第36卷 第20期 计 算 机 工 程 2010年10月
Vol.36 No.20 Computer Engineering October 2010
·开发研究与设计技术· 文章编号：1000—3428(2010)20—0241—03 文献标识码：A 中图分类号：TP391

不定长RFID标签反碰撞识别算法

李 慧，张治国
(中山大学信息科学与技术学院，广州 510275)

摘 要：射频识别技术中存在标签冲突问题，通常采用定长标签的反碰撞识别方法来解决。针对不定长标签的反碰撞识别，设计一种对RFID 标签进行识别的高效二进制树形反碰撞算法——U-算法。模拟实现的测试与分析结果表明，U-算法具有二进制树形算法的优点，能达到41.67%的系统识别效率。

关键词：射频识别；不定长标签；反碰撞算法；二进制树

Anti-collision Identification Algorithm for Unfixed-length RFID Tags

LI Hui, ZHANG Zhi-guo
(School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510275, China)

【**Abstract**】Tag collision problem is a key issue in Radio Frequency Identification(RFID) system. The efficient algorithms for solving this problem usually deal with fix-length of tag encoding. For solving this problem with unfixed-length of tag encoding, a fast anti-collision algorithm called U-algorithm is presented. The U-algorithm is also of the advantages of binary tree algorithms. Simulation implementation experiments show that the proposed algorithm has the efficient identification ratio of 41.67%.

【**Key words**】Radio Frequency Identification(RFID); unfixed-length tags; anti-collision algorithm; binary tree

1 概述

射频识别(Radio Frequency Identification, RFID)是一种非接触式的自动识别技术。它利用射频信号及其空间耦合和传输特性，实现对静止或移动物品的自动识别[1-3]。现已广泛应用于工业自动化、商业自动化、交通运输控制管理、产品证件防伪和防盗等众多领域。RFID系统主要包括读写器和标签2个部分[4]。如果读写器的作用范围内有多个标签，当2个或2个以上的标签同时返回信息给读写器时，就会引起信号冲突，使读写器不能正确识别标签，这种现象称为标签冲突[5]。解决冲突的算法称为反碰撞算法。反碰撞算法对于系统的运行和可靠性有着非常大的影响，是RFID的关键技术之一。在现有的RFID系统中，反碰撞算法主要包括频分多路(FDMA)、空分多路(SDMA)和时分多路(TDMA)三大类[4-5]。其中，时分多路在ISO标准中占主导地位，它包括ALOHA算法和二进制树形算法等反碰撞算法[3]。

然而现存算法都是对定长的标签进行识别。目前，由于标准不统一，使得标签的编码方式和长度都不同，另一方面也为了充分利用编码，开发不定长RFID编码标签成为一种趋势。因此，本文提出一种对不定长RFID标签进行反碰撞识别的二进制搜索算法——U-算法。该算法充分利用了二进制算法的优点，不但可以达到与搜索定长二进制标签编码相当的效率，还能实现对不定长二进制标签编码的反碰撞识别。

2 时分多路反碰撞算法

2.1 ALOHA算法

ALOHA算法属于不确定算法[3]，其标签利用随机时间响应读写器的命令。此算法主要有ALOHA算法、时隙ALOHA算法、帧时隙ALOHA算法、动态时隙ALOHA算法。其中，系统识别效率最高的是动态帧时隙ALOHA算法，其系统识别效率可保持在34.6%~36.8%之间。在ALOHA算法中，虽然系统设计比较简单，但是这种算法存在无法识别某个或某些标签的可能。

2.2 二进制树算法

二进制算法的系统识别效率比较高，不存在错误判断，但系统的设计比较复杂[3]。其基本思想是将处于冲突的标签分成左右2个子集0和1。先查询子集0，若没有冲突，则正确识别标签；若仍有冲突则再分裂，把子集0分成00和01两个子集，以此类推，直到识别出子集0中所有的标签，再按此步骤查询子集1。一次查询中实现读写器发送请求和标签响应读写器请求并返回数据的过程。二进制树算法主要有随机二进制树算法、查询树算法、前缀查询树算法、二进制搜索算法、跳跃式动态树形算法等。系统识别效率较高的是前缀查询算法和跳跃式动态树形算法，它们的系统识别效率分别可以达到43%和50%。

3 U-算法原理

U-算法是一种基于二进制树算法的搜索算法。为了能准确辨认出读写器中数据碰撞的比特位置，本文采用Manchester编码。其特点[4]是在位持续时间内，位值由电平的改变来表示。在半个比特周期时的负边沿表示二进制数1，这时前半个比特周期含有副载波信号；在半个比特周期时的

基金项目：中山大学实验教学研究基金资助项目
作者简介：李 慧(1987－)，女，硕士研究生，主研方向：RFID，模型验证；张治国，副教授
收稿日期：2010-05-25 **E-mail：**lihui161@126.com

正边沿表示二进制数 0，这时后半个比特周期含有副载波信号。当 2 个或 2 个以上标签同时返回到终端的编码有不同值时，就将正负边沿抵消了。在整个比特周期中，终端得到的是不间断的副载波信号，这样的波形信号将不能识别为 0 或 1，终端认为在该数据位上发生了碰撞。

为了便于 U-算法描述，本文定义了 3 个命令：*call*(*epc*, *m*)，*sleep*(*epc*, *m*)，*request*(*epc*)。

(1)*call*(*epc*, *m*)[4]的含义为：读写器向覆盖范围内的标签发送 *call* 命令，如果标签 EPC 码的前 *m* 位与 *call* 命令中的 *epc* 参数相同，那么满足条件的 EPC 标签做出应答，向读写器发送 EPC 码，不满足条件的 EPC 标签则处于空闲状态不做应答。这样做可以有效地缩小选择范围。

(2)*request*(*epc*)命令的含义与 *call* 命令类似，即读写器向覆盖范围内的标签发送 *request* 命令，如果标签 EPC 码与 *request* 命令中的 *epc* 参数相同，则做出应答；否则标签处于空闲状态，不做应答。

(3)*sleep*(*epc*, *m*)[4]命令的含义为：读写器向覆盖范围内的标签发送 *sleep* 命令，如果标签 EPC 码与 *sleep* 命令中的 *epc* 码前 *m* 位相同，则使此 EPC 标签进入睡眠状态，再次收到 *call* 指令不做应答。要想重新激活睡眠状态的标签，必须使其脱离读写器的作用范围。

下面以一个实例来直观地描述 U-算法的原理。假设在某时刻有 7 个标签同时进入读写器的作用范围，它们的 EPC 编码分别为 EPC1=1、EPC2=01、EPC3=010、EPC4=1011、EPC5=01001、EPC6=010110、EPC7=1001010。算法的执行过程如图 1 所示。

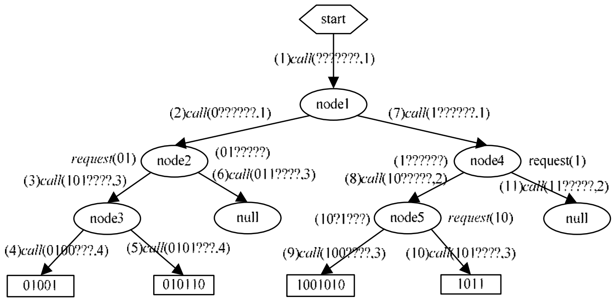

图 1　U-算法执行过程

程序的执行顺序如下：

(1)发送 call(???????,0)命令，所有标签均做出应答，混叠的数据返回给读写器，此时读写器识别为(??????)。然后从最高位计算到产生冲突的前一位，得到 *epc* 参数为 null，形成 *request* 的 *epc* 参数。发送 *request*()，没有标签响应。接下来算法将最高冲突位置 0，由此计算出 *m* 参数的值，形成下一次 *call* 命令。并将最高冲突位置 1，保存在用于存储树形节点信息的栈中。

(2)发送 *call*(0??????,1)命令，EPC2、EPC3、EPC5、EPC6 均做出应答，混叠的数据返回给读写器，此时读写器识别为(01????)。然后从最高位计算到产生冲突的前一位，得到 *epc* 参数 01，形成 *request* 的 *epc* 参数。发送 *request*(01)，此时只有 EPC2 满足条件，读写器收到的信息仅由 EPC2 发出，不存在碰撞问题。读写器将 EPC2 的 EPC 码 01 存入 RAM 中等候处理。然后发送 *sleep*(01,2)命令，强制 EPC2 进入睡眠状态。接下来算法将最高冲突位置 0，由此计算出 *m* 参数的值，形成下一次 *call* 命令。并将最高冲突位置 1，保存在用于存储树形节点信息的栈中。

(3)发送 *call*(010????, 3)命令，此时 EPC3、EPC5、EPC6 均做出应答。读写器接下来重复类似(1)中的操作，形成 *request* 命令中的 *epc* 参数 010，发送 *request*(010)，此时只有 EPC3 满足条件，读写器收到的信息仅由 EPC3 发出，不存在冲突问题。读写器将 EPC3 的 EPC 码 010 存入 RAM 中等候处理。然后发送 *sleep*(010,3)，强制 EPC3 进入睡眠状态。接下来将 *epc* 参数最高冲突位置 0，并由此计算出 *m*，形成下一次 *call* 命令的参数 *epc*=0100???，*m*=4，为下一次 *call* 命令做准备。并将最高冲突位置 1，存入栈中。

(4)发送 *call*(0100???, 4)命令，此时只有 EPC5 满足 *call* 命令条件，不存在冲突问题。读写器收到的信息仅由 EPC5 发出，读写器将 EPC5 的 EPC 码 01001 存入 RAM 中等候处理。然后，发送 *sleep*(01001,5)命令，强制 EPC5 进入睡眠状态。最后算法读出栈中的节点信息，*epc*=0101???，计算出 *m*=4，形成下一次 *call* 命令的参数。

(5)发送 *call*(0101???, 4)命令，此时只有 EPC6 满足条件，不存在冲突。读写器将 EPC6 的 EPC 码 010110 存入 RAM 中等候处理。然后，发送 *sleep*(010110,6)命令，强制 EPC6 进入睡眠状态。最后算法读出栈中的节点信息，*epc*=011????，并计算出 *m*=3，发送 *call*(011????, 3)命令，此时读写器没有收到任何标签发送过来的 EPC 编码信息，表明识别为空。然后，算法读取栈中的节点信息，形成下一次 *call* 命令的参数，*epc*=1??????，*m*=1。为下一次 *call* 命令做准备。

(6)重复上面的步骤，直到发送 *call*(11?????, 2)，没有满足条件的标签做出应答，读写器识别为空。此时栈为空，说明算法已经遍历了二进制树的每一个节点。至此，整个算法执行完毕，EPC1、EPC2、EPC3、EPC4、EPC5、EPC6、EPC7 的 EPC 码已经被识别出来，并保存在 RAM 中，以便于读写器进行下一步操作。

4　U-算法的效率分析

对于一个有 N 个叶子的二进制树，它的分支节点数目为 $N-1$，总的分支数即为 $2N-2$，再加上主干后数目为 $2N-1$。对于本算法，二进制树的分支节点数即为读写器发送 *request* 命令的次数，二进制树干与分支的数目总和即为读写器发送 *call* 命令的次数。因此，发送 *request* 命令和 *call* 命令的总和为 $3N-2$。左侧 $N/2$ 个叶节点一定是被识别出来的标签，右侧叶节点和中间节点则有可能是被识别出来的标签，假设概率都是 50%。那么识别出标签数目的期望如式(1)所示：

$$M=\frac{N}{2}+(0+1+2+\cdots+\frac{3N-2}{2})\times\frac{2}{3N}=\frac{5N-2}{4} \tag{1}$$

其中，M 为识别出标签数目的期望；N 为二进制树叶的个数。

分辨 $\frac{5N-2}{4}$ 个标签共需要 $3N-2$ 个 *request* 命令和 *call* 命令时隙。算法的效率如式(2)所示：

$$\eta=\frac{5N-2}{4(3N-2)}\times 100\% \tag{2}$$

其中，η 为不定长 RFID 标签识别的效率；N 仍是二进制树叶的个数。当 N 趋近于无穷时算法的效率趋近 41.67%。

5　测试与分析

由程序产生一些随机数进行测试，取标签最大编码长度为 7，并且保持不变，此时最多能取 128 个不同的二进制数。使标签的数目(*Num*)从 1 个变化到 127 个，得到发送 *call* 命令

的次数(CN)、*request* 的命令的次数(RN)，从而可得到标签的响应效率(通过(CN+RN)/Num 计算得到)。同法可得到标签最大编码长度为 6 和 8 时标签响应效率的测试结果，将它们绘到同一个图形上，以方便比较和观察。当标签最大编码长度保持不变时标签个数与响应效率的关系如图 2 所示。

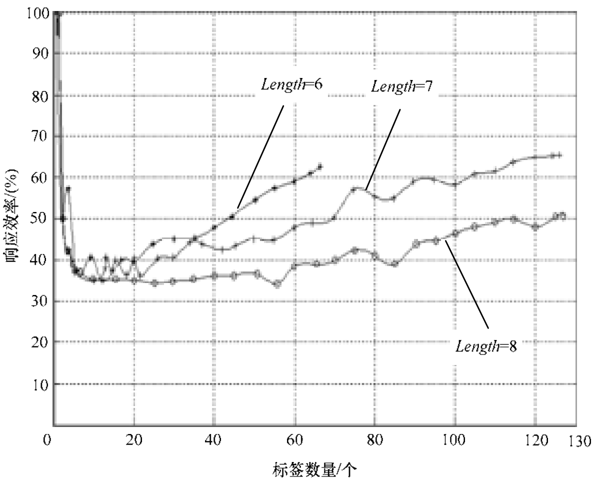

图 2　标签数量与响应效率的关系

从图 2 可以观察到，当固定标签最大编码长度不变时，刚开始时标签的识别效率为 100%(因为只有一个标签，所以只需发送一次 *call* 命令)，然后识别效率迅速降到一个比较低的值，随着标签数量的增多，标签的效率逐渐升高，当标签数量达到标签最大长度所能表示的数量范围时，标签的识别效率达到最大，标签最大长度越短，识别效率增加速度越快。因为随着标签数量的增加，响应 *call* 命令和 *request* 命令的标签数量也相对增加，没有标签响应的 *call* 命令和 *request* 命令相对减少，当标签数量达到或接近标签最大长度所能表示的数量范围时，几乎所有的 *call* 命令和 *request* 命令都有标签响应。例如对于 N 个标签，假设每个 *call* 命令和 *request* 命令都有标签响应。则读写器发送 *request* 命令的次数为 $N/2$ 次，发送 *call* 命令 $N+1$ 次。此时，标签的识别效率为 $\frac{2N}{3N+2}\times 100\%$，当 N 逐渐增大时，算法的效率趋近于 66.7%。而这与上文分析的 U-算法的效率为 41.67%并不矛盾，因为上面的情况是在一般情况下。从图中可以看出，识别效率的值大部分情况下在 41.67%左右摆动。

固定标签数量不变，取标签的数量为 40，使标签的长度从 6 变化到 15，来测试标签的效应效率。同法可得到标签为 30 和 50 的测试结果。将它们绘制在同一个图中以方便对比与观察。标签数量保持不变时，标签编码长度与响应效率之间的关系如图 3 所示。

从图 3 可以看出，当固定标签数量不变时，随着标签最大长度的增加，即标签最大长度所能表示的数量范围增加时，标签的识别效率不断降低，最后标签的识别效率几乎稳定在一个值。因为随着标签最大长度的增加，所能表示的数的范围不断增加，在产生的随机数中一些标签是其他标签前缀的概率就减小，所以 *request* 命令识别标签的概率也相对减小。假设有 N 个标签，没有任何一个标签是其他标签的前缀，识别这 N 个标签需要 N 次 *request* 命令，$2N+1$ 次 *call* 命令，此时标签的识别效率为 $\frac{N}{3N+1}\times 100\%$，当 N 逐渐增大时，识别效率趋近于 33.3%。

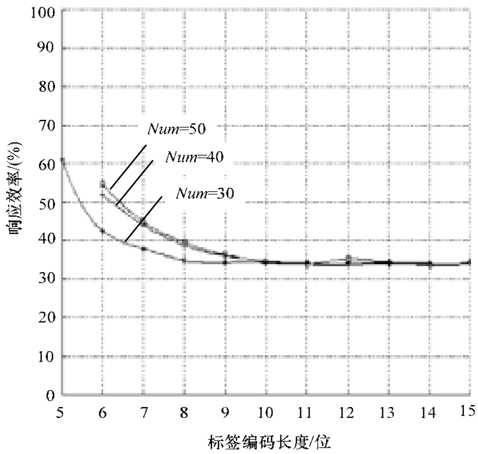

图 3　标签编码长度与响应效率之间的关系

从上面的测试和分析可以得出结论，在读写器作用范围内标签比较多，或有些标签是其他标签前缀的不定长标签编码的 RFID 系统中，该算法识别效率比较高。在最佳情况下(除读写器作用范围内仅有一个标签的情况)，标签的识别效率可以达到 66.7%，而在最坏情况下标签的识别效率为 33.3%。一般情况下标签的识别效率为 41.67%。

6　结束语

本文讨论了 RFID 系统中的反碰撞算法，在充分利用二进制树算法的优点基础上，提出一种对不定长标签反碰撞识别的 U-算法。理论分析与模拟测试结果表明，U-算法在一般情况下可达到 41.67%的系统识别效率，这与在定长 RFID 识别系统中前缀查询树算法 43%的系统识别效率接近。

参考文献

[1] 余松森，詹宜巨，王志平. 跳跃式动态树形反碰撞算法及其分析[J]. 计算机工程, 2005, 31(9): 19-20.

[2] Myung J, Lee W, Srivastava J. Adaptive Binary Splitting for Efficient RFID Tag Anti-collision[J]. IEEE Communications Letters, 2006, 10(3): 144-146.

[3] 赵　曦，张有光. 一种新颖的 RFID 多标签反碰撞算法[J]. 北京航空航天大学学报, 2008, 34(3): 276-279.

[4] 余松森，詹宜巨，彭卫东，等. 基于后退式索引的二进制树形搜索反碰撞算法及其实现[J]. 计算机工程与应用，2004，40(16): 26-28.

[5] 韩　磊，张　虹，马海波. 散列树形搜索反碰撞算法的研究[J]. 计算机应用, 2006, 26(12): 3019-3022.

编辑　顾姣健