

# 一种高压缩 Wallace 树的快速乘法器设计

朱鑫标, 施隆照

(福州大学 物理与信息工程学院, 福建 福州 350002)

**摘 要:** 介绍了一种 32 位有符号/无符号乘法器. 该乘法器采用改进的 Booth 编码减少了部分积个数, 并通过符号扩展的优化, 减少中间资源消耗, 对部分积进行统一的符号操作, 简化了程序设计的复杂性. 采用了 7:2 压缩结构的 Wallace 树及 64 位 Brent Kung 树结构超前进位加法器, 有效地提高了乘法器计算速度. 整个设计采用 Verilog 语言编写, 通过 Modelsim 仿真验证设计功能的正确性. 采用 Synopsys 的 Design Compiler 进行基于 SMIC 的 0.18 微米标准库的综合并得到性能参数.

**关键词:** 乘法器; 7:2 压缩器; Wallace 树; Brent Kung 树; Booth 算法

**中图分类号:** TN79

**文献标识码:** A

**文章编号:** 1000-7180(2013)02-0046-04

## A Design of Multiplier With High—Compress Wallace Tree

ZHU Xin-biao, SHI Long-zhao

(College of Physics and Informattion, Fuzhou University, Fuzhou 350002, China)

**Abstract:** This  $32 \times 32$  multiplier supports both signed and unsigned integer multiplication by an additional sign bit. It adopts the Booth algorithm to reduce the number of partial product and optimize it. The multiplier increases the calculation speed through the Wallace tree with 7:2 compressor and Brent Kung tree. It is designed by verilog language and simulated correctly by Modelsim. The performance parameters are obtained by Synopsys' Design Compiler with SMIC's 0.18 $\mu$ m target library.

**Key words:** multiplier; 7:2 compressor; Wallace tree; Brent Kung; Booth algorithm

### 1 引言

乘法器是 CPU 内核和 DSP 内核中被调用频率很高的部件<sup>[1]</sup>, 是多媒体处理的重要组成部分. 速度和面积是乘法器研究的主要性能参数. 目前, 乘法器的设计方案和技术已较为成熟, 广泛采用改进的 Booth 编码加 4:2 压缩的 Wallace 树结构, 实现减少部分积个数和加快部分积压缩的目的. 本文设计的 32 位有符号/无符号乘法器, 采用基于改进的 Booth 编码加 7:2 压缩的 Wallace 树的总体结构, 对部分积采用符号优化方案避免增加一个符号补偿的部分积. 针对本设计将产生的 17 个部分积, 采用两级 7:2 压缩结构, 相比于采用 4:2 压缩的结构, 不仅具有压缩速度快, 延时短的特点, 而且充分利用了模块资源, 发挥了结构优势.

### 2 乘法器结构

乘法器在计算处理中, 先根据被乘数和乘数生成部分积, 再将部分积进行相加得到计算结果. 为了提高乘法器的性能, 可以通过减少部分积个数, 再对部分积进行并行相加<sup>[2]</sup>. 本设计的乘法器结构如图 1 所示.

### 3 改进的 Booth 算法编码及符号优化

采用改进的 Booth 算法可以减少一半的部分积个数, 是提高乘法运算速度的一个关键部分. 对部分积进行符号优化, 减少由于符号延伸引起的资源消耗, 减少了资源面积及后续累加的操作数个数.

#### 3.1 改进的 Booth 算法

改进的 Booth 编码算法是针对有符号数而言的,

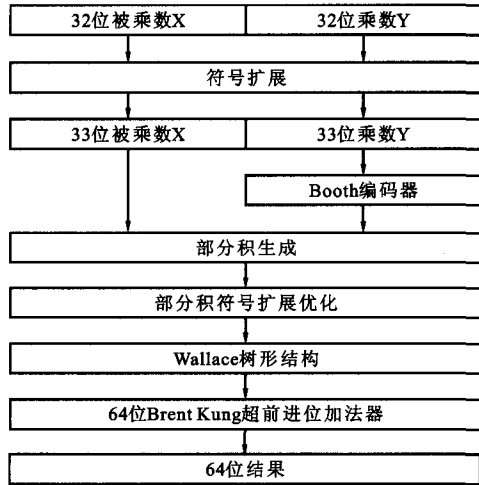


图 1 乘法器的结构图

假设二进制被乘数、乘数分别为  $X = \{x_{31}x_{30}\cdots x_1x_0\}$  和  $Y = \{y_{31}y_{30}\cdots y_1y_0\}$ , 为区分是有符号数和无符号数, 要先进行符号扩展, 使改进的 Booth 编码能兼容无符号数的乘法操作, 即在乘数和被乘数的高位增加一位用于符号控制。若为无符号乘法, 则分别扩展一位  $x_{32} = 0, y_{32} = 0$ ; 若为有符号乘法, 则扩展一位  $x_{32} = x_{31}, y_{32} = y_{31}$ 。改进的 Booth 算法原理: 乘数  $Y$  按下式进行编码, 得到一组全新的编码结果:

$$\begin{aligned} Y &= -y_{n-1}2^{n-1} + \sum_{k=0}^{n-2} y_k 2^k \\ &= \sum_{k=0}^{n/2-1} (-2y_{2k+1} + y_{2k} + y_{2k-1})2^{2k} \\ &= \sum_{k=0}^{n/2-1} E_k 2^{2k} \end{aligned}$$

其中  $y_{-1} = 0$ 。式中,  $n$  是偶数, 如果是奇数位则在高位补符号位, 以扩展成偶数位。根据上式可知部分积只有位数的一半。

表 1 为不同的  $y_{2k+1}, y_{2k}, y_{2k-1}$  经改进的 Booth 编码后得到结果  $E_k$  及其将对被乘数  $X$  的操作。

表 1 Booth 编码操作表

| $y_{2k+1}$ | $y_{2k}$ | $y_{2k-1}$ | $E_k$ | 执行的操作            |
|------------|----------|------------|-------|------------------|
| 0          | 0        | 0          | 0     | 加 0              |
| 0          | 0        | 1          | +1    | 加 $X$            |
| 0          | 1        | 0          | +1    | 加 $X$            |
| 0          | 1        | 1          | +2    | 左移 $X$ , 相加      |
| 1          | 0        | 0          | -2    | 对 $X$ 求补, 左移, 相加 |
| 1          | 0        | 1          | -1    | 对 $X$ 求补, 相加     |
| 1          | 1        | 0          | -1    | 对 $X$ 求补, 相加     |
| 1          | 1        | 1          | 0     | 加 0              |

由表 1 可知, 改进的 Booth 算法将产生 0, 1, -1, 2, -2 这 5 种倍数关系, 其中, 系数为 -2 的部分积可按公式  $-2X = \{X, 1\} + 1$  实现, 即先将  $X$  左移

一位并取反, 然后在末尾加 1, 最后结果再加 1。程序设计中, 对 Booth 编码结果为负的部分积项进行对应的取反和相应的移位, 对补码计算中的加 1 操作统一在 Wallace 树压缩中进行, 对编码结果为正的部分积进行加 0 操作, 统一两种情况即执行加符号  $S$ 。根据 Booth 编码符号扩展规律可知, 最后一个 Booth 编码的结果  $E_{16}$  为 0 或 1, 约定最后一行部分积不进行末尾加  $S$  的操作, 这样就保证了总体部分积个数不变, 不会对 wallace 树加法器产生影响。

### 3.2 符号优化

由于乘法过程中被乘数的符号是不确定的, 需要进行符号延伸, 补齐所有的符号位。为减少这部分由符号延伸引起的资源消耗, 可进行符号优化产生统一格式的符号位。对 32 位乘法器而言, 产生的部分积中最长的有效位为 33 位, 所以可以先对所有部分积的第 32 位以后的符号进行求和,  $\text{sign}$  为符号位的求和结果,  $s_i$  为第  $i$  位符号,  $\bar{s}_i$  为  $1 - s_i$ ,  $2^i$  为  $s_i$  对应的权位:

$$\text{sign} = s_0 \sum_{i=33}^{65} 2^i + s_1 \sum_{i=35}^{65} 2^i + \cdots + s_{16} \sum_{i=65}^{65} 2^i$$

$$\text{因为 } \bar{s}_i = 1 - s_i, \sum_{m=i}^n 2^m = 2^{n+1} - 2^i, \text{ 则:}$$

$$\begin{aligned} \text{sign} &= (1 - \bar{s}_0)(2^{66} - 2^{33}) + (1 - \bar{s}_1)(2^{66} - 2^{35}) \\ &+ \cdots + (1 - \bar{s}_{16})(2^{66} - 2^{65}) \\ &= (17 - \sum_{i=0}^{16} \bar{s}_i)2^{66} + \sum_{i=0}^{16} \bar{s}_i 2^{2i+33} - \sum_{i=0}^{16} 2^{2i+33} \\ &= (16 - \sum_{i=0}^{16} \bar{s}_i)2^{66} + \sum_{i=0}^{16} \bar{s}_i 2^{2i+33} + 2^{33} + \sum_{i=0}^{15} 2^{2i+34} \end{aligned}$$

由于结果只有低 66 位有效, 所以计算符号扩展的和只需计算第 0 到第 65 位即可。

$$\begin{aligned} \text{sign} &= \sum_{i=0}^{16} \bar{s}_i 2^{2i+33} + 2^{33} + \sum_{i=0}^{15} 2^{2i+34} \\ &= \sum_{i=1}^{16} \bar{s}_i 2^{2i+33} + \sum_{i=1}^{15} 2^{2i+34} + 2^{33}(3 + s_0) \end{aligned}$$

将  $s_0$  具体到 0 或 1 时可得第一个部分积的符号优化结果为从高位第 35 位至低位第 33 位的符号位依次为  $\bar{s}s$ , 第二到第十五个部分积的优化结果的第 34 至 33 位的符号位为  $1s$ , 第十六个部分积的第 33 位符号位为  $\bar{s}$ , 然后将每个部分积相对于前一个部分积左移两位即得到最后的具有实际权位的部分积结构。

32 位无符号数乘法器的计算结果的范围为

0000000000000000~0FFFFFFF00000001, 其中  $Z^{65} = Z^{64} = 0$ ; 32 位有符号数乘法器的计算结果的范围为  $3C000000FFFFFF \sim 03FFFFFF00000001$ , 其中  $Z^{65} = Z^{64} = Z^{63}$ , 即  $Z^{65}$ 、 $Z^{64}$  为  $Z^{63}$  的符号扩展. 所以  $32 \times 32$  位乘法器扩展成  $33 \times 33$  乘法操作, 其结果仍可以用 64 位表示<sup>[3]</sup>.

#### 4 Wallace 树结构

采用 Wallace 树结构对部分积进行进位保留压缩, 使 17 个部分积快速化累加成 2 个部分积.

##### 4.1 压缩器设计

7:2 压缩器采用 5 个 3:2 的全加器排列实现, 可以使乘法计算过程中相同列数据实现将 7 个输入项和两个进位输入项快速叠加生成两个输出项和两个进位输出项, 其中两个进位输入依次来自于当前处理数据的邻近低位和次邻近低位产生的进位输出. 其结构如图 2 所示.

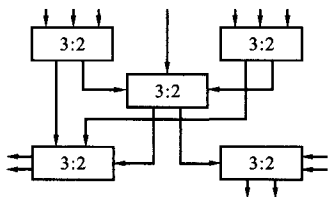


图 2 7:2 压缩器原理图

其中 3:2 压缩器原理如图 3 所示.

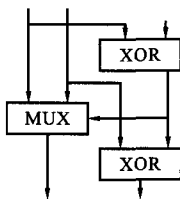


图 3 3:2 压缩器原理图

目前应用比较广泛的 4:2 结构, 输入 4 个输入数据及 1 个进位输入, 输出两个输出数据及一个进位输出. 其结构如图 4 所示. 其中的 3:2 压缩器如图 3 所示.

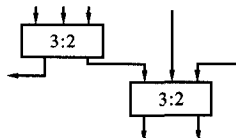


图 4 4:2 压缩器原理图

##### 4.2 Wallace 树阵列结构

采用 4:2 压缩器的 Wallace 树算法对于位数较少的乘法器而言, 具有较高的速度, 但对于位数较多的乘法器, 其优势受到限制. 在对本设计的对象及部分积算法进行充分分析后, 选择使用 7:2 压缩器对

部分积进行叠加. 本设计执行的累加过程如下公式所示. 该结构将 17 个部分积分成两级执行, 充分利用了 7:2 压缩器的资源.

$$P = \sum_{n=0}^{17} p_n$$

$$= (p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6) + (p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} + p_{13}) + p_{14} + p_{15} + p_{16}$$

表 2 压缩器性能表

| 分析对象   | 压缩结构      | 消耗资源(以 XOR 和 MUX 为单位) | 延时(以 XOR 为单位) |
|--------|-----------|-----------------------|---------------|
| 17 部分积 | 7:2       | 30XOR, 15MUX          | 12            |
|        | 4:2       | 32XOR, 16MUX          | 16            |
|        | 3:2 和 4:2 | 30XOR, 15MUX          | 14            |
| 22 部分积 | 7:2       | 40XOR, 20MUX          | 12            |
|        | 4:2       | 40XOR, 20MUX          | 16            |

表 2 为两种压缩结构处理 17 和 22 个部分积时消耗的资源及延时. 通过对比可知, 在处理 17 个部分积的过程中, 7:2 结构不仅比 4:2 结构有优势, 而且与 3:2 和 4:2 压缩器相结合的新型结构相比也有一定的速度优势. 从表中可知, 在处理 22 个部分积时, 两种结构虽然使用了相同的资源数, 但 7:2 结构相比于 4:2 结构有可观的速度优势. 7:2 结构的优势在多部分积极的结构中将更为明显.

#### 5 Brent Kung 加法器

17 个部分积进过 Wallace 树压缩后, 根据 32 位乘法器的结果范围, 合理截取两个部分积的前 64 位, 采用 64 位的 Brent Kung 加法器计算得到最后结果. Brent Kung 通过二元树形结构克服了电路的扇入、扇出问题, 以常规的结构获得运算的高速度. 输入数据为  $X = (x_{63}, x_{62}, \dots, x_0)$  和  $Y = (y_{63}, y_{62}, \dots, y_0)$ , 第  $i$  位进位产生信号和进位传输信号分别为  $g_i = x_i y_i$  和  $p_i = x_i \oplus y_i$ . 采用“ $\cdot$ ”运算符对每个进位产生信号和进位传输信号进行运算<sup>[4]</sup>. 其运算规则如下所示:  $(g, p) \cdot (g', p') = (g + pg', pp')$  其树形结构如图 5 所示.

首先将两个 64 位的部分积通过相同位进行异或运算得到对应位的  $p_i$  数据, 由相同位数据的与运算得到对应位的  $g_i$  数据. 由  $(p_i, g_i)$  构成了 Brent kung 树形结构的原始数据点, 通过树形结构迭代运算后, 再将  $p_i$  与对应位的上一级进位进行异或运算即得到每一位的计算结果.

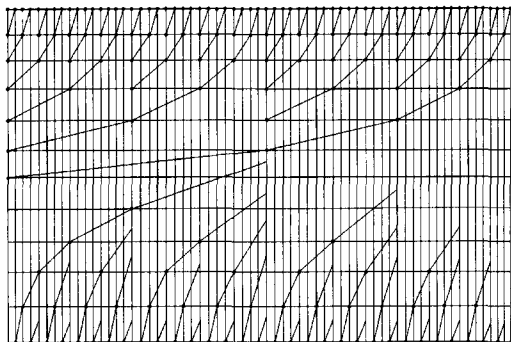


图5 64位 Brent Kung 树形结构

## 6 设计结果

本文设计的 32 位有符号/无符号乘法器采用 Verilog 语言编写代码,并通过 Modelsim6.5 仿真验证,仿真结果如图 6 所示。

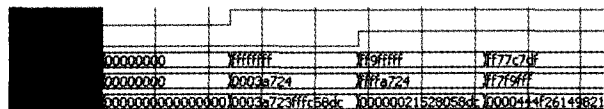


图6 Modelsim6.5 仿真结果

其中 rst 为复位控制信号, rst 为 0 时执行复位操作,为 1 时执行正常的计算操作. Sign 为数据类型标识信号, sign 为 0 时执行无符号乘法操作,为 1 时执行有符号数乘法操作. 测试数据的选取采用的是全面性和随机性相结合的方法. 乘数 Y 取 3a724、ffffa724 可遍历改进的 Booth 编码产生的 8 种输入情况产生的 5 种输出结果. 随机数选取可以检查 Wallace 树、Brent Kung 树的正确性. 通过仿真结果可知,本设计可对 32 位有符号数、无符号数正确计算得到 64 位结果,满足设计要求。

设计采用 Xilinx 公司的 XC2VP100 为目标芯片,综合结果的延时为 39.1ns,使用 1485 个 slice、2397 个 LUT 及 110 个 IOB. 以 SMIC 的 0.18 微米标准库为目标库,使用 Synopsys 的 Design Compiler 进行综合得到本设计的最小延时为 4.2ns,所需

面积为 0.1629mm<sup>2</sup>. 其中采用 7:2 压缩的 wallace 模块延时 1.9ns, 占用面积 0.0674mm<sup>2</sup>; 64 位 Brent Kung 加法器模块延时 1.66ns, 占用面积 0.007mm<sup>2</sup>. 与文献[5]的由 3:2 和 4:2 压缩器构成的新型 Wallace 结构及设计结果相比,速度提高了 3%。

## 7 结束语

文中介绍了一种有符号/无符号的 32×32 乘法器的实现方法,使用一个符号标识位识别有符号数和无符号数操作,通过改进的 Booth 编码减少了部分积个数,并通过符号延伸的优化减少了中间资源消耗,对部分积进行统一的符号操作,简化了程序设计的复杂性. 本设计采用了 7:2 压缩结构的 Wallace 树及 64 位 Brent Kung 树结构超前进位加法器,有效地提高了计算速度. 仿真验证了设计的正确性,并通过综合得到性能参数,整个设计简单高效。

## 参考文献:

- [1] 吴辉,张磊. 一种新颖的可重组乘法器设计[J]. 微电子学与计算机, 2004, 22(9): 161-163.
- [2] 刘东. 采用 Booth 算法的 16×16 并行乘法器设计[J]. 现代电子技术, 2003(9): 21-22.
- [3] 汤晓慧,杨军. 基于 Booth 算法的 32×32 乘法器的 IP 核设计[J]. 电子器件, 2005, 28(1): 218-220.
- [4] 陶智德,林涛. 32 位高速复合加法器设计[J]. 电子工程师, 2004, 30(11): 22-23.
- [5] 李军强. 32×32 高速乘法器的设计与实现[J]. 微电子学与计算机, 2009, 26(12): 23-26.

## 作者简介:

朱鑫标 男, (1988—), 硕士研究生. 研究方向为数字集成电路设计.

施隆照 男, (1968—), 副教授, 硕士生导师. 研究方向为集成电路芯片设计.

(上接第 45 页)



何洪波 男, (1970—), 硕士研究生, 高级工程师. 研究方向为大规模科学数据管理及应用、基于 WEB 的数据可视化处理.



尹海龙 男, (1983—), 工程师. 研究方向为多媒体与交互技术.