应用族标识符

byte slot—时隙规定 “S” 的值，即当前的时隙

$S = \text{时隙} + 1$ 且 $1 \leq S \leq N - 1$

byte[] pupi—pupi 是一四字节数组，其包含唯一的伪唯一 PICC 标识码 (“PUPI”)。

PUPI 可位于在 REQB、WUPB、或 MARKER 命令之后接收到的 ATQB 响应中

byte cid—cid 是 CID 的分配，其用于在活动状态命令期间处理具体的卡

$CID = \text{cid}$, $1 \leq \text{cid} \leq 14$

输出:

byte []—包含来自 PICC 的任何响应的一数组

int—可具有下列值的一整数

>0—从 PICC 返回的数据数组的长度

ERR_NO_RESPONSE—来自 PICC 的 Error No Response (无响应错误)

ERR_Collision—在 PICC 之间的 Error Collision (冲突错误)

查找卡的算法递归式地自我调用。以下是算法界面的一解释:

```
(rf1443card[], int listRfCards(
    byte wupbStatus = WUPB_FIRST_ITER,
    byte stateChangeStatus = MAKE_CARDS_ACTIVE,
    byte afi = 0x00,
    byte n = 0x00,
    byte cid = 0x01
    boolean isCardFound = true
    boolean isCollisionFound = true );
```

以下是界面的输入和输出。

输入:

byte wupbStatus—2 个可能的值

WUPB-FIRST_ITER—在算法的第一迭代期间发送 WUPB，以在编辑处于读卡器场中的卡的列表之前首先“唤醒”最初处于暂停状态中的卡。

WUPB-OFF—当使用该输入时，使用 REQB 命令以进行轮询。

byte stateChangeStatus—2 个可能的值

MAKE_CARDS_ACTIVE—会将前 14 个所发现的卡置于活动状态中，然后使随后发现的卡转到暂停状态，但在所述列表中计及。

MAKE-CARDS_HALTED—使所发现的卡直接转到暂停状态中。

byte afi—由某些应用程序使用的 AFI，以仅使所轮询的卡的一子集响应于防冲突

命令。

byte n—n 的初始大小可设定为 0x00，此意味着 N = 1；如果在算法转时需要，则可增大 n。

byte cid—这是下一个处于活动状态中的可用的 CID。最初该值可设定为 0x01。

boolean isCardFound—真或假，在该算法的上一迭代中是否发现卡？在调用时，该值应为真。

boolean isCollisionFound—真或假，在该算法的上一迭代中是否发现冲突？在调用时，该值应为真。

输出：

Rf1443card[]—ISO 14443 RF 智能卡对象的数组，所有 ISO 14443 RF 智能卡对象均标识其所处的状态、其 PUPID 值、及 CID 值（如果有）。

int—从该方法还返回卡数组中的条目的数量。

算法的以下实例性形式假设当算法开始时没有任何卡或 PICC 处于活动状态。这并不难实施，因为活动状态命令集通常包含一反选命令，以使 PICC 或卡转到暂停状态中。因为在该实施例中仅允许有 14 个不同的 CID，为每一 CID 执行反选命令可清除活动状态。

```
(rf14443card[], int) listRfCards( byte wupbStatus,
    byte stateChangeStatus, byte afi, byte n, byte cid,
    Boolean isCardFound, Boolean isCollisionFound )

{
    // Number of Cards Found
    int numCards = 0;
    // Stores the Cards discovered this iteration
    rf14443card iterCards[16];
    // Stores all the cards from this and recursive
    // iterations (Size is determined later)
    rf14443card[] retRfCards;
    // RF Card Response Variables
    int responseLength;
    byte[] response;
```

```
// Number of timeslots available this iteration
byte numSlots = (0x01 << N); // 2 ^ n

// Stopping Condition No Card and No Collisions
// Discovered
if (isCardFound || isCollisionFound)
{
    isCardFound = false;
    isCollisionFound = false;

    // For a first iteration send WUPB, otherwise
    // REQB
    if (wupbStatus == WUPB_FIRST_ITER) {
        (response, responseLength) =
            sendWupbCmd(n, afi);
    } else {
        (response, responseLength) =
            sendReqbCmd(n, afi);
    }

    // Loop Through the timeslots
    for (byte slot = 0x00; slot < numSlots; slot++) {
        if (slot > 0x00) {
            (response, responseLength) =
                sendSlotMarkerCmd(slot);
        }
        // Check for an ATQB Response from the
        // REQB/WUPB/Marker
        if (responseLength > 0) {
            isCardFound = true;

            // Get card information from the response
            iterCards[numCards].parseAtqb(response);
        }
    }
}
```

```
// Attempt to move card to Active State
if (stateChangeStatus ==
    MAKE_CARDS_ACTIVE) {
    (response, responseLength) =
        sendAttribCmd(
            iterCards [numCards].getPupi
                (), cid);
    // Successful Response
    if (responseLength > 0) {
        cid++;
        iterCards
            (numCards).setCid
                (cid);
    }

    // Run out of Card ID Numbers
    if (cid >= 0x0F) {
        stateChangeStatus =
            MAKE_CARDS_HALTED;
    }

    // Attempt to move card to .
    // Halted State
} else {
    (response,
        responseLength) =
        sendHltbCmd(
            iterCards
                [numCards].getPup
                    i());
}

// Successful Response from ATTRIB or
// HLTB commands
```

```
        if (responseLength > 0) {
            numCards++;
        }

        // Card wasn't detected successfully,
        // check for the first collision since last
        // WUPB/REQB
    } else if (responseLength == ERR_COLLISION &&
               !isCollisionFound) {

        isCollisionFound = true;

        if (n < 4) { // n does not exceed 4
            n++;
        }
    } // end-forloop(slot)

    // Recursive Call
    rf14443card[] saved RfCards;
    int numSaved RfCards;
    (savedRfCards, numSavedRfCards) =
        listRfCards(WUPB_OFF, stateChangeStatus, afi,
                     n, cid, cardFound, collisionFound);
    // {Insert Code
    // Store all of the cards in both iterCards[] and
    // savedRfCards[]
    // into an array of length numSavedRfCards +
    // numCards called
    // retRfCards (declared at the beginning of the
    // method)
    // End Insert Code}

    numCards += numSavedRfCards;
}

return (retRfCards, numCards);
}
```

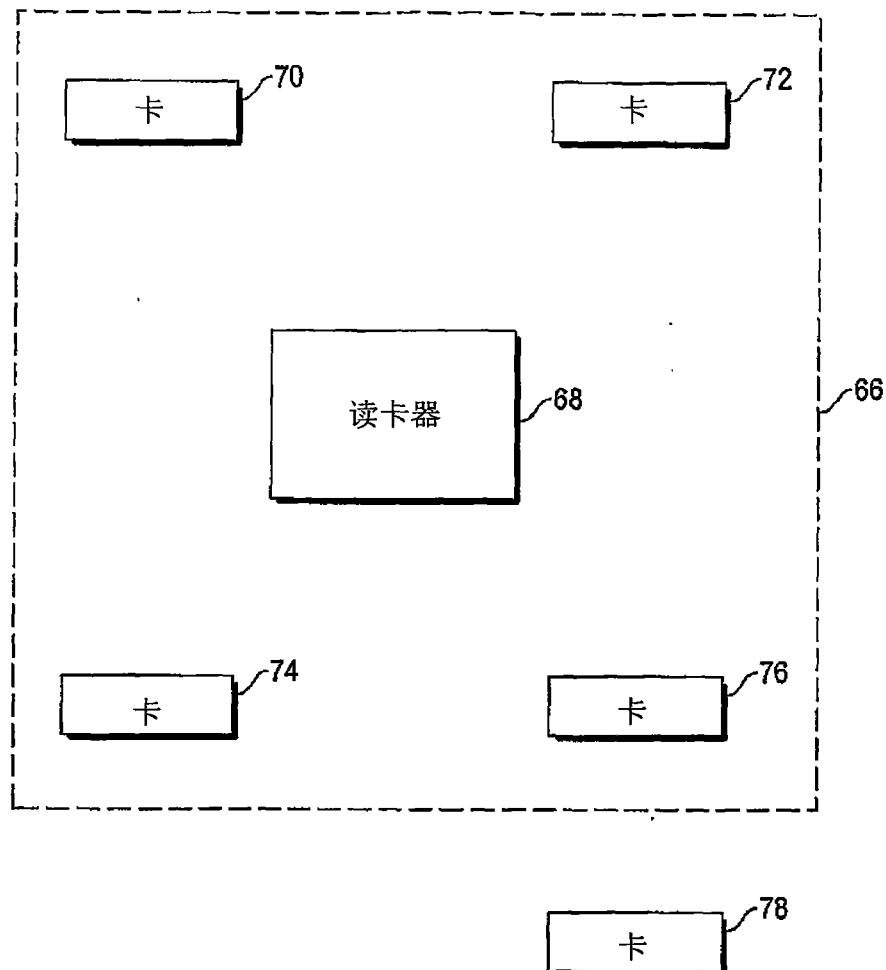


图 1 (现有技术)

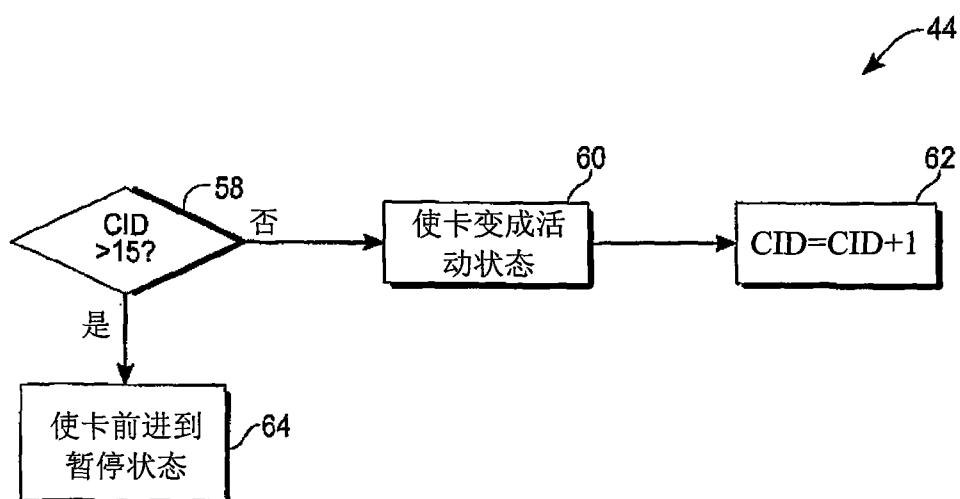
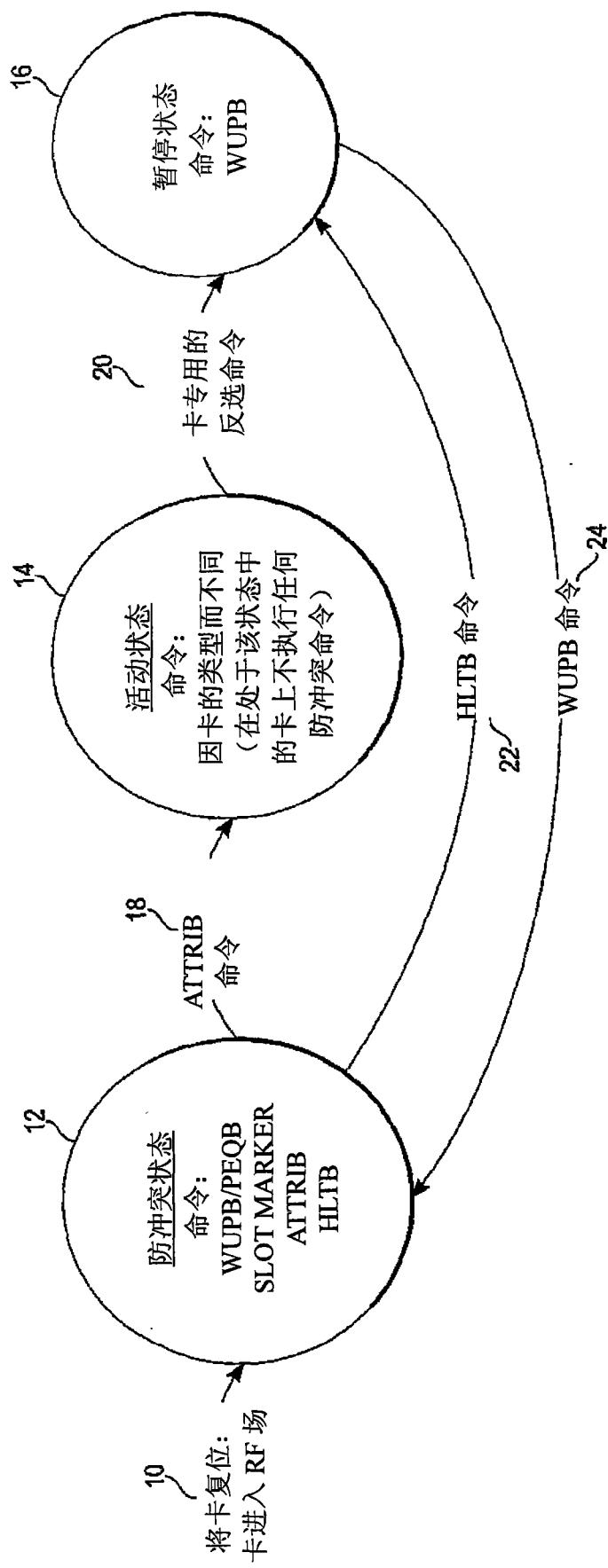


图 4



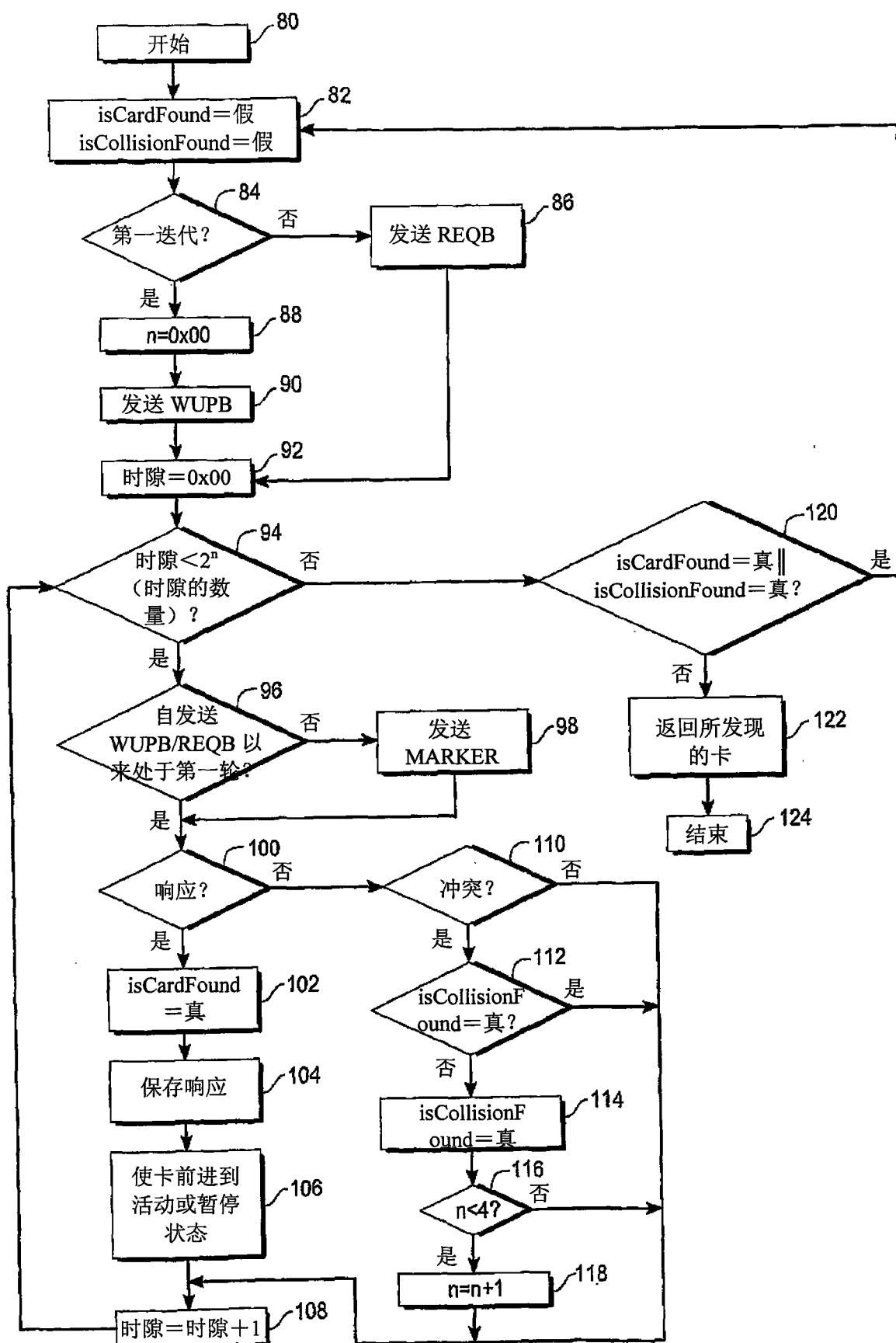


图 3