# 基于四叉树的高速乘法器算法研究

刘 磊, 严晓浪, 孟建熠, 葛海通
(浙江大学 超大规模集成电路设计研究所, 杭州 310027)

**摘 要**: 提出了一种基于四叉树结构的高速乘法器自动综合优化算法以提升乘法器运算速度。首先对延时较大的高位积采用四叉树递归直接构建,取代传统部分积进位链,缩短关键路径时延,进而进行分支折合和合并,相邻乘法结果共享部分四叉树,降低硬件开销。算法同时支持不同面积约束下的自动综合。依此算法的乘法器相比基于 Booth 算法和 Wallace 树的乘法器速度提高了 10%。

**关键词**: 进位链; 延迟; 四叉树; 分支合并; 分支折合; 遍历
**中图分类号**: TP309 **文献标志码**: A **文章编号**: 1001-3695(2010)10-3727-04
doi:10.3969/j.issn.1001-3695.2010.10.032

# High speed multiplier algorithm based on four-tree structure

LIU Lei, YAN Xiao-lang, MENG Jian-yi, GE Hai-tong
(*Institute of VLSI Design*, *Zhejiang University*, *Hangzhou* 310027, *China*)

**Abstract**: This paper used an auto-synthesizing optimized arithmetic for speed up multiplier velocity. It used four-tree recursive method to reduce critical path, instead of traditional part product carry-in chain to generate high part of long-latency multiplier. Then it reduced hardware cost by branch breaking and combing to implement logic share between tree structures. Also it supported auto-synthesize within different area constraints. This new multiplier gets 10% faster than the one based on Booth arithmetic and Wallace tree.

**Key words**: carry chain; delay; four-tree; branch combining; branch breaking; tree traversal

## 0 引言

乘法器是处理器进行科学计算和数字信号处理的基本硬件结构。由于传统乘法运算基于部分积累加,随着现代处理器的运算宽度不断提升,部分积累加进位链产生的运算延时也日益增加,使其成为制约处理器运行速度的瓶颈之一。高速乘法器设计通常分为三个关键步骤:部分积产生、部分积累加和最终结果获得。目前对部分积产生过程普遍采用改进 Booth 算法[1,2],有效减少部分积加法项;对部分积的累加通常采取 Wallace 树[3]阵列进行深度压缩;而最终结果的获得则以一个根据部分积累加结果到达时间的不同进行延迟优化的加法器将累加结果和累加进位相加而得。部分积的产生算法相对成熟,近期研究主要集中在部分积累加和最终结果获得上。文献[4]提出了一种通过分析 Booth 乘法阵列延迟结构,先行计算最后一行的加一加法,因此删减了这一级的进位,将阵列深度减少 1。文献[5]采用一种更为高效的 4-2 压缩器进行部分积累加,以进位选择和超前进位相结合的加法器实现最终结果获得。文献[6]则采用 CSMA(conditional-sum adder)和 CCA(conditional-carry adder)相结合的最终加法器达到优化效果。从本质上看,这些算法以阵列相加为核心思想,因而不论何种改进,其进位链始终由阵列低位向高位传播,位数越高,进位链越长,由此造成积的延迟由低向高逐渐增加[3]。这是基于阵列加和原则的乘法器所不可避免的延迟方式。

分析传统乘法器部分积的累加特性,笔者发现对于长延时的乘法高位结果,仅需通过乘数高位与部分进位信息就可以准确求值,而无须采用传统的完整进位链。因此可以通过一种新型的乘法结果高位产生方法,降低乘法高位结果产生的运算延时。基于上述基本思想,本文提出了一种采用四叉树结构综合乘法高位结果,降低高位结果运算延迟的方法,消除传统阵列结构中长进位链对乘法器性能的影响。本文进一步提出了利用链表结构生成乘法树,并通过分支折合和分支合并降低硬件开销的方法。经综合工具分析,以增加部分面积为代价,将最长路径控制在积的中部,相比传统乘法器降低时延约 10%。

## 1 乘积快速求值的基本算法

传统 $n\times n$ 阵列乘法器,乘数之间交叉位与操作共产生 $n^2$ 个部分积参与阵列加和运算。积的第 $i$ 位表达式由递归式得出

$$\{Co_i, P_i\} = \begin{cases} \sum_{m=i-n+1}^{n-1} x_{m,(i-m)} + Co_{i-1} & i \geqslant n \\ \sum_{m=0}^{i} x_{m,(i-m)} + Co_{i-1} & i < n \end{cases} \quad (1)$$

表现为本位和(式中求和项)与低位进位之和。$Co$ 为进位,$P$ 为积 $i$ 位值,$x$ 为部分积集合。$P_i$ 的值由 $Co_0 \sim Co_{i-1}$ 逐级递归式传递得到。因此,阵列乘法器的延迟主要是由低向高传播的进位链决定,位数越高,延迟越大。

**收稿日期**: 2010-04-06; **修回日期**: 2010-05-23

**作者简介**: 刘磊(1986-),男,陕西渭南人,硕士研究生,主要研究方向为嵌入式处理器设计(liulei@vlsi.zju.edu.cn);严晓浪(1947-),男,教授,主要研究方向为超大规模集成电路设计、VLSI 设计自动化;孟建熠(1982-),男,博士,主要研究方向为高性能低功耗嵌入式处理器设计与研究;葛海通(1972-),男,博士,主要研究方向为嵌入式系统设计.

注意到 $n^2$ 个部分积之间存在的紧密联系,当乘数的某一位发生变化,会导致 $n$ 个部分积均随之改变,使得阵列加法的某一行或者某一列整体发生变化。由于这种改变使本位和仅有一位发生改变,而进位从 $Co_0 \sim Co_{i-1}$ 均有改变,那么本位和对积的影响将更为明显,因为进位是逐级传递的,其改变会在传递过程中被高位信息屏蔽或者相互抵消。观察式(1),当 $i$ 超过 $n$,本位和的输入逐渐减少,而进位的逻辑复杂度增强,那么由乘数低位的改变导致积的变化也就越来越迟钝,因而乘数可能需要较大范围的变化才会引起积高位的改变。这也就解释了多位数乘法乘数高位变化更容易引起积高位改变的现象。

由此得出乘积高位的特点:求值过程可以不依赖完整的进位信息,而主要由本位和以及部分关键的进位点所决定。因此,只需找到决定高位值的这几个关键位,就可以准确得到计算结果,不必通过复杂的进位链。

根据这个特点,本文算法思路是先处理对结果最重要的高位,然后采用由高向低逐级判断进行求值。积第 $i$ 位表达式为

$$P_i = N00_{n-1}F00_{n-1} \parallel N01_{n-1}F01_{n-1} \parallel N10_{n-1}F10_{n-1} \parallel N11_{n-1}F11_{n-1}, i \geqslant n$$

$$NKL_j = (a_j == K)\&(b_j == L), K, L \in \{0,1\} \quad (2)$$

$FKL_{n-1}$ 为 $NKL_{n-1}$ 成立时使 $P_i$ 为 1 的函数,可以展开为 $FKL_{n-2}$ 的函数。进一步推导,$FKL_j$ 为 $\&NKL_m (m = n-1 \rightarrow j)$ 成立时使 $P_i$ 为 1 的函数。$FKL_j(a_{j-1}, \cdots, a_0, b_{j-1}, \cdots, b_0)$ 依赖于如下递归关系:

$$FKL_j = N00_{j-1}F00_{j-1} \parallel N01_{j-1}F01_{j-1} \parallel N10_{j-1}F10_{j-1} \parallel N11_{j-1}F11_{j-1}, 0 \leqslant j < i$$

$$FKL_0 = [(\sum_{p=1}^{n-1} K_p 2^p + K_0) \times (\sum_{q=1}^{n-1} L_q 2^q + L_0)]\&2^i ? \ 1ń \ 0 \quad (3)$$

这样 $n$ 位乘法最终出现的递归 $F0$ 项将会有 $4^n$ 之多。然而由于乘积高位的取值大多由乘数高位决定,绝大多数 $FKL_0$ 项都为常量。最终 $P_i$ 的表达式可化为简单的组合逻辑,并且位数越高其逻辑越简单。

乘法器随着位数的增加表达式的复杂度呈指数级增长,因此计算机自动设计就成为一个合理高效的方法。本文提出的四叉树结构乘法器算法即以上述推论为基础,通过计算机编程来自动分析产生简化的表达式。

## 2　四叉树结构的自动综合

算法根据式(3)展开,以乘数对应位的逻辑组合为原子操作,得到自高向低依次展开的树型结构来表达积的高位值。

一个 $n \times n$ 运算,两个乘数在对应位 $a[i]$ 和 $b[i]$ 上的值的组合只有四种:11、10、01、00。因此,使积的第 $j$ 位 mult$[j]$ $(j > n)$ 为 1 的一个表达式的值应当是 $a[i]$ 和 $b[i]$ $(i$ 从 $(n-1)$ 变化至 0)以上述值组合为原子的逻辑与关系,例如:

$$(\sum_{p=0}^{n-1} K_p 2^p \times \sum_{q=0}^{n-1} L_q 2^q)\&2^j == 1, j \geqslant n \quad (4)$$

式(4)表明积第 $j$ 位 mult$[j]$ 为 1 的条件为

$$\text{mult}[j] = \&[(a[p] == Kp)(a[p] == Lp)], p = 0 \rightarrow n-1 \quad (5)$$

使 mult$[j]$ 值为 1 的所有表达式的逻辑或关系,即为这一位的值表达式。由于乘数对应位的关系为固定的四种,采用四叉树型结构表示,具体原则如下:

a) mult$[j]$ 的树型表达自乘数高位向低位生长,每一个节点 $i$ 代表两个乘数第 $i$ 位的某一数值组合(11/10/01/00 )。

b) 节点 $i$ 最多包含 4 个子节点,即 $(i-1)$ 位四种可能的值组合;如果某种组合不能使 mult$[j]$ =1,则对应的子节点不存在。

c) 每个叶子代表一种使 mult$[j]$ 为 1 的条件,其表达式为从根节点到这个叶子的所有节点的与关系。

d) 所有叶子的逻辑表达式的或关系组成了 mult$[j]$ 完整的表达式。

以 3×3 为例描述这种四叉树型数据结构。积最高位的四种表达式如图 1 所示。对应的树型结构如图 2 所示。

mult[5]=1 的表达式
101×111=100011
110×111=101010
111×111=110001
110×110=100100

图 1　3×3 乘法积 5 位的值表达式

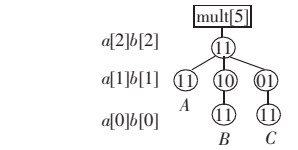


图 2　3×3 乘法积 5 位的树型结构表达

对四叉树自上而下遍历得到:

```
mult[5] = (a[2]&& b[2])&&(
(a[1]&& b[1]) ‖ //节点 A
(a[1]&&! b[1])&&(a[0]&& b[0]) ‖ //节点 B
(! a[1]&& b[1])&&(a[0]&& b[0]));//节点 C
```

树型结构的计算机生成即基于上述方法。本文采用链表结构,依递归函数进行树的生成[7]。以 $n \times n$ 最高位为例,伪代码实现如下:

```
struct node{node * child00/01/10/11};
global a[n] = {0};b[n] = {0};mult[2n-1] = a[n] * b[n];
treeGen(int opbit,tree node){//opbit 为正在操作的乘数位号
if(! opbit){//抵达最低位,递归结束
a[0] = p;b[0] = q;
if(mult[2n-1]) node -> childpq = 'y';
else node -> childpq = 'n';(pq = 00/01/10/11)}
else{
a[opbit] = p;b[opbit] = q;node -> childpq = new nodepq; treeGen
(opbit-1,nodepq);(pq = 00/01/10/11)}}
```

由于积的高位不需要全部进位信息的特点,这个生成树是很稀疏的,有很大的时延优化空间,这即是本算法可以改善高位时延的基础所在。

## 3　树的分支合并和折合优化

第 2 章生成树已经初步达到了时延优化的效果,然而因为依赖递归关系,没有体现树的相似特点,导致结构比较复杂,面积很大,而且信号负载重,直接影响了关键路径时延。所以需要对生成树进行同项比较,化简相近部分。其基本思想为树的相似度分析。

定义以下两种分支:

a) 完全分支。从分支主节点 $i$ 向下,每个节点都有四个子节点。完全分支表明 $i$ 位以下的取值对最终结果没有影响。

b) 不完全分支。完全分支的补集。

根据四叉树结构的定义,容易推出以下三种分支关系:

a) 相等。分支从其主节点向下所有的节点和叶子都完全相同。

b) 加和。几个分支(加数分支)相加得到一个新分支(和分支),加数分支的每一个节点都属于和分支,即和分支为所有加数分支的或关系。

c) 归约。如果一个分支第 $i$ 级以下的子分支都相等,而 $i$ 级以上为完全分支,那么这个完全分支逻辑可以忽略。

上述关系作为乘法树的逻辑化简理论基础。图 3 描述了

这几种分支关系。

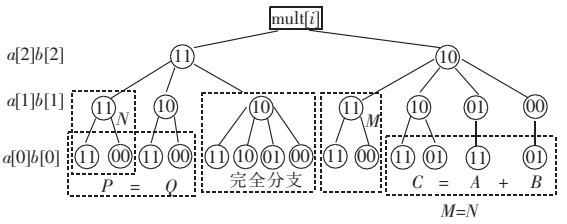


图 3　树型结构分支关系示意图

四叉树结构的化简步骤如下:

a)分析所有的完全分支并进行化简。

b)化简可归约分支。

c)对各分支进行相等性比较,合并处理相等分支。

d)对不处于关键路径上的分支进行加和,然后与其他分支进行相等比较,如相等则可以合并化简。

e)对于复杂分支,找到其公共组成分支来表示。例如分支 $A$ 可以拆分为已有分支 $A_1+A_2$,分支 $B$ 可以拆分为 $A_1+B_1$,则用 $A_1$、$A_2$、$B_1$ 三个已有子分支的加和就可以表示 $A$、$B$ 两分支。

f)首先对于每一位的树作内部化简,然后所有位的树作联合化简,已达到尽可能优的化简效果。

上述步骤用于简化树的逻辑表达式,降低硬件开销。以图 3 为例进行化简:化简完全分支,合并 $P$、$Q$ 两分支,$A$、$B$ 加和后作为 $C$ 的表达式。最终简化结果如图 4 所示。

对于处在关键路径上的分支,可以将其从中间折断,变为两个子分支的与关系,然后附加在原先的树上(或关系),这样将分支的时延关系从由低向高的传递转变为从高低两边向中部传递,代价是增加了这部分的分支面积。图 5 描述了延迟分支折合的化简方案。

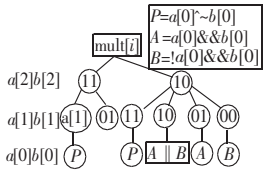


图 4　树型结构的分支合并化简

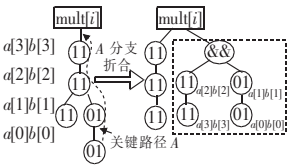


图 5　树型结构的延迟分支折合

分支的合并和折合很大程度上影响了最终的面积大小和时延优化。由于分支的关系可能出现很多种情况,一个不适当的合并或折合有可能会使时延变差,如何选择一种合理有效的分支相似度分析算法决定着最终面积时延的优化结果。

最后,本文设计采用深度遍历法进行表达式生成,伪代码实现如下:

```
struct node { node * child11/10/01/00 }
logicGen( int opbit,tree node,fstream * file) {
if( ! opbit ) {//抵达最低位,递归结束
switch( { node→child11/10/01/00 } ) {
case "yynn" : * file < < "a[0]"; break;
//y 表示这种组合使结果为 1
case "yyyn" : * file < < "a[0] ‖ b[0]"; break;
……}
} else {//递归主体
//child 11
if( node - > child11 = = 'y')
//y 表示结果已确定,不需要接下来的信息
 * file < < "a[opbit] && b[opbit]";
else if( node - > child11 ! = null) {
//有子节点,需要继续生成逻辑表达式
 * file < < "a[opbit] && b[opbit] && (";
logicGen( opbit - 1 , node - > child11, file);
 * file < < ")"; }
else;//无子节点,这种组合不可能使结果为 1
//child 10, 01, 00, 与上相同
……}}
```

## 4　实验结果与分析

采用 Design Complier (DC)在 SMIC13 工艺下对生成的乘法器进行综合实验,算法成功地将积高位延迟控制在中部延迟之下,使得整体延迟为中部高,高低位逐次下降,整体时延减少 10% ~15%。图 6 为积各位延迟的变化曲线。实线为四叉树型快速乘法器的延迟,虚线为 Booth 乘法器的延迟。

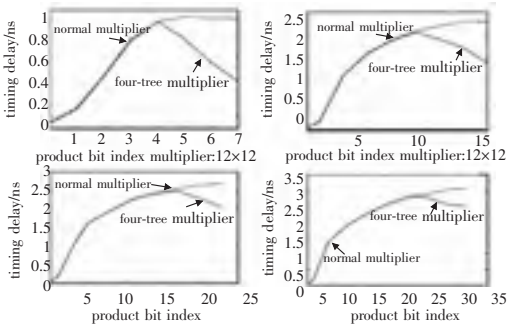


图 6　乘法器延迟结果比较

由于本算法对于积的高位时延优化比较明显,用于低位则会导致面积过大,可采用四叉树结构和阵列结构的混合体以控制由四叉树结构带来的面积代价,同时保证中部的时延基本不变。完整算法显示了三步优化方法:首先采用四叉树结构优化乘法器时延,牺牲部分面积;进而采用分支折合和合并优化树型结构冗余面积;最后采用折中体再次减少低位冗余面积。图 7 显示了 4 位乘法器经三步优化的效果。对比 Booth 乘法器,经四叉树优化时延降低 12%,面积增大 50%;通过分支折合和合并,时延降低至 15%,面积缩小 10%;最后可变比例的四叉树—阵列混合结构显示了进一步的面积时延折中(横坐标表示四叉树比特数—阵列比特数),即四叉树型结构位数越多,时延优化越好,同时面积逐渐增大。图中采用 3-5 组合的乘法器时延几乎不变,而面积仅为 Booth 乘法器的 1.2 倍。使用者还可根据不同需求选择其他的位数组合,达到最优的应用。

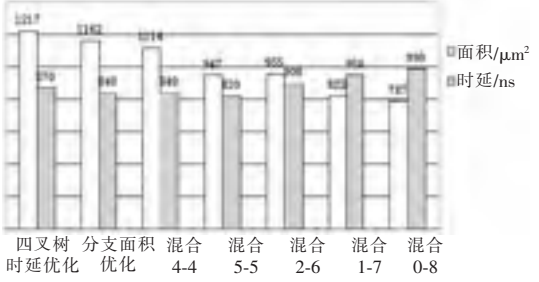


图 7　四叉树算法面积时延优化效果

最后给出经综合的完整 4×4 无符号乘法器逻辑图,灰色部分示意了部分合并的分支。可以清楚地看出,逻辑复杂度从中部向高位逐渐降低,作为典型的树型结构,这个乘法器也继承了乘法树诸如图形不规则、不易高质量布局布线等缺陷。综合结果时延 0.87 ns,面积 1 063 $\mu m^2$,相比 Booth 乘法器速度提高约 15%,硬件增加约 25%。

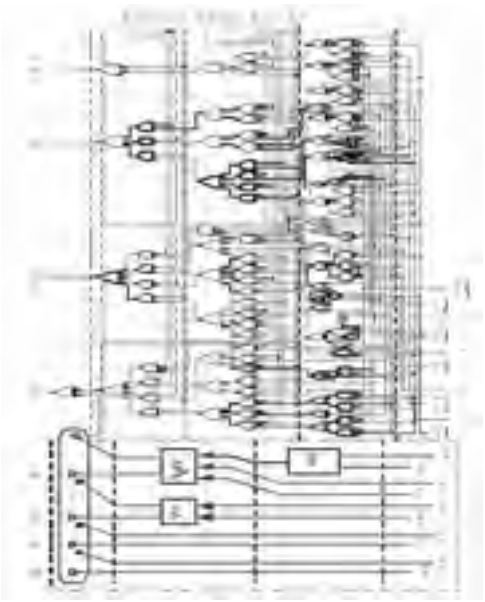

图 8　4×4 四叉树快速乘法器门级逻辑图

## 5　结束语

本文基于分析乘法器高位求值的特殊性,提出了一种用于降低乘法器高位延时的四叉树乘法器自动综合算法。该算法不同于传统乘法器的自低向高逐级进位链的计算方式,而是将乘积逻辑自乘数高位至低位分解,先行计算对结果影响最大的乘数高位运算部分,从而简化了积高位逻辑,进而降低整体延迟。本文采用 C++ 编程,利用链表结构生成乘法树,通过分支的折合和合并进行时延面积的优化,最后经遍历化分析产生逻辑表达式。经过 DC 工具综合,成功地将乘法器延迟控制在积的中部,速度提高 10% 以上。

从仿真结果来看,延迟整体上提高近 10%;而面积上,尽管采用分支合并有效控制了乘法器面积的增加,但未来还有一定的优化空间。合理高效的面积优化新算法是未来研究的主要方向。

**参考文献:**


[1] BOOTH A D. A signed binary multiplication technique[J]. Quarterly Journal of Mechanics and Applied Mathematics, 1951, 4(2): 236-240.

[2] MACSORLEY O L. High speed arithmetic in binary computers[J]. Proceedings of the IRE, 1961, 49(1): 67-69.

[3] WALLACE C S. A suggestion for a fast multiplier[J]. IEEE Trans on Electron Computers, 1964, 13(1): 14-17.

[4] KANG J Y, GAUDIOT J L. A fast and well structured multiplier[C]//Proc of the Digital System Design, Euromicro Symposium. 2004: 508-515.

[5] OHKUBO N, SUZUKI M, SHINBO T. A 4.4 ns CMOS 54×54-b multiplier using pass-transistor multiplexer[J]. IEEE Journal of Solid-State Circuits, 1995, 30(3): 251-257.

[6] YEH Wen-chang, JEN Chein-wei. High-speed Booth encoded parallel multiplier design[J]. IEEE Trans on Computers, 2000, 49(7): 692-701.

[7] BESLI N, DESHMUKH R G. A novel redundant binary signed-digit (RBSD) Booth's encoding[C]//Proc of IEEE SoutheastCon. 2002: 426-431.

[8] ELGUIBALY F. A fast parallel multiplier-accumulator using the modified Booth algorithm[J]. IEEE Trans on Circuits and Systems, 2000, 47(9): 902-908.

[9] WEISS M A. Data structures and algorithm analysis in C[M]. 2nd ed. [S.l.]: Addison Wesley, 1996.

[10] DANYSH A N, SWARTZLANDER E E. A recursive fast multiplier[C]//Proc of the 32th Asilomar Conference on Signals, Systems and Computers. 1998: 197-201.

[11] 梁峰,邵志标,梁晋. Radix-16 Booth 流水线乘法器的设计[J]. 西安交通大学学报,2006, 40(10): 1111-1114, 1133.

[12] 葛亮,唐志敏. 一种支持无符号数的流水线乘法器[J]. 微电子学与计算机,2002, 19(10): 17-19.


(上接第 3726 页)以来是一件很困难的事情。本文根据前人对供应链柔性的研究成果,运用新的研究方法——可拓分析,建立了供应链柔性的物元模型,并结合案例分析,对模型的各个柔性维度作了定量分析。但是,本文的评价模型只考虑了供应链中主要的影响因素,实际使用时可根据具体情况适当增加和扩展。模型中经典域、节域的取值范围,也会随时间和供应链的运作情况动态变化。对模型的完善和拓展,以及相关问题的实证研究是下一步研究工作的重点内容。

**参考文献:**


[1] SLACK N. The flexibility of manufacturing systems[J]. International Journal of Operations and Production Management, 1987, 7(4): 35-45.

[2] VOUDOURIS V. Consulting mathematical programming techniques to debottleneck the supply chain of the chemical industries[J]. Computers and Chemical Engineering, 1996, 20(6): 1269-1274.

[3] VISWANADHAM N, SRINIVASA RAGHAVAN N R. Flexibility in manufacturing enterprises[J]. Sadhana, 1997, 22(2): 135-163.

[4] VICKERY S, CALANTONE R, DROGE C. Supply chain flexibility: an empirical study[J]. The Journal of Supply Chain Management, 1999, 35(3): 16-24.

[5] SABRI E H, BEAMON B M. A multi-objective approach to simultaneous strategic and operational planning in supply chain design[J]. The International Journal of Management Science, 2000, 28(5): 581-598.

[6] VOKURKA R J, O'LEARY-KELLY S. A review of manufacturing flexibility empirical research [J]. Journal of Operations Management, 2000, 18(4): 485- 501.

[7] 方明,邓明然. 供应链柔性综合评价的探讨[J]. 武汉理工大学学报:信息与管理工程版, 2002, 24(6): 23-25.

[8] DUCLOS L K, VOKURKA R J, LUMMUS R R. A conceptual model of supply chain flexibility[J]. Industrial Management & Data Systems, 2003, 103(6): 446-456.

[9] 徐健. 供应链柔性的增强途径研究[J]. 物流技术, 2006, 25(2): 52-54.

[10] 孟军,张若云. 供应链柔性综合评价体系研究[J]. 中国管理信息化, 2007, 10(9): 56-59.

[11] 蔡文. 物元模型及其应用[M]. 北京:科学技术文献出版社, 1994.

[12] 杨莉,李南,李桥兴. 软件项目的可拓分析[J]. 计算机与应用化学, 2008, 25(5).

[13] 王中兴,李桥兴. 依据主、客观权重集成最终权重的一种方法[J]. 应用数学与计算数学学报, 2006, 20(1): 87-91.

[14] 李桥兴,刘思峰. 一般位值公式及一般初等关联函数构造方法[J]. 系统工程, 2006, 24(6): 116-118.

[15] 李桥兴,刘思峰. 基于关联度离差赋权的可拓方法[J]. 广东工业大学学报, 2007, 24(1): 1-4.