DOI:10.19651/j.cnki.emt.2416555

基于符号扩展的 booth 乘法器设计与实现*

熊书伟[1,2] 宋树祥[1,2]
(1.广西类脑计算与智能芯片重点实验室 桂林 541004; 2.广西师范大学电子与信息工程学院 桂林 541004)

摘 要: 针对 RISC-V 处理器中的乘法器部分延时较高以及功耗较大的问题,本文在 booth2 算法的基础上,提出一种改进的基于符号扩展的乘法器优化设计,减少了处理器中乘法指令的执行周期并同时支持有/无符号数的运算。改进了 CSA32 压缩器,并选择交替使用 3-2 压缩器和 4-2 压缩器的 Wallace 树形结构,提高了部分积的压缩效率,还缩短了关键路径的延时,提高了乘法器的运算速度。利用 NC-verilog 等验证工具对乘法器进行编码验证以及功能仿真,使用 Design complier 在 SIMC 180 nm 工艺下进行综合分析,结果表明本文设计的乘法器相较于 PicoRV32,乘法指令执行周期缩短了 88.2%,面积与功耗也优于同类乘法器。

关键词: RISC-V 处理器;乘法器;booth 算法;符号扩展;Wallace 树形结构
中图分类号: TN791 **文献标识码**: A **国家标准学科分类代码**: 520.502

Design and implementation of booth multiplier based on symbol extension

Xiong Shuwei[1,2] Song Shuxiang[1,2]
(1. Guangxi Key Laboratory of Brain-inspired Computing and Intelligent Chips, Guilin 541004, China;
2. Education Department of Guangxi Zhuang Autonomous Region, Guangxi Normal University, Guilin 541004, China)

Abstract: Aiming at the problems of high delay and high power consumption of the multiplier part in RISC-V processors, this paper proposes an improved multiplier optimisation design based on symbol extension on the basis of the booth2 algorithm, which reduces the execution cycle of multiplication instructions in the processor and supports the operation of signed/unsigned numbers at the same time. The improved CSA32 compressor and the choice to alternate the Wallace tree structure with a 3-2 compressor and a 4-2 compressor improves the compression efficiency of the partial product, and also reduces the critical path delay and improves the speed of the multiplier operation. The coding verification and functional simulation of the multiplier are carried out using verification tools such as NC-verilog, and the comprehensive analysis is carried out using. Design complier at SIMC 180 nm process, and the results show that the multiplier designed in this paper reduces the multiplication instruction execution cycle by 88.2% compared with PicoRV32, the area and power consumption are better than those of the same type of multiplier.

Keywords: RISC-V processor; multiplier; booth algorithm; symbol extension; Wallace tree structure

0 引 言

芯片行业的新晋热门—RISC-V 架构[1],与 X86 与 ARM 架构相比,RISC-V 架构以精简指令集和开放性赢得了关注[2],它在功耗、兼容性及生态建设方面展现出强大潜力,正逐步打破传统界限,被广泛应用于嵌入式系统、物联网、电力行业、ICT 和机器人技术[3]。

在处理器的设计中,乘法器作为核心运算单元,对整个处理器的运算速度有着重要的影响。因此在嵌入式领域[4]中,对乘法算力要求较高的场景如神经网络的卷积运算,研究如何提升处理器中乘法器的性能同时降低功耗是十分必要的。乘法运算通常包括部分积生成、部分积压缩和最终结果相加这 3 个步骤[5]。目前乘法器的设计中主要有 booth 编码和 Wallace 树形结构[6]。现阶段已有众多研究对这两部分理论进行了改良优化,乘法器的运算效率也逐步提高。文献[7]基于符号补偿改进了基 4-booth 编码以及采取对称的 Wallace 树形结构来减少压缩层次,但不同时支持有符号数和无符号数的计算且在取反计算时并未进

收稿日期:2024-07-29
*基金项目:国家自然科学基金(62061005)、广西自然科学基金(2022GXNSFBA035646)项目资助

行针对性的优化,会造成资源的浪费。文献[8]通过优化取反加 1 的方法直接生成被乘数的相反数,同时采用符号补偿使得部分积阵列更加规整,针对每行部分积的数据特征,优化了 Wallace 结构,不过同样也不支持无符号的计算,并且通过单个全加器处理压缩器的中间进位会使得关键路径延时较长。文献[9]提出了一种新型 5-2 压缩器和 4-2 压缩器结合的 Wallace 树形结构,该结构提高了压缩效率同时减少了乘法器关键路径延时,而高阶压缩器消耗的硬件资源更多,导致芯片面积较大。

在不增加处理器功耗的情况下,为了提高处理器的运算速度[10],优化乘法器的性能,本文提出了一种基于符号扩展、booth2 算法[11]以及改进型 Wallace 树形结构[12]的乘法器优化设计,通过对负数符号扩展[13]产生的连续比特 1 进行优化,可以减少在累加过程中产生的大量进位和比特翻转,从而节省资源并降低功耗。改进 Wallace 树形结构,采取 3-2 压缩器和 4-2 压缩器交替使用的对称结构,有助于减少结构层次,缩短关键路径的延时,降低指令执行周期,从而提高乘法器的运算效率。在 SIMC 180 nm 的工艺下对乘法器进行综合分析与性能评估,最终结果表明,本文所设计的乘法器性能有显著提高,功耗和面积也有所降低。

1 Radix-4 Booth 算法

A. D. Booth 于 1951 年在其论文"A Signed binary multiplication technique"中提出一种新型乘法——booth 算法[14],旨在解决有符号乘法运算中复杂的符号修正问题。booth1 算法因为其不足之处而导致使用率并不高,在计算 n 比特数相乘的过程中(n 位偶数),会生成 n 个部分积,想要计算出最终结果,需要对这 n 个部分积进行压缩运算,比如说如果是两个 32 比特数相乘,电路就需要对这 32 个部分积进行压缩运算。因此,随着部分积个数的增加,压缩部分积所需的逻辑资源和延时也会相应增加。

booth1 编码器并没有减少生成的部分积的数量,在其基础上进行改进,提出了 booth2 编码,该编码器明显减少了生成的部分积数量,大幅降低了电路的延时与功耗,其算法的数学表达式如下:

对于 n 比特数 B 来说:

$$B = -B_{n-1}2^{n-1} + \left(\sum_{i=0}^{n-2} B_i \times 2^i\right) \tag{1}$$

booth2 乘法器的基系数为:

$$B = -B_{n-1}2^{n-1} + \left(\sum_{i=0}^{n-2} B_i \times 2^i\right) = (-2B_{n-1} + B_{n-2} + B_{n-3})2^{n-2} + \cdots + (-2B_1 + B_0 + B_{-1})2^0 \tag{2}$$

其中,B_{-1} 为 0,将 A 与 B 相乘,则:

$$A \times B = A \times (-2B_{n-1} + B_{n-2} + B_{n-3}) \times 2^{n-2} + \cdots + (-2B_1 + B_0 + B_{-1}) \times 2^0 \tag{3}$$

以下是 booth2 编码表,其中 A 为被乘数,B 为乘数。相应的编码规则表如表 1 所示。

表 1 booth2 编码规则表

Table 1 booth2 coding rule list

$B_{i+1}B_iB_{i-1}$	$-2B_{i+1}+B_i+B_{i-1}$	部分积操作
000	+0	0
001	+1	1A
010	+1	1A
011	+2	2A
100	−2	−2A
101	−1	−1A
110	−1	−1A
111	−0	0

通过对比 booth1 编码可以发现,booth2 编码生成的部分积数量显著降低。不过 booth2 编码的系数仍可能出现负数的情况,所以在压缩时需要注意负数部分积造成的影响。如果部分积为负数,通过符号扩展来补齐位数时将其符号位补齐为 1,就会使得累加计算时出现大量进位和翻转,乘法器的功耗也会随之增加。

2 Wallace 树形结构

1964 年,一种高效快速的加法树结构由 C. S. Wallace 提出,后人称之为 Wallace 树。booth2 算法对整型乘法中生成的部分积数量进行了优化,降低了部分积累加次数,而 Wallace 树的引入进一步减少了累加操作的延迟。该结构将每 3 个加数作为一组输入,经过压缩器压缩为计算结果和进位。依次循环计算,得到最终的累加结果,其结构如图 1 所示。

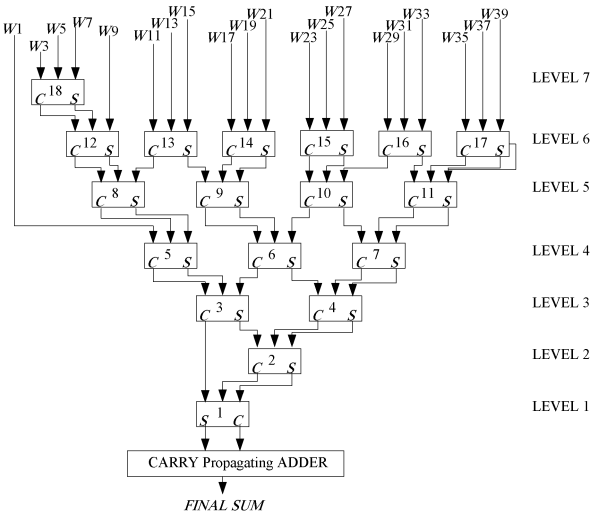

图 1 Wallace 树形结构

Fig. 1 Wallace tree structure

Wallace 树型结构提高了硬件压缩器的复用率,有效地缩短了累加操作的执行时间,减少了关键路径的时延,从而

加快了乘法器的计算速度。不过传统 Wallace 树形结构的硬件压缩器功能较为简单，且结构层次仍然较多，此外，若是出现由于数据个数导致结构不对称的情况，不仅使得电路的结构更加复杂，而且会造成不规则的布局布线，这样的结构还会消耗更多的资源，进而增加乘法器的面积和功耗。

随着应用程序的处理需求呈指数级增长，研究乘法器算力与功耗的优化变得尤为重要，为此各种高阶压缩器被引入应用，同时调整优化 Wallace 树型结构，在不影响其功耗和面积的前提下，以提高乘法器性能。

3　乘法器的优化设计

3.1　乘法器整体结构

本文设计的乘法器主要模块包括：基于改进的 booth2 编码器生成部分积，部分积符号扩展模块提供一个额外的部分积，再将所生成的部分积通过改进的 Wallace 树型结构以及新型 4-2 压缩加法器对部分积进行压缩计算。设计的乘法器整体架构图如图 2 所示。

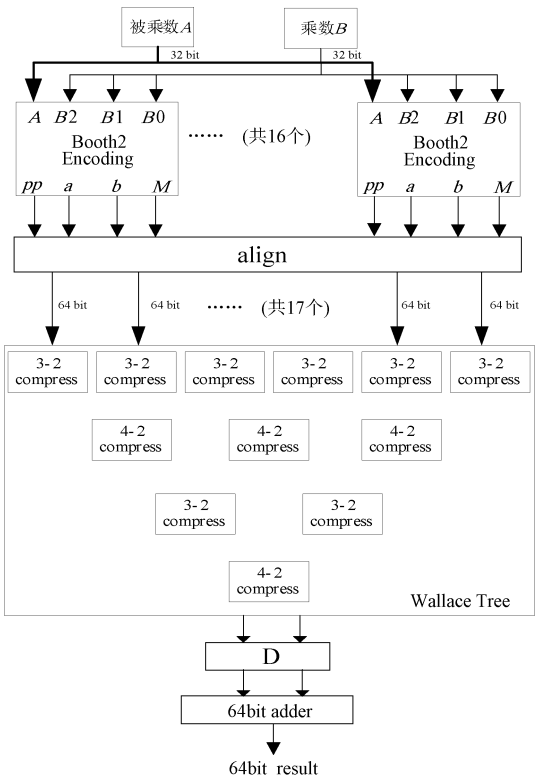

图 2　乘法器整体架构图

Fig. 2　Multiplier overall architecture diagram

两输入被乘数 A 与乘数 B 均为 32 bit。主要分为以下 3 部分：

1)对 B 进行 booth2 编码，A 作为基量，在 booth2 算法中生成 16 个 booth 部分积。对于 32 bit 的乘法，一共需要 16 个 booth2 编码器。

2)在 align 模块中对部分积进行对齐，补偿。为了兼容无符号乘法以及对 A 取反中的加法运算的补偿，会额外添加一个部分积，共 17 个部分积。

3)对于 17 个 64 bit 的加法，为了尽可能地减少 Wallace 树形结构的层次，并降低其延迟，在不提高电路功耗的情况下，使用一种交替的 3-2 压缩器与 4-2 压缩器的树型结构，且该两数进位与结果分离，不依赖于前位的进位，因此时序较好。在一系列的组合压缩后，最终会生成 2 个 64 bit 的数，将两数相加便能得到乘法结果。64 bit 的加法具有一定的延迟，因此在进行 64 bit 的加法前，进行一次插拍，以保证整体的时序。

本文将对补码的相关计算，同时支持无符号和有符号的计算以及符号位的扩展与 booth2 编码相结合，对 booth2 算法进行了改进，对于多比特数相乘且部分积较多的情况下可以节省大量的面积和功耗，同时对于后级的 Wallace，也可以使用更少的压缩加法器。减少了树形结构的层次，还减少了硬件加法器的数量，降低关键路径的时延，最后通过 CLA(carry lead adder)得到乘法计算结果。

3.2　改进 booth2 算法(基 4 的 booth 算法)

对于 32 bit 的乘法器而言，若使用常规的移位乘法器，则需要进行 32 个部分积相加才能得到结果，其硬件开销以及延迟都比较大。引入 booth2 编码减少了部分积的数量，可以显著地减少加法的次数[15]，表 2 是相应的编码规则表。

表 2　改进 booth2 编码规则表

Table 2　Improvement of the booth2 coding rule table

$B_{i+1}B_iB_{i-1}$	操作	说明
000	+0	置为 0
001	+[A]补	保持不变
010	+[A]补	保持不变
011	+2[A]补	A 左移一位
100	−2[A]补	A 取反左移一位，加 2
101	−[A]补	A 取反加 1
110	−[A]补	A 取反加 1
111	−0	置为 0

为了最大限度复用资源，优化时序，上述的加一/加二操作并不会在当前的 booth2 算法编码时直接与 A 相加，而是生成相应的 a、b 信号，放入加法树中与剩余的部分积一同相加。a 表示需要+2，b 表示需要+1。

1) 符号位扩展的优化

假设 32×32 有符号乘法器的所有部分积均为正数，由于在 booth2 编码中有乘 2 操作，所以除了底部的部分积为 16 bit，其他部分积的位宽均为 17 bit，为了对部分积的计算更加方便，将其进行补齐，部分积的对齐示意图如图 3 所示。

需要注意以下几点：

(1)ai，bi 分别代表第 i 行的部分积需要进行+2/+1

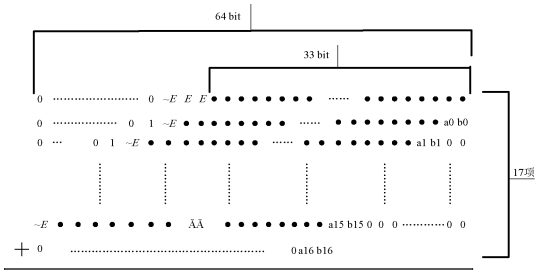

图 3 乘法器符号位对齐示意图

Fig. 3 Multiplier symbol bit alignment schematic

($i=0,1,2\cdots16$);

(2)因为第 i 行的部分积的+2/+1 部分,在第 $i+1$ 行进行补偿,因此 32 bit 的乘法器的部分积一共需要 32/2+1=17 项,其中最后一项称之为补偿项;

(3)本文改进的 booth2 算法并未采用常规的高位全补符号位的符号位扩展方法,而是采用了优化后的符号位扩展,并同时支持无符号数和有符号数的计算。

根据符号位补偿可知,符号位补齐的数与部分积的正负号有关:当部分积为正时,应将前面不足的位全部补 0;当部分积为负时,应将前面不足的位全部补 1,此时在进行累加时,就会产生大量的进位和比特翻转,这将增加资源消耗并且提高功耗。为了应对这种符号位扩展的情况,减少符号位扩展以及补码造成的资源消耗,有研究提出了一种有效的符号补偿方式[16]。本文基于该符号补偿方式将补码计算与 booth2 编码相结合,同时支持有符号数和无符号数的计算,形成改进后的 booth2 编码,下面进行详细介绍。

假设部分积全为负数,则最终的部分积如图 4 所示。

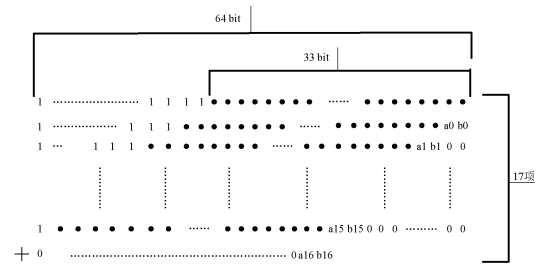

图 4 部分积全为负的示意图

Fig. 4 Schematic of a partial product that is all negative

因为符号位扩展前面部分全是 1,可以先将左上角扩展的符号位全部相加,也就是通过对 1 竖向相加,将进位的方式进行化简,可以得到图 5 所示的部分积。

假设其中某一项部分积不是负数的时候,则该部分积末尾加 1 的操作将会使连续的 1 被变回连续的 0,而实际上末尾加 1 只发生在乘数和与负数相乘的时候,即 booth 编码为 101,110 的时候。此时只需要在部分积符号位的前一位加上一个 1,将该行转化为正数情况下的符号扩展,如图 6 所示。

上述中当 E 为 0 时表示部分积为正数;E 为 1 时表示部分积为负数。可见在经过优化后,基于符号扩展的

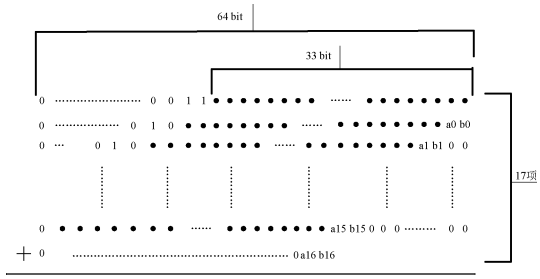

图 5 化简后全为负的部分积示意图

Fig. 5 Schematic diagram of partial products that are all negative after simplification

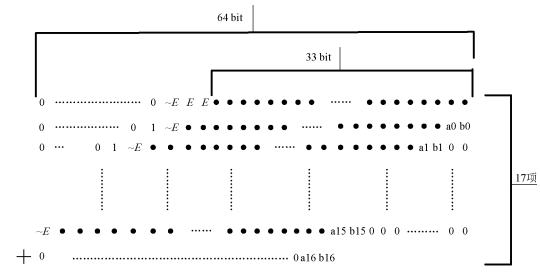

图 6 符号位扩展优化的部分积示意图

Fig. 6 Partial product schematic for symbol bit expansion optimisation

booth2 编码,并不需要将所有的符号位进行扩展,从而避免了连续的 1 造成的进位和翻转,使得资源的使用有了一定程度的降低,并且能够降低乘法器一部分的功耗。

2) 无符号数的支持

本文所设计的乘法器除了支持有符号乘法外,还能同时支持无符号乘法的计算。通过对比,与有符号乘法不同的是,无符号乘法的最高位的权重不再是 $-B_{n-1}\cdot2^{n-1}$ 符号位,而是 $B_{n-1}\cdot2^{n-1}$。由此可得:

$$B=B_{n-1}\cdot2^{n-1}+B_{n-2}\cdot2^{n-2}+\cdots+B_1\cdot2^1+B_0\cdot2^0=\sum_{n=0,B_{-1}=0}^{(n-1)}(B_i)\cdot2^i \tag{4}$$

与有符号乘法的 booth2 展开式相比较可以发现,此时唯一的区别只有 B_{n-1} 位,因此在计算无符号乘法时,可以复用有符号乘法的部分积,只需要对 B_{n-1} 位进行补偿,方法为 $2B_{n-1}\cdot2^{n-1}$,即在进行 a/b 加法补偿的同时,若 $B[31]$ 为 1,还需要再加上 A 左移 n 位。

复用有符号乘法的部分积,并对最后一行部分积进行修改,便能得到无符号乘法的部分积,如图 7 所示($B[31]=1$)。

为了能够同时支持有符号乘法和无符号乘法的计算,可以将部分积改为图 8 所述结构。

3.3 Wallace Tree

1) 传统 CSA32 压缩器

CSA32 的原理与全加器类似,即输入 3 个信号,一个输出结果信号,一个输出进位信号,压缩比为 3∶2。与全加器不同的是,3-2 压缩器的进位信号,需要左移一位

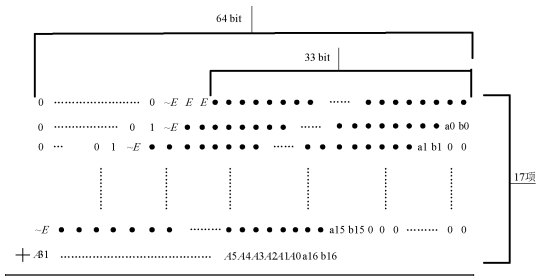

图 7　无符号部分积对齐示意图

Fig. 7　Unsigned partial product alignment schematic

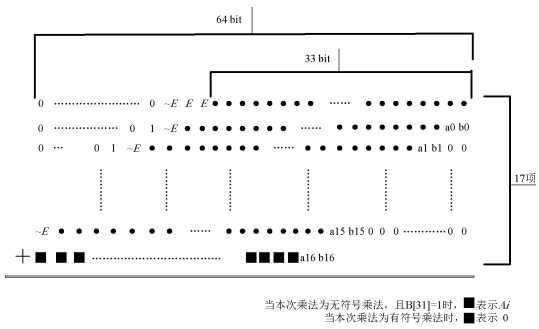

图 8　乘法器符号位扩展示意图

Fig. 8　Multiplier sign bit expansion schematic

($carry$ 应比 sum 高一位)，舍掉最高位。其中 CSA32 压缩器的逻辑表达式如下：

$$sum = in0 \oplus in1 \oplus in2 \tag{5}$$

$$carry = in0 \cdot in1 + in0 \cdot in2 + in1 \cdot in2 \tag{6}$$

CSA32 的结构如图 9 所示。

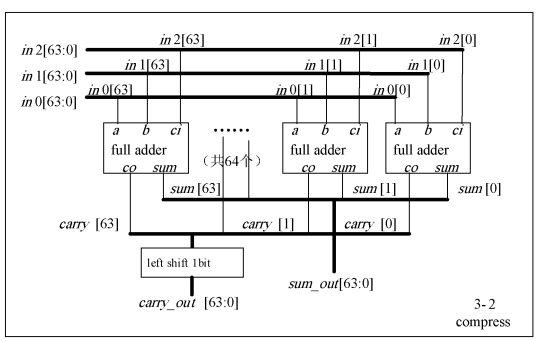

图 9　CSA32 结构图

Fig. 9　CSA32 structure diagram

2）改进 CSA42 压缩器

4-2 压缩器与 3-2 类似，区别仅为输入是 4 个数，压缩比为 2∶1。4-2 压缩器可以有两种实现方式：使用 3-2 压缩器搭建的 4-2 压缩器，以及根据全加器原理改进后直接使用门电路搭建 4-2 压缩器。两种 4-2 压缩器的搭建方式分别如下：

基于 3-2 压缩器搭建的 4-2 压缩器：由上述 3-2 压缩器可知：可以将 4 输入进行拆分：$in0-in2$ 作为第一级 3-2 压缩器的输入，两位的输出加上 $in3$ 作为第二级 3-2 压缩器的输入，如图 10 所示。

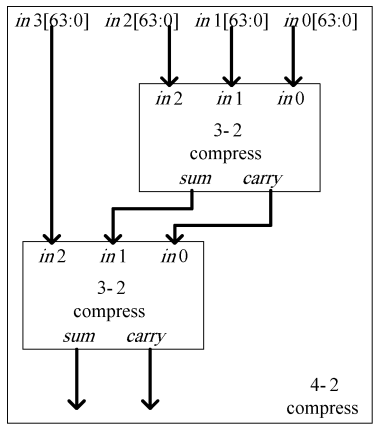

图 10　由 3-2 压缩器组成的 4-2 压缩器

Fig. 10　4-2 Compressor from 3-2 compressor

改进后的 CSA42 压缩器的逻辑表达式如下：

$$c_i = in0 \cdot in1 + in0 \cdot in2 + in1 \cdot in2 \tag{7}$$

$$in_{or} = in0 \oplus in1 \oplus in2 \oplus in3 \tag{8}$$

$$sum = in_{or} \oplus c_{i-1} \tag{9}$$

$$carry = in_{or} \cdot c_{i-1} + \overline{in_{or}} \cdot in3 \tag{10}$$

式中：sum 为当前压缩器的压缩结果，c_i 为同级压缩器的进位 $carry$ 为该压缩器的进位。

3）改进 Wallace 树形结构

基础的 Wallace 树型结构利用压缩器完成所有部分积的快速累加[17]，其核心器件通常为 3-2 压缩器，3 个位权相同的部分积作为输入，生成求和与进位两个结果，通过树型结构可以观察到，加法器的数量明显减少，加法器的复用性显著增加，有效提高了累加操作的并行性。但若是仅采用单一的 3-2 压缩器作为基本加法器，会增加 Wallace 树型结构的关键路径延时[18]，同时也无法充分发挥设计的灵活性。高阶压缩结构虽然有更高的压缩率，但也必须考虑到更复杂的结构带来的资源消耗和延时代价

本文针对部分积数量和资源利用进行了分析，采取 3-2 压缩器和 4-2 压缩器交替使用的树形结构，该压缩结构对称，压缩结构仅为四级，降低了压缩层次，本文 4-2 压缩器的关键路径为 3 级 XOR 门，与传统的 4-2 压缩器相比减少了 1 个 XOR 门的延时。关键路径延时较少，提高了压缩效率并节省了面积。改进后的 Wallace 树形结构如图 11 所示。

4　功能验证与性能仿真

4.1　功能验证与分析

本文设计的 32 位乘法器由 Verilog 语言进行描述，并利用 Cadence 公司下的 NC-Verilog 仿真工具进行编译仿真，用于验证设计的正确性，图 12 为仿真测试图。

采用随机函数生成 1 万组补码形式的数据投入本文的乘法器中，因为仿真的数据量比较多，故选取一部分仿真结

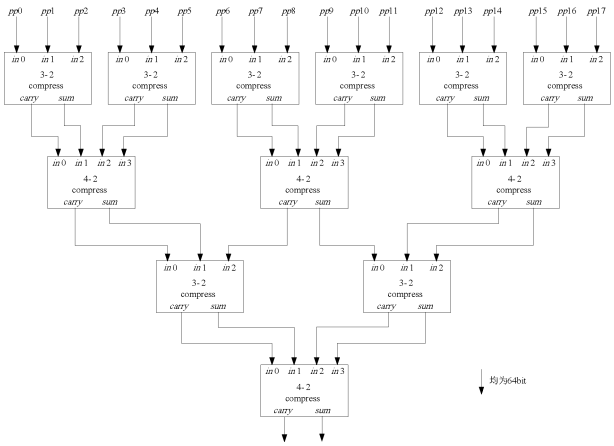

图 11　改进后的 Wallace 树形结构

Fig. 11　Improved Wallace tree structure

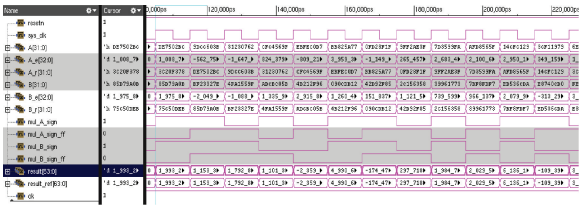

图 12　乘法器功能仿真波形图

Fig. 12　Multiplier function simulation waveform

果图进行展示。通过波形图 12 可以看出,在乘数和被乘数输入后,在第 1 个时钟周期计算累加结果和进位,在第 2 个时钟周期通过 64 位的加法器将两者进行相加,在经过 2 个时钟周期后,即可得到最终输出结果。因此本文设计的乘法执行周期为 2 个时钟周期,大大缩短了乘法指令的执行延迟,提升了乘法器的运算速度。

通过加入信号 *mul_a_sign* 和 *mul_b_sign* 来确定输入数据 *A* 和 *B* 是不是有符号数,因为有符号数和无符号数的最高位代表的意思不一样,若是对输入的最高位不分情况就进行处理,这样就会导致最终计算出来的结果也是不一样的,所以在输入无法确定是有符号还是无符号的情况下,并不能人为的直接选择波形图为 *signed* 或者 *unsigned*,此时最好的办法就是加入自动比对,因此在测试激励文件里面加入自带乘法符号的运算结果与本文设计的乘法器的结果相比对,并输出计算结果,*signed* 为 0 时表示为无符号数,为 1 时表示为有符号数。信号 *result_ref* 是自带符号的运算结果,信号 *result* 是乘法器完整运算结果,以前者的结果为标准进行核对,将两者的数据进行对比,若 *OK* 为 1 则表示两者数据完全一致,设计的乘法器结果计算正确。

之后再通过编写的 testbench 产生的随机数进行激励测试,对设计进行验证比对,并在终端输出打印结果用来验证计算结果,如图 13,对本文设计的乘法器进行功能验证,经过仿真显示乘法器计算结果与自带乘法符号的运算结果一致,表明计算正确,终端打印的结果也同样表示结果正确,验证本文设计的乘法器功能正确。

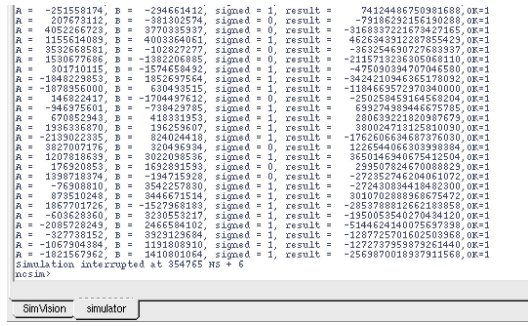

图 13　终端打印

Fig. 13　Terminal printing

在 Ubuntu 环境下,使用 VCS 工具为 Makefile 脚本添加代码覆盖率功能,通过设置-cm 参数开启不同类型的覆盖率统计,并通过 dve 查看覆盖率报告。表 3 为本文乘法器的代码覆盖率报告表,总覆盖率为 99.47%。

表 3　代码覆盖率表

Table 3　Code coverage table　%

覆盖率	booth 算法	Wallace 树	乘法器
行覆盖率	100	100	100
翻转覆盖率	100	100	100
条件覆盖率	97.88	100	97.88
分支覆盖率	100	100	100

4.2　性能仿真与分析

本文将基于 SMIC 公司 180 nm 的工艺库,使用 Synopsys 公司下的 Design Compiler 工具对设计的乘法器进行综合分析并得到门级网表,对其面积进行了相应的评估,最终所得到综合后的性能结果如表 4 所示。

表 4　乘法器性能对比表

Table 4　Multiplier performance comparison table

乘法器	PicoRV32	文献[9]	本文
周期	17	2	2
面积/μm^2	9 472.598	12 000	10 933.927

综合结果表明,本文设计的乘法器总单元数目为 6 311,文献[9]电路总单元数目为 6 844。根据表 3,本文的乘法器完成乘法运算的执行周期与文献[9]相同,同时与 PicoRV32 处理器进行对比,本文设计的乘法器乘法执行周期缩短了 88.2%,大幅减少了乘法指令的执行时间,提升了处理器的性能,可用于完成乘法指令的高效运算。同时对比文献[9],本文的乘法器在面积上低于文献[9],减少了 8.9%,因此本文设计的乘法器优于文献[9]。

5　结　　论

本文通过对 RISC-V 处理器中乘法指令的研究以及乘法器的研究，提出了一种基于符号扩展的 booth2 算法以及使用改进后的压缩器的 Wallace 树形结构，实现了 RISC-V 处理器中乘法器的优化。使用 NC-Verilog 等验证工具对设计的乘法器进行编码验证和功能仿真，通过 Design Complier 对乘法器进行综合分析，并与现已有的乘法器进行比对分析。结果表明本文设计的乘法器功能正确，通过查看代码覆盖率报告，可以看出整体覆盖率较为满意，不过还需查找设计上的缺陷，从而进行修正。完成乘法运算所需要的时钟周期为 2 个时钟，与 PicoRV32 相比乘法运算速度改善了 88.2%，大幅提高了乘法运算的计算速度，在乘法指令运算的执行周期与现已有的乘法器相同的情况下，面积和功耗均优于现已有研究，并适用于嵌入式领域对乘法运算需求较高的应用场景。后续还需要对加法器进行一定的优化，减少加法延迟并降低面积，并且将设计的乘法器投入到 RISC-V 处理器中完成最终的处理器优化。

参考文献

[1] TAHERI F, BAYAT-SARMADI S, HADAYE-GHPARAST S. RISC-HD: Lightweight RISC-V processor for efficient hyperdimensional computing inference[J]. IEEE Internet of Things Journal, 2022, 9(23): 24030-24037.

[2] LI T Z, CUI E F, WU Y T, et al. TeleVM: A lightweight virtual machine for RISC-V architecture [J]. IEEE Computer Architecture Letters, 2024, 23 (1): 121-124.

[3] 王旭东，朱蕴璞. 基于 ARM 多处理器的 CAN 总线分布式控制系统设计[J]. 国外电子测量技术，2015，34(5)：52-55.
WANG X D, ZHU Y P. Design of the CAN bus distributed control system based on multiprocessor[J]. Foreign Electric Measurement Technology, 2015, 34(5): 52-55.

[4] 肖景，杨会平，贺达江. 参数化的嵌入式乘法器测试技术研究[J]. 电子测量技术，2016，39(6)：98-101.
XIAO J, YANG H P, HE D J. Parameterized method for fault test of embedded multiplier[J]. Electronic Measurement Technology, 2016, 39(6): 98-101.

[5] 赵创，张为. 基于 HCORDIC 的浮点运算协处理器的设计[J]. 电子测量与仪器学报，2020，34(11)：58-65.
ZHAO CH, ZHANG W. Design of floating pointarithmetic coprocessor based on HCORDIC[J]. Journal of Electric Measurement and Instrumentation, 2020, 34(11): 58-65.

[6] PARK G, KUNG J, LEE Y. Simplified compressor and encoder designs for low-cost approximate radix-4 Booth multiplier[J]. IEEE Transactions on Circuits and Systems, 2023, 70(3): 1154-1158.

[7] 高嘉轩，刘鸿瑾，施博，等. 基于符号补偿的 RISC-V 处理器乘法器优化[J]. 计算机测量与控制，2023，31(7)：258-264，270.
GAO J X, LIU H J, SHI B, et al. Optimization of RISC-V processor multiplier based on sign compensation [J]. Computer Measurement and Control, 2023, 31(7): 258-264, 270.

[8] 李娅妮，郎世坤，王雅，等. 一种高效 16 位有符号数乘法器设计[J]. 集成电路与嵌入式系统，2024，24(6)：41-45.
LI Y N, LANG SH K, WANG Y, et al. Design and implementation of an efficient 16-bit signed number multiplier [J]. Integrated Circuits and Embedded Systems, 2024, 24(6): 41-45.

[9] 唐俊龙，汤孟媛，吴圳羲，等. 32 位 RISC-V 处理器中乘法器的优化设计[J]. 电子设计工程，2022，30(6)：61-65.
TANG J L, TANG M Y, WU ZH X, et al. Optimal design of multiplier in 32-bit RISC-V processor[J]. Electronic Design Engineering, 2022, 30(6): 61-65.

[10] CHENG Q, DAI L Y, HUANG M Q, et al. A low-power sparse convolutional neural network accelerator with pre-encoding radix-4 booth multiplier[J]. IEEE Transactions on Circuits and Systems, 2023, 70(6): 2246-2250.

[11] YEH W C, JEN C W. High-speed booth encoded parallel multiplier design[J]. IEEE Transactions on Computers, 2000, 49(7): 692-701.

[12] CHEN Y H. An accuracy-adjustment fixed-width booth multiplier based on multilevel conditional probability[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2015, 23(1): 203-207.

[13] AIZAZ Z, KHARE K. Area and power efficient truncated booth multipliers using approximate carry-based error compensation[J]. IEEE Transactions on Circuits and Systems, 2022, 69(2): 579-583.

[14] 汤孟媛. 基于 RISC-V 指令集微处理器的乘加单元的研究与设计[D]. 长沙：长沙理工大学，2022.
TANG M Y. Research and design of multiplier-accumulator unit based on RISC-V instruction set

microprocessor[D]. Changsha: Changsha University of Technology, 2022.

[15] 黄焘, 闰闰, 胡毅,等. 一种高能效基 4-Booth 编码并行乘法器设计[J]. 电子技术应用, 2023, 49(4): 117-122.
HUANG T, RUN R, HU Y, et al. An energy efficient radix-4 Booth encoding parallel multiplier design[J]. Electronic Technology Applications, 2023, 49(4): 117-122.

[16] ZHANG Z J, HE Y J. A low-error energy-efficient fixed-width booth multiplier with sign-digit-based conditional probability estimation [J]. IEEE Transactions on Circuits and Systems, 2018, 65(2): 236-240.

[17] 盛勇侠, 梁华国, 肖远,等. 基于新型压缩树的近似 Booth 乘法器[J]. 微电子学, 2022, 52(3): 425-430.
SHENG Y X, LIANG H G, XIAO Y, et al. An approximate booth multiplier based on novel wallace tree[J]. Microelectronics, 2022, 52(3): 425-430.

[18] 王佳乐, 胡越黎. 基于新型 booth 选择器和压缩器的乘法器设计[J]. 微电子学与计算机, 2020, 37(3): 5-8.
WANG J L, HU Y L. Design and implementation multiplier based on new booth selector and compressor[J]. Microelectronics and Computer, 2020, 37(3): 5-8.

作者简介

熊书伟,硕士研究生,主要研究方向为数字集成电路设计。

E-mail:515448453@qq.com

宋树祥(通信作者),教授,博士生导师,主要研究方向为集成电路设计,智能检测系统。

E-mail:songshuxiang@mailbox.gxnu.edu.cn