

基于改进 Wallace 树的 Posit 乘法单元优化

高志勇¹, 王磊¹, 刘博文², 英津瑞², 王盼龙²

(1. 郑州大学计算机与人工智能学院, 河南 郑州 450001; 2. 中原工学院网络空间安全学院, 河南 郑州 450007)

摘要: Posit 格式作为一种新的浮点数表示方法, 虽然在动态范围和舍入误差处理上相比 IEEE 754 浮点格式具有显著优势, 但其硬件实现尤其是尾数乘法器的设计存在挑战。为此, 通过增加特定的计数器、重新设计部分积求和阶段计数器布局以及改进最终求和阶段使用的加法器, 提出一种名为 3L-Wallace 树的改进 Wallace 树算法, 以减少部分积求和的阶段数, 降低硬件资源消耗和整体延迟。随后, 基于 3L-Wallace 树对 Posit 乘法单元进行了优化。此外, 还引入模块化设计方法, 将大位宽乘法器划分为更易于实现的小模块, 简化了设计过程并减小了实现难度。同时, 设计一种动态选择算法, 根据运行时尾数位宽动态选择合适位宽的乘法器, 避免硬件资源浪费。实验结果显示, 3L-Wallace 树算法硬件资源消耗相较于传统方法平均减少约 9.5%, 功率平均降低约 8.1%, 时延平均降低约 10.4%, 整体表现优于传统方法, 特别是在大位宽乘法器的实现上表现突出。

关键词: Posit 格式; Wallace 树; 乘法单元; 尾数乘法器; 计数器; 现场可编程门阵列

源代码链接: <https://github.com/zhiyongshuji/PositMultiplication-3L-WallaceTree.git>

中图分类号: TP332.2

文献标志码: A

DOI: 10.19678/j.issn.1000-3428.0070244

Optimization of Posit Multiplication Unit Based on Improved Wallace Tree

GAO Zhiyong¹, WANG Lei¹, LIU Bowen², YING Jinrui², WANG Panlong²

(1. School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, Henan, China;

2. School of Cyberspace Security, Zhongyuan University of Technology, Zhengzhou 450007, Henan, China)

【Abstract】 The Posit format, a novel floating-point representation, offers significant advantages over the IEEE 754 standard in terms of dynamic range and rounding error management. However, its hardware implementation, particularly the design of the mantissa multiplier, poses challenges. Therefore, this paper introduces an enhanced Wallace tree algorithm named 3L-Wallace tree, which reduces the number of stages in partial product summation, thereby decreasing both hardware resource consumption and overall latency. This improvement is achieved by adding specific counters, redesigning the layout of the partial product summation stage counters, and enhancing the adders used in the final summation stage. Furthermore, the paper implements the 3L-Wallace tree in the optimization of the Posit multiplication unit. Additionally, a modular design approach is introduced, dividing large bit-width multipliers into smaller, more manageable modules, thereby simplifying the design process and easing implementation difficulties. A dynamic selection algorithm is also designed, which dynamically selects multipliers of appropriate bit-width based on runtime mantissa width to avoid hardware resource waste. Experimental results show that the 3L-Wallace tree algorithm reduces hardware resource consumption by an average of 9.5%, power consumption by an average of 8.1%, and latency by an average of 10.4%, outperforming traditional methods, particularly in the implementation of large bit-width multipliers.

【Key words】 Posit format; Wallace tree; multiplication unit; mantissa multiplier; counters; Field-Programmable Gate Array (FPGA)

0 引言

浮点数通常以 IEEE 754-2008 标准格式表示^[1]。IEEE 754 标准已被广泛用于科学计算和基于实数的应用, 但此标准相比于 Posit 存在诸如舍入方式复杂和非数(NaN)浪费位模式等问题, 此外, 浮点数的舍入误差在大规模、长时程的计算中会产生累积, 导致计算结果不可靠甚至完全错误^[2]。由于在性能和准确性方面的要求不断提升, 一种有望替代 IEEE 754 浮点标准的更好的浮点数表示被提出。Gustafson 博士在 2017 年提出了一种新的浮点数表示格式, 即 Posit 格式^[3], 与现有的 IEEE 754 标准相比, Posit 具有更宽的动态范围、更好的精度、更小的舍入误差和对称的十进制精度等

基金项目: 中国科学院软件研究所合作项目(22001742)。

作者简介: 高志勇, 男, 硕士研究生, 主研方向为高性能计算; 王磊(通信作者), 教授; 刘博文、英津瑞、王盼龙, 硕士研究生。

收稿日期: 2024-08-13

修回日期: 2024-09-15

E-mail: zzu173920@sina.com

优势^[4-5]。此外,已经证明 n 位浮点加法器、乘法器可以安全地被 m 位 Posit 加法器、乘法器取代 ($m < n$)^[6]。

一些工作已经对 Posit 数字系统在某些领域的应用和带来的好处进行了研究,如神经网络、天气气候和航天模型等计算密集型应用^[7-8]。然而,关于使用它来代替浮点标准的说法仍存在一些争议^[8]。Posit 格式最常受到质疑的方面是其算术单元的硬件成本,Posit 算术单元是专门设计用于高效执行基于 Posit 数加、减、乘、除等基本运算的运算单元。相比于 IEEE 754 标准,Posit 引入了 regime 这一可变字段,这固有地增加了硬件方面的开销。再加上缺少开发这种格式的工具,这意味着 Posit 与 IEEE 754 格式相比仍是缺乏竞争力的^[9]。有关 Posit 算术单元的设计和成本的研究^[10-11]表明,乘法器和除法器是能耗最高的算术单元。

乘法器、除法器与 IEEE 754 浮点数乘法器的实现阶段相似,包括操作数的解码、尾数的乘法、指数部分的计算和结果的编码。尾数乘法器是迄今为止资源消耗最高的模块,因此,与浮点格式一样,降低尾数乘法器的复杂度对于优化整个 Posit 乘法单元的功耗至关重要。

一般乘法的计算过程由两个步骤组成:部分积的生成和部分积的求和。目前已知提升乘法器性能的方法,要么减少生成的部分积的数量,要么加速部分积的求和,也有方法综合了这两种方式。这些方法试图克服诸如面积、速度和功耗等方面的问题。Dadda 树^[12]、Booth 算法^[13-14]和 Wallace 树^[15]都是这类设计的例子,Wallace 树乘法器是最好的并行设计之一,其通过并行地添加部分乘积来减少乘法器的延迟。

为了降低 Posit 乘法单元的设计难度,减少 Posit 乘法单元的资源消耗,本文在前人工作的基础上,对 Posit 乘法单元的尾数乘法器进行了优化,主要工作如下:

1)在文献^[15]中提到的改进 Wallace 树的基础上增加了(5,3)计数器和(7,3)计数器,用于部分积的求和阶段,并且在部分积求和的初始阶段重新设计了几种计数器的使用布局,使得前期阶段能更多地减少元素的数量,从而减少部分积求和的阶段数,降低乘法器的硬件资源消耗和整体延迟。

2)提出一种模块化设计方法用于大位数乘法器的实现。将大位数乘法器划分为子模块,若子模块仍较宽,则进一步划分,直到子模块能较为方便地采用本文提出的 3L-Wallace 树算法实现。将此方法

应用于 Posit 尾数乘法器的设计阶段,能一定程度降低 Posit 乘法器的设计难度。

3)提出一种 Posit 尾数乘法器选择算法。该算法能根据 Posit 尾数位宽运行时变化的特性,动态选择合适位宽的乘法器进行生成,避免生成固定位宽乘法器而造成的硬件资源浪费。

1 相关工作

1.1 研究现状

从 Gustafson 教授提出 Posit 数字系统以来,在硬件实现层面,已有诸多工作开展。

CHAURASIYA 等^[16]开发了一个参数化的 Posit 算术硬件生成器,该生成器支持生成各种配置的 Posit 加法器和乘法器,显示出在面积和能效上优于 IEEE 754-2008 标准的 FPU,同时保持了较高的数值精度,为高性能和嵌入式计算系统提供了新的硬件实现方案。

RAVEENDRAN 等^[17]提出了一种参数化 Posit 算术硬件生成器,该生成器支持融合除法和平方根计算,能在单一硬件单元中同时处理这两种操作,显示出较传统 IEEE 754 实现更高的效率和速度。该设计已在现场可编程门阵列(FPGA)板上实现并验证,证明了其在硬件资源利用和处理速度上的优势。

文献^[1]提出了一种针对 Posit 数字系统的硬件实现方案,主要通过交替加减法替代常规的 Newton-Raphson 方法,实现了低成本的除法器和平方的平方根器。该方案在降低 FPGA 实现的硬件面积方面显示出优势,尤其适用于需要嵌入式低功耗设备的应用场景。

在研究 FPGA 上的 Posit 算术生成器方面,JAISWAL 等^[18]提出了一种开源算术硬件生成器,支持基本运算如浮点数与 Posit 间的转换及加减乘运算。该算法已在不同位宽的 FPGA 平台上实现,展示了其参数化的灵活性。通过这一开源解决方案,研究者可以更深入地探索 Posit 系统,推动 Posit 算术在各类应用中的测试与验证。

目前已有不少工作对 Posit 基本运算的实现进行了研究,但对于 Posit 单元中最耗费硬件资源的部分——Posit 乘法单元的尾数乘法器,鲜有针对其硬件实现进行优化的相关工作,因此本文尝试围绕此方面展开研究。

1.2 Posit 格式

Posit 定义为 Posit 位宽(N)和指数大小(E)的组合,即(N, E)。Posit 数可以用式(1)的形式表示,其中, s 为 Posit 的符号位,regime 域是基于相同

位($r \cdots r$)的长度 m 进行编码的,以 regime 域终止位 r 或 N 位 Posit 的末尾结束,regime 域的值由式(2)计算, e 为 Posit 的指数域,其余位为 Posit 的尾数位。Posit 格式中的 regime、指数和尾数字段在运行时会发生变化。注意,不需要区分 -0 和 $+0$,因为只有一个位模式($00 \cdots 0$)表示 0 。

$$P(E, N) =$$

$$s \quad rrr \cdots r \quad e_1 e_2 e_3 \cdots e_E \quad f_1 f_2 f_3 \cdots \quad (1)$$

$$k = \begin{cases} -m, & r=0 \\ m-1, & r=1 \end{cases} \quad (2)$$

Posit 的值由式(3)确定,其中, s 、 E_{exp} 和 f 分别是 Posit 的符号、真实指数和尾数,Posit 使用 2 的补码来表示负数。Posit 数与浮点数的主要区别是其使用的是无符号和无偏移的指数段,并且增加了 regime 字段。关于 Posit 格式的更多细节可以在文献[19]中找到。

$$P_v = \begin{cases} \text{NaR}, & P=100 \cdots 000 \\ 0, & P=000 \cdots 000 \\ (-1)^s \times 2^{E_{\text{exp}}} \times 1.f, & \text{其他} \end{cases} \quad (3)$$

1.3 Wallace 树

乘法运算大致可分为 3 个阶段:1)使用 AND 门阵列生成部分积;2)对部分积进行累加求和;3)对最终阶段的部分积求和^[20-22]。

传统 Wallace 树^[23]通过部分积分组求和的方式进行运算,在部分积求和的每个阶段,每 3 行分为一组,所有部分积分组后仍有一行或两行剩余,对剩余的部分不进行处理而是直接传递到下一阶段。前期所有的阶段都重复这一过程,直至部分积只有两行,最后使用传统的加法器对这两行进行求和^[24]。在部分积求和过程中对分组的处理是使用全加器或半加器进行求和,从而减少部分积的行数和其中的元素数。图 1 以 8×8 乘法器为例,说明了使用传统 Wallace 算法实现的乘法器中部分积求和的过程。

1.4 加法器

在乘法器部分积求和的最终阶段,需要使用加法器对最后的两行元素求和,即可得到最终的乘法运算结果,如图 1 阶段 5 所示。下面是对常见的两种加法器的简要介绍。

1.4.1 纹波进位加法器

纹波进位加法器(RCA)是最简单的加法器之一,其基本原理是将两个二进制数逐位相加,并将最低位的进位逐位向最高位传播。RCA 的优点是结构简单,易于实现,且消耗较少的硬件资源,但缺点也很明显:进位的传播会导致较大的延迟,尤

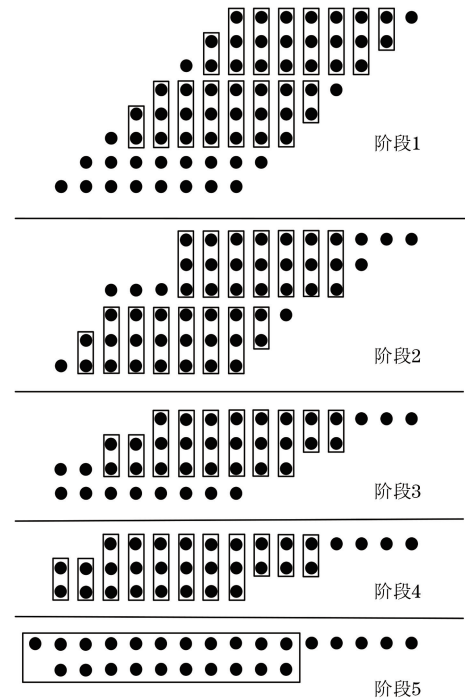


图 1 8×8 传统 Wallace 树部分积求和示意图

Fig.1 8×8 traditional Wallace tree partial product summation diagram

其对于大位数的加法,此缺点更加明显,因为进位是逐位传播的。

RCA 的基本工作原理是通过一系列的全加器(full adders)来逐位计算两个二进制数的和。每个全加器负责处理一个位的加法,并处理来自低位的进位。

全加器是实现 RCA 的核心组件,它有 3 个输入: A (加数的一位)、 B (加数的另一位)和 C_{in} (来自低一位的进位输入)。全加器产生两个输出: S (该位的和)和 C_{out} (产生的进位输出)。全加器的逻辑可以用式(4)表示:

$$\begin{cases} S = A \oplus B \oplus C_{\text{in}} \\ C_{\text{out}} = (A \& B) \mid (C_{\text{in}} \& (A \oplus B)) \end{cases} \quad (4)$$

在 RCA 中,每个全加器的进位输出 C_{out} 连接到下一位全加器的进位输入 C_{in} 。这种设计使得每一位的计算必须等待其前一位的进位计算完成,因此产生了“级联”效应。

1.4.2 超前进位加法器

超前进位加法器(CLA)旨在通过减少进位传播延迟来提高加法器的速度,其基本思想是提前计算出每一位的进位,而不是像 RCA 那样逐位进行进位传播。CLA 的主要原理包括以下几点:

1)进位计算。

CLA 的进位计算基于以下两个基本概念:

生成(G):如果某两个位的和直接产生进位,则

称这一对位会生成进位(例如 $1+1=10$)。

传递(P):如果某两个位的和的进位依赖于前一位的进位(即可能会传递进位),则称这一对位会传递进位(例如 $1+0=01$,如果前一位有进位则这一位也会有进位)。

这些“生成”和“传递”信号可以用逻辑表达式预先计算出来,具体表达式如式(5)所示:

$$\begin{cases} G_i = A_i \cdot B_i \\ P_i = A_i + B_i \end{cases} \quad (5)$$

式中: A_i 和 B_i 是加数的第 i 位。

2) 总体进位公式。

基于“生成”和“传递”信号,每一位的进位 C_i 可以预先计算出来,如式(6)所示:

$$C_{i+1} = G_i + (P_i \cdot C_i) \quad (6)$$

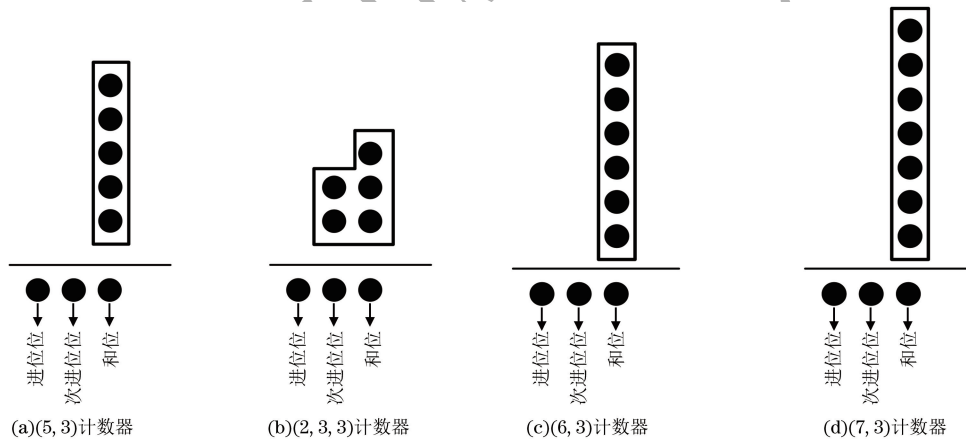


图 2 4 种计数器的示意图

Fig.2 Diagram of four types of counters

本文在(6,3)计数器基础上新增(7,3)计数器的原因是(6,3)计数器的最大计数值为6,二进制表示为(110)₂,而三位二进制数最大值为(111)₂,十进制值为7,因此理论上输出为三位二进制数的最优计数器为(7,3)计数器。同样是三位输出,(7,3)计数器比(6,3)计数器能多减少一位部分积的元素。

增加(5,3)计数器的原因是(5,3)计数器与(2,3,3)计数器输入均为5个部分积元素,输出均为3个元素,不同之处是(2,3,3)计数器横跨两列,(5,3)仅占据一列,如图2所示,即两种计数器的布局不同,新增(5,3)计数器能让部分积求和阶段使用更多的计数器。另外部分积求和过程还会用到全加器和半加器,其原理图如图3所示。

增加两种计数器,能让部分积求和的前期阶段尽可能多地使用计数器,最大程度地减少部分积的元素数量,最终达到减少部分积和阶段数的目的。

式中: C_i 表示第 i 位的进位输入,而 C_{i+1} 是该位产生的进位输出。

2 3L-Wallace 树算法

2.1 计数器对全加器与半加器的取代

传统的 Wallace 树在部分积的求和阶段每3行分为一组,对组中的每一列,三元素列(满3个元素)使用全加器,双元素列(仅有两个元素)则使用半加器。

本节讨论 3L-Wallace 树算法。在 ABED 等^[25]工作的基础上,本文参考(2,3,3)计数器的特点增加了(5,3)计数器,并在(6,3)计数器的基础上增加了(7,3)计数器。使用的最后一种计数器为(2,3,3)计数器。4种计数器的示意图如图2所示。

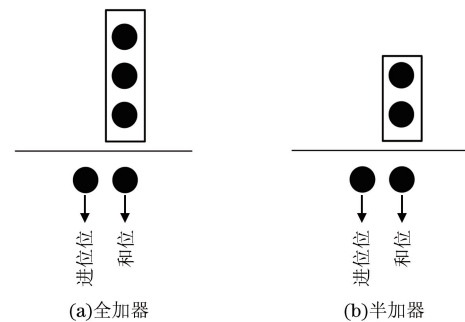


图 3 全加器和半加器的示意图

Fig.3 Diagram of full adder and half adder

2.2 基于计数器真值表的 always 语句直接实现方案

在 ABED 等^[25]的工作中,计数器是使用若干全加器和若干半加器以特定的组合连接方式实现的。如(6,3)计数器使用3个全加器和1个半加器构建。但这种实现方式有一个弊端是开发者难以确定全加器和半加器的使用数目以及两者的连接方式,增加了计数器的设计难度。

因此,本文抛弃了全加器、半加器的使用,采取了一种简单的思路。在 FPGA 开发平台上,以组合全加器、半加器方式实现的计数器综合后使用的逻辑硬件资源为查找表(LUT)。因此,结合计数器的真值表,本文使用 always 语句直接实现计数器的功能,这种实现方式简单明了,且最后综合的结果显示,该方法在 LUT 消耗和功率消耗上与前者的实现方法接近。以(6,3)计数器为例,两种实现方式的代码如下:

1)(6,3)计数器 always 语句实现代码如下:

```
module counter63(
    input wire [5:0] inputs;//6 位输入
    output reg [2:0] outputs;//3 位输出
);
//计算输入的所有位中 1 的数量
integer i, count;
always @( * ) begin
    count=0;
    //计数循环
    for (i=0; i<6; i=i+1) begin
        count=count+inputs[i];
    end
    //根据输入中 1 的数量结合真值表决定输出
    case (count)
        0: outputs = 3'b000;
        1: outputs = 3'b001;
        2: outputs = 3'b010;
        3: outputs = 3'b011;
        4: outputs = 3'b100;
        5: outputs = 3'b101;
        6: outputs = 3'b110;
        default: outputs = 3'b000;
    endcase
end
endmodule
```

2)使用全加器和半加器实现(6,3)计数器的代码如下:

```
module counter63(
    input wire [5:0] inputs;//6 位输入
    output wire [2:0] sum;//3 位输出
);
//中间变量,用于全加器和半加器的输出
wire [1:0] sum1, sum2;
wire carry1, carry2, carry3;
//使用 3 个全加器和 1 个半加器,三者以特定的方
//式连接
full_adder fa1(//第一个全加器
    .a(inputs[0]),
```

```
    .b(inputs[1]),
    .cin(inputs[2]),
    .sum(sum1[0]),
    .cout(carry1)
);
full_adder fa2(//第 2 个全加器
    .a(inputs[3]),
    .b(inputs[4]),
    .cin(inputs[5]),
    .sum(sum2[0]),
    .cout(carry2)
);
half_adder ha1(//半加器
    .a(sum1[0]),
    .b(sum2[0]),
    .sum(sum[0]),
    .cout(carry3)
);
full_adder fa3(//第 3 个全加器
    .a(carry1),
    .b(carry2),
    .cin(carry3),
    .sum(sum[1]),
    .cout(sum[2])
);
endmodule
```

使用本文提出的方法简化了计数器的实现过程,能够在一定程度上提高 Posit 尾数乘法器的开发效率。

2.3 混合进位加法器在部分积求和最后阶段的应用

本文综合前面两种进位加法器(RCA 和 CLA)的优点,融合两种进位加法器,提出了改进的混合进位加法器(HCLA),其原理如图 4 所示(彩色效果见《计算机工程》官网 HTML 版,下同)。

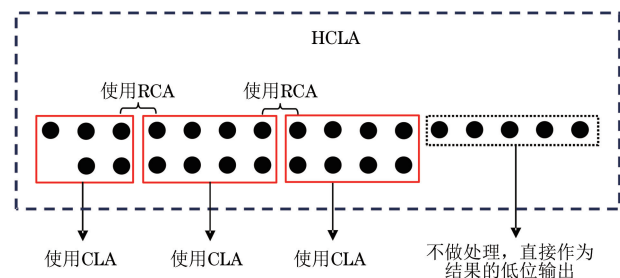


图 4 HCLA 示意图

Fig.4 Diagram of HCLA

由于 RCA 需要逐位传播来自低位的进位,而 CLA 是直接使用当前位计算进位,不对来自低位的进位进行传播,因此综合两种加法器的优缺点,将最后阶段的求和设计成分组的形式,每组处理若干位。

这样可以平衡硬件复杂性和速度。每组内部使用 CLA 计算进位,而组之间使用更慢的 RCA 方法来进一步传递进位,形成混合的加法器设计。求和时其中一个加数可能缺少一些低位,因此可以直接取另一个加数对应位置的数作为结果的低位输出,而无需参与求和运算。

以图 4 为例,该图为 8×8 乘法器最后阶段求和的示意图,整体位宽为 16 位,其中一个加数低位缺少 5 位,因此低 5 位直接输出,剩余 11 位参与运算。余下的 11 位每 4 位分为一组,不足 4 位的单独成一组,组内使用 CLA,组间使用 RCA。

2.4 3L-Wallace 树

以 8×8 乘法器为例,3L-Wallace 树算法的部分积求和流程如图 5 所示。对比图 1 可以看到改进的 Wallace 树在阶段 1、阶段 2 大量使用了本文提到的 4 种计数器,并且相较于传统 Wallace 树的实现方法,其部分积的求和过程减少了一个阶段。

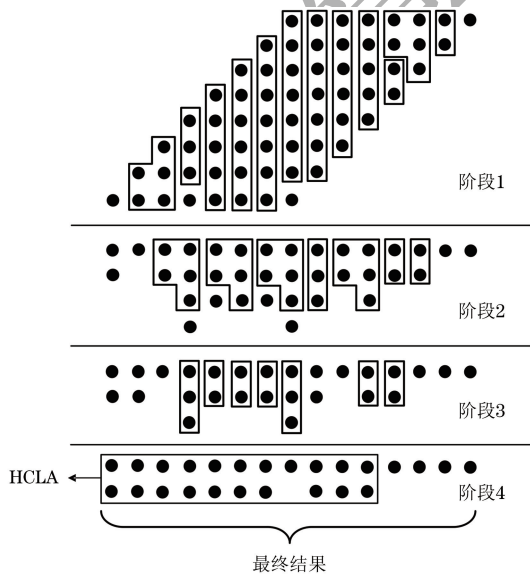


图 5 8×8 3L-Wallace 树部分积求和过程示意图

Fig.5 Diagram of the 8×8 3L-Wallace tree partial product summation process

针对 Wallace 树改进算法的具体实现流程,本文提出了 3 个普适性的改进原则:

1) 在部分积求和的前期阶段,布局好计数器的使用位置后,将部分积分为左右两个部分,左半部分为高位权部分,右半部分为低位权部分,对于高位权部分余下的某些双元素列,不使用半加器对其进行求和,同样的情形在低位权部分中则使用半加器。理由是:半加器输入输出均为两个元素,并不会为减少部分积元素作贡献。部分积求和过程使用半加器,目的仅仅是将本列的元素转移到相邻的高位权列,起到将元素左移的作用。这样做的好处是,逐阶

段累积下来,当来到最终的求和阶段时,右侧会生成较宽的单元列部分,而这部分可以直接作为最终结果的低位进行输出,右侧单元列越宽,则余下需要进行求和的列越窄,耗费的硬件资源也就越少,如图 6 所示。

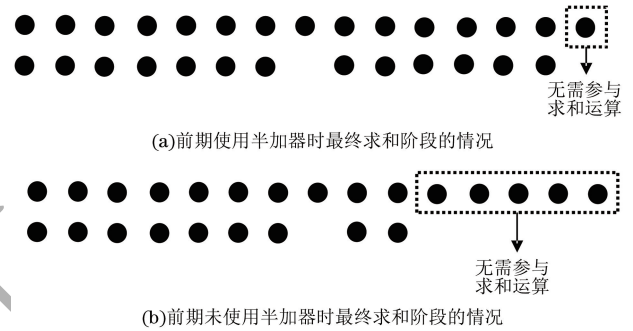


图 6 半加器使用与否最终求和阶段情况对比图

Fig.6 Comparison diagram of the final summation stage with and without using half adders

2) 部分积求和的倒数第 2 个阶段不再使用计数器,原因是经过前面几个阶段的处理后,此阶段仅有少数列为三元素列,不再适合计数器的使用。

3) 对于最右侧的双元素列,当有 3 个以上连续的双元素列时,不再遵循原则 1),如图 7 所示。原因是经过本文的实验验证,当双元素列超过 3 列时,其产生的负面作用已经超过其对最终求和阶段起到的正面作用。

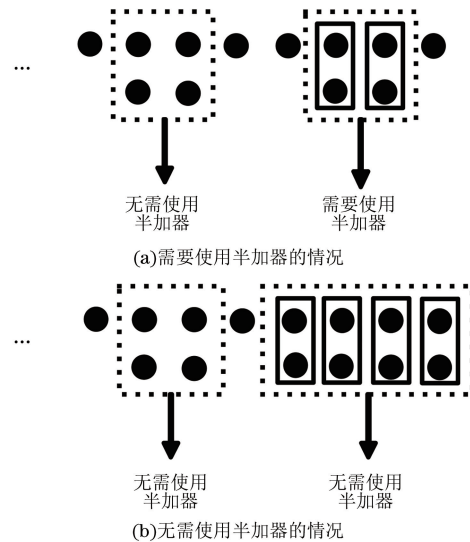


图 7 无需使用半加器和需要使用半加器的情况

Fig.7 Scenarios without and with using half adders

3 Posit 尾数乘法器的优化

3.1 Posit 尾数乘法器的模块化设计

在采用 Wallace 树算法实现乘法器的过程中,随着乘法器输入位宽的增加,乘法器设计的难度也会不断提升。原因是输入位宽越大,则生成的

部分积规模越大,这就导致:1)部分积求和的前期阶段计数器的使用布局较难设计;2)由于部分积行数的增多,达到一定程度时,部分积求和的阶段数也会增加。由于使用 Wallace 算法设计乘法器时,部分积求和的所有阶段都需要开发人员手动设计,上面两种情况无疑给设计增添了很大的难度。

出于降低开发难度、提升开发效率的目的,本文参考文献[25],对于大位数乘法器的设计,采取了模块化的设计方式。以 16 位尾数乘法器的设计为例,输入为 X 、 Y ,其中, X 可以分为高 8 位部分 X_H 、低 8 位部分 X_L , Y 同理,其模块化设计原理如图 8 所示。

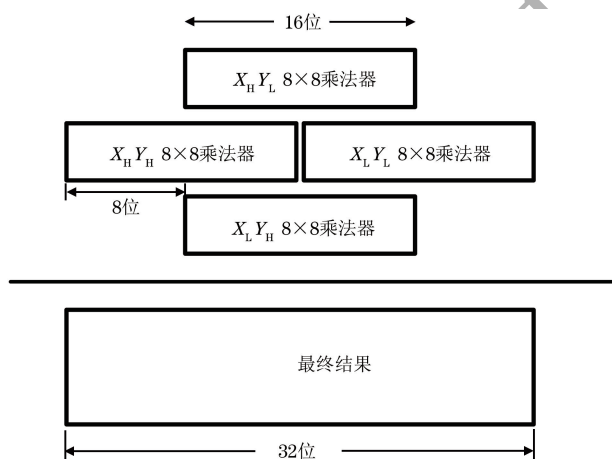


图 8 16 位尾数乘法器模块化设计示意图

Fig.8 Diagram of the modular design of a 16 bit mantissa multiplier

设 n 为位宽,当 n 为奇数时,意味着输入不能划分为位宽相等的高位部分和低位部分。本文采取的方法是在输入的最高位添 0,对其进行一位扩展,使其位宽为 $n+1$ 位偶数位。这样做既能保证输入的值不发生改变,又能方便乘法器内部进行模块化处理。以 15 位尾数乘法器为例,对输入 X 、 Y 均进行一位 0 扩展并进行高低位的划分,使 X 划分为高 8 位 X_H 、低 8 位 X_L , Y 同理,其模块化设计原理如图 9 所示。

由于两个输入最高位都扩展了一位 0,导致结果的高位扩展了两位 0,结果输出之前需要进行高位截零处理,这样就能保证输出正确的结果。

对于输入位宽为奇数时的处理方式,虽然可能会引入一些微小的额外硬件资源消耗,但在设计上带来的简化效果却是显而易见的。因为如果输入位宽为奇数,那么采用模块化设计不便于进行高低位的划分,并且如果划分后高低位部分的位宽不等,则子模块还会产生乘数位宽不同的乘法运算,从而给

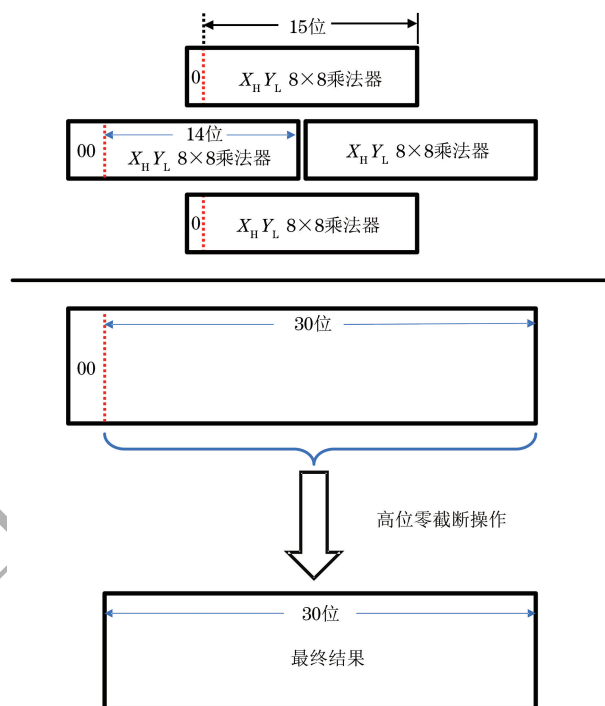


图 9 15 位尾数乘法器模块化设计示意图

Fig.9 Diagram of the Modular design of a 15 bit mantissa multiplier

乘法器的设计增加较大的难度。

对于 n 位的大位数乘法器,本文的做法如下:先将其划分为 4 个位宽为 $n/2$ 的乘法器作为子模块;而对 $n/2$ 位宽的子模块,如果其位宽仍比较大,可以以 $n/4$ 位宽的乘法器为次子模块对其进行模块化处理。自顶向下重复上面的模块化设计方法,直到最底层的模块能够较为简单地使用 3L-Wallace 树算法实现。

3.2 1~32 位尾数乘法器的设计

对于 1~32 位位宽范围内乘法器的设计,本文采取如下的设计方法:

对于 1~5 位乘法器的实现,由于乘法器的位宽较短,使用 3L-Wallace 算法设计综合的结果与采用 Verilog 语法自带的“*”运算符设计综合的结果相比在硬件资源消耗和功率消耗等方面的表现并不会明显的提升,因此此时使用 Verilog 语法自带的乘法运算符“*”进行乘法器的简单实现无疑是更好的选择。

对于 6~32 位乘法器的实现,本文采取如下的实现策略:当 $5 < n < 11$ 时,直接使用改进的 Wallace 树算法实现;当 $n \geq 11$ 时,使用前面介绍的模块化方法;对于 $n \geq 11$ 的情况,参考表 1。

6~32 位乘法器最终都是使用表 1 中最左边一列的次子模块乘法器来实现的。对于表中没有出现的奇数位宽乘法器,先对其进行 0 扩展,扩展

表 1 6~32 位尾数乘法器模块化实现参考表
Table 1 Reference table for modular implementation of
6 to 32 bit mantissa multipliers 单位:位

$n/4$ 位次子模块 乘法器位宽	$n/2$ 位子模块 乘法器位宽	n 位大位数 乘法器的位宽
6	12	24
7	14	28
8	16	32
9	18	—
10	20	—

为偶数位宽,然后再依照表 1 进行子模块的划分。若子模块的位宽仍未出现在表中,则再重复上面的操作,最后依据表 1 进行子模块的实现;对于表中未出现的偶数位宽乘法器,要先对其进行子模块划分,然后进行 0 扩展后再依据表 1 进行子模块的实现。

在进行大位数乘法器设计时,其子模块在进行更低一级的模块划分时,也有可能进行高位零扩展。以设计 21 位乘法器为例,位宽 $n_0=21$ 为奇数,扩展后位宽为 22 位,子模块位宽 $n_1=(n_0+1)/2=11$ 位,为奇数;对子模块进行进一步划分时, n_1 需要先扩展到 12 位,然后将其划分为位宽 $n_2=6$ 位的次子模块,此时可以直接使用本文提出的 3L-Wallace 树算法进行实现。

3.3 尾数乘法器选择算法

对于 6~10 位尾数乘法器,本文直接使用改进的 Wallace 树算法进行了实现,后续将其作为大位数乘法器的子模块或次子模块。本文在文献[17]的基础上,对 Posit 乘法器的尾数乘法部分进行了改进。由于 Posit 数的尾数宽度是运行时变化的,在尾数进行乘法运算时,在硬件层面进行优化或选择适当位宽的乘法器能较大程度上节约硬件资源。因此,本文对 Posit 的尾数运算设计了一个动态乘法器选择算法,其伪代码如算法 1 和算法 2。

算法 1 尾数乘法器选择算法 MulSel

输入 2 个 Posit 数较宽尾数的位数 n

输出 位宽为 n 的尾数乘法器

```
1. if  $n < 6$  then//判断是否  $n < 6$ 
2. VerGen( $n$ );//使用 Verilog 中的“*”运算符实现//乘法器
3. else if  $6 \leq n < 11$  then//判断是否  $6 \leq n < 11$ 
4. WalTreeSel( $n$ );//使用算法 2,输入为  $n$ 
5. else if  $(n+1)/2 \geq 11$  then//判断是否  $(n+1)/2 \geq 11$ 
6. WalTreeSel( $(n+1)/4$ );//使用算法 2,输入为  $(n+1)/4$ 
7. else WalTreeSel( $(n+1)/2$ );//使用算法 2,输入为
```

$(n+1)/2$

算法 2 3L-Wallace 树乘法器选择算法 WalTreeSel

输入 指定位宽 n

输出 3L-Wallace 树乘法器

```
1. if  $n \% 2 == 0$  then//判断  $n$  是否为偶数
2. WalMul( $n$ );//选择使用位宽为  $n$  的 3L-Wallace//树乘法器
3. else {ZeroExtend();//进行高位零扩展
4. WalMul( $n+1$ );//选择使用位宽为  $(n+1)$  的//3L-Wallace 树乘法器
5. ZeroTrunc();//对乘法器的输出结果进行高位 0//截断
```

算法开始时,首先利用算法 1 根据不同的尾数位宽选择不同的乘法器。当 $n < 6$ 时,算法直接选择 Verilog 自带的乘法运算符进行尾数乘法器的实现;当 $6 \leq n < 11$ 时,利用算法 2 直接根据尾数的位宽选择已经实现好的特定位宽的 3L-Wallace 树乘法器;当 $n \geq 11$ 时,需要先进行一次子模块划分,子模块位宽 $n_1=(n+1)/2$,然后判断 n_1 的位宽,若 $n_1 < 11$,则利用算法 2 选择指定的乘法器,若 $n_1 \geq 11$,则需要对子模块进行一次模块划分得到次子模块,次子模块的位宽 $n_2=(n_1+1)/4$,最后使用算法 2 进行乘法器的选择。

4 实验与结果分析

本节将对前面提到的算法进行实验验证。本文的算法是在一台 CPU 型号为 Intel i7-8565U、运行频率为 1.80 GHz、基于 Windows 10 平台、内存为 8 GB 的 64 位机器上运行的。算法使用 Verilog 硬件描述语言实现,并采用 Vivado 2018.3 工具进行模拟。

4.1 3L-Wallace 树算法和 Posit 尾数乘法器选择算法的实验和评估

本节对前面提到的两种算法进行实验评估。分别用传统 Wallace 树和本文提出的 3L-Wallace 树实现 6~10 位尾数乘法器,然后使用 Vivado 工具对乘法器进行综合,采用两种方法实现的尾数乘法器 LUT 消耗情况、动态功率和时延情况如图 10 所示。

分析上面的实验结果,可以发现随着乘法器位宽的增加,3L-Wallace 树算法相比传统 Wallace 树算法能减少更多的硬件资源消耗,平均减少约 12.7%,原因可能是位宽越大,乘法器部分积求和阶段就越能使用到更多的计数器。图 10(a)、图 10(b)的分布基本上一致,这是因为功率消耗与硬件资源消耗是呈正相关的。对于这 5 种位宽的尾数乘法

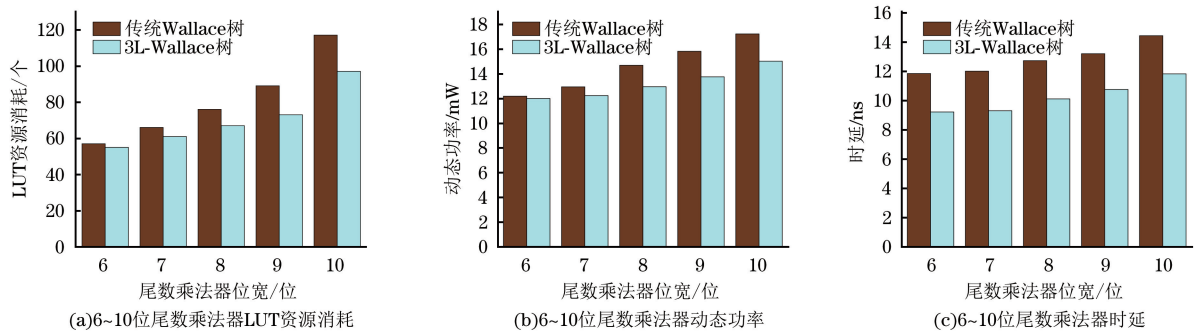


图 10 6~10 位尾数乘法器综合结果

Fig. 10 Synthesis results of 6 to 10 bit mantissa multipliers

器, 3L-Wallace 树算法与传统 Wallace 树相比时延相差基本相等, 3L-Wallace 树算法平均约减少 2.2 ns 的时延。

随后, 结合本文提到的 Posit 尾数乘法器选择算法, 将前面实现的尾数乘法器模块整合到文

献[25]的开源代码中, 作为尾数乘法运算的一部分。由于 3L-Wallace 树算法实现的乘法器位宽为 6~10 位, 因此采用 Posit(16, 2) 格式进行实验。使用不同位宽尾数乘法器实现的 Posit 乘法单元的硬件资源 LUT 消耗、动态功率和时延情况如图 11 所示。

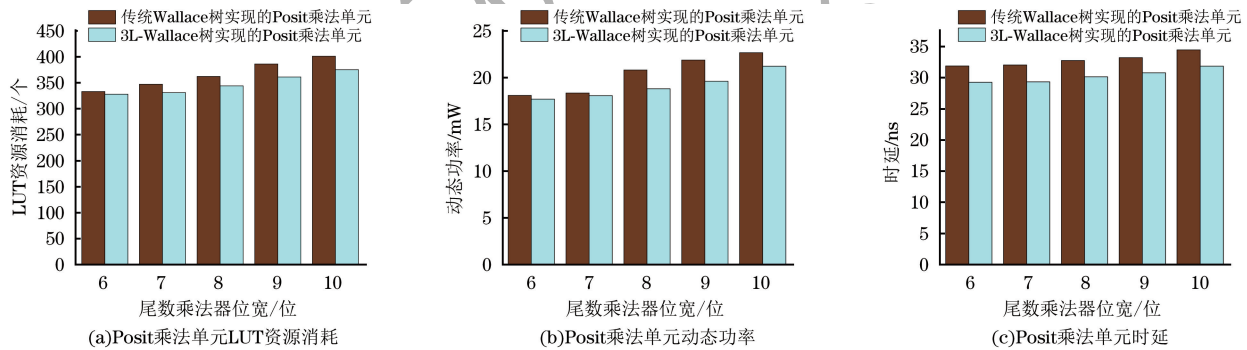


图 11 6~10 位宽尾数乘法器实现的 Posit 乘法单元综合结果

Fig. 11 Synthesis results of posit multiplication units implemented with 6 to 10 bit mantissa multipliers

分析实验结果, 可以发现对于 Posit 乘法器, 使用 3L-Wallace 树算法与使用传统的 Wallace 树算法相比, 硬件资源消耗减少约 6.8%, 动态功率降低约 7.3%, 时延降低约 8.2%。原因是 Posit 乘法器还有编解码、regime 域的计算和指数域的计算等模块。实验结果的分布情况整体和前面实验的结果相似。

4.2 大位宽尾数乘法器的实验和评估

本节对以模块化方式实现的尾数乘法器进行实

验评估。对于前面提到的大位宽尾数乘法器, 分别以传统 Wallace 树尾数乘法器为子模块和以 3L-Wallace 树尾数乘法器为子模块进行实现, 然后使用 Vivado 对大位宽尾数乘法器进行综合, 使用两种方法实现的乘法器的逻辑资源 LUT 消耗、动态功率和时延情况如图 12 所示。

对上面的实验结果进行分析, 当 $n < 21$ 时, 对于 n 位奇数位宽尾数乘法器, 其与 $(n+1)$ 位偶数位宽尾数乘法器消耗同样数量的 LUT, 并且动态功

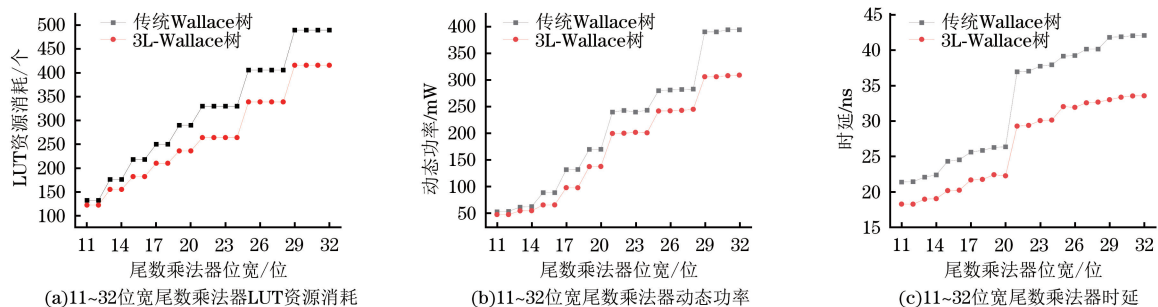


图 12 11~32 位宽尾数乘法器综合结果

Fig. 12 Synthesis results of 11 to 32 bit mantissa multipliers

率和时延也相同。原因是 n 位奇数位宽乘法器会对输入进行高位 0 扩展,对输出进行高位 0 截断,而这样的操作是不消耗 LUT 的,并且其消耗的功率和产生的时延也是可以忽略不计的,除此之外与 $(n+1)$ 位偶数位宽乘法器无其他不同。因此 n 位奇数位宽乘法器本质上是 $(n+1)$ 位偶数位宽乘法器。

当 $n \geq 21$ 时,从 $n=21$ 开始,每 4 个为一组,每一组内 4 种位宽乘法器在资源消耗、动态功率消耗和时延方面的表现基本一样,原因是最底层使用的子模块都一样,因此本质上为同一种乘法器。以第一组为例,对 $n=21, 22, 23, 24$ 这 4 种乘法器按照表 1 进行子模块划分,最底层使用的均为 6×6 乘法器。

观察延迟折线图, $n=20$ 到 $n=21$ 这段折线时延陡增,这是因为位宽 $n \leq 20$ 的乘法器只需一次子模块的划分,而位宽 $n > 20$ 的乘法器则需两次子模块划分,模块化操作会给整体引入一定的延迟。

另外,上下两条折线的差值从左往右整体上是逐渐增大的,这意味着随着乘法器位宽的增大,3L-Wallace 树带来的优化效果会更加明显。

本节实验中,Posit 除尾数乘法模块外的其余模块(如解码、指数域计算、编码等模块)基本未作改动,因此这些模块所消耗的 LUT 资源、动态功率等指标在全体实验中基本是保持不变的。由于实验 1 已对尾数乘法器在 Posit 乘法单元中的应用进行了实验评估,本节可以直接参考实验 1 的结果,故不再对这部分内容进行实验。

4.3 实验结果综合分析

综合前面的几组实验,可以发现本文提出的方法对于 Posit 乘法单元的优化取得了较为可观的效果。相比于传统 Wallace 树,将本文提出的 3L-Wallace 树实现的尾数乘法器运用到 Posit 乘法单元,结合本文提出的 Posit 尾数乘法器动态选择算法,综合前面几组实验的结果,LUT 消耗平均减少约 9.5%,功率平均降低约 8.1%,时延平均降低约 10.4%。

5 结束语

为了优化 Posit 乘法单元中的尾数乘法器,本文采用 Wallace 树算法,并在传统 Wallace 树的基础上对其进行了改进,使用 3 种计数器代替全加器、半加器,并对部分积求和最后阶段使用的传统进位加法器进行了改进,提出了混合进位加法器;之后本文又对大位数尾数乘法器进行了实现,为

了降低大位数尾数乘法器的实现复杂度,本文采用模块化的方式;最后提出了 Posit 尾数乘法器选择算法应用到 Posit 乘法单元中。本文对 Posit 乘法运算尾数部分的改进为 Posit 乘法单元的硬件实现及优化提供了一种新的思路,但实验仅针对 Posit 尾数乘法器进行了优化,并且本文提出的一些算法只在 Vivado 平台进行了模拟,实际表现有待进一步的实验验证。鉴于 FPGA 具有可并行工作、可编程的特点^[26],未来会考虑将本文的工作在 FPGA 板上进行进一步的实验验证,另外,也会考虑对 Posit 乘法单元的其他模块或 Posit 其他算术单元进行优化。

参考文献

- [1] MARKSTEIN P. The new IEEE-754 standard for floating point arithmetic[EB/OL]. [2024-02-10]. <https://drops.dagstuhl.de/storage/16dagstuhl-seminar-proceedings/dsp-vol08021/DagSemProc.08021.20/DagSemProc.08021.20.pdf>
- [2] 关明晓,刘嘉莹,张鸿锐,等.基于 FPGA 误差可控的浮点运算加速器研究[J].计算机工程,2024,50(5):291-297.
GUAN M X, LIU J K, ZHANG H R, et al. Research on FPGA-based floating-point accelerator with controllable error[J]. Computer Engineering, 2024, 50(5): 291-297. (in Chinese)
- [3] 许瑾晨,曹浩,李典,等. Posit:一种挑战浮点计算标准的新数据类型的研究[J].信息工程大学学报,2021,22(4):427-432.
XU J C, CAO H, LI D, et al. Posit: a study of a new data type challenging the floating-point computation standard[J]. Journal of Information Engineering University, 2021, 22(4): 427-432. (in Chinese)
- [4] UGUEN Y, FORGET L, DINECHIN F. Evaluating the hardware cost of the posit number system[C]//Proceedings of the 29th International Conference on Field Programmable Logic and Applications (FPL). Washington D. C., USA: IEEE Press, 2019: 106-113.
- [5] GUSTAFSON J L, YONEMOTO I T. Beating floating point at its own game: Posit arithmetic[J]. Supercomputing Frontiers and Innovations, 2017, 4(2): 71-86.
- [6] KLÖWER M, DÜBEN P D, PALMER T N. Posits as an alternative to floats for weather and climate models[C]//Proceedings of the Conference for Next Generation Arithmetic. Washington D. C., USA: IEEE Press, 2019: 1-8.
- [7] ERCEGOVAC M D, LANG T. Digital arithmetic[M]. Amsterdam, the Netherlands: Elsevier, 2004.
- [8] DINECHIN F, FORGET L, MULLER J M, et al. Posits: the good, the bad and the ugly[C]//Proceedings of the Conference for Next Generation Arithmetic. Washington D. C., USA: IEEE Press, 2019: 1-10.
- [9] MURILLO R, BARRIO A A, BOTELLA G, et al. PLAM: a Posit logarithm-approximate multiplier[J]. IEEE Transactions on Emerging Topics in Computing, 2021, 10(4): 2079-2085.
- [10] JAISWAL M K, SO H K H. PACoGen: a hardware posit arithmetic core generator[J]. IEEE Access, 2019, 7: 74586-74601.
- [11] DADDA L. Some schemes for parallel multipliers[J]. Alta Frequenza, 1965, 34, 349-356.
- [12] BOOTH A D. A signed binary multiplication technique[J].

- The Quarterly Journal of Mechanics and Applied Mathematics, 1951, 4(2): 236-240.
- [13] 范文兵, 周健章. 基于 Radix-4 Booth 编码的并行乘法器设计[J]. 郑州大学学报(工学版), 2025, 46(1): 26-33.
- FAN W B, ZHOU J Z. Design of parallel multiplier based on Radix-4 Booth encoding[J]. Journal of Zhengzhou University (Engineering Edition), 2025, 46(1): 26-33. (in Chinese)
- [14] MURILLO R, BARRIO A A, BOTELLA G. Customized posit adders and multipliers using the FloPoCo core generator[C]//Proceedings of 2020 IEEE International Symposium on Circuits and Systems (ISCAS). Washington D. C., USA: IEEE Press, 2020: 1-5.
- [15] ABED S, MOHD B J, AL-BAYATI Z, et al. Low power Wallace multiplier design based on wide counters [J]. International Journal of Circuit Theory and Applications, 2012, 40(11): 1175-1185.
- [16] CHAURASIYA R, GUSTAFSON J, SHRESTHA R, et al. Parameterized Posit arithmetic hardware generator [C]//Proceedings of the 36th International Conference on Computer Design (ICCD). Washington D. C., USA: IEEE Press, 2018: 334-341.
- [17] RAVEENDRAN A, JEAN S, MERVIN J, et al. A novel parametrized fused division and square-root POSIT arithmetic architecture [C]//Proceedings of the 33rd International Conference on VLSI Design and the 19th International Conference on Embedded Systems (VLSID). Washington D. C., USA: IEEE Press, 2020: 207-212.
- [18] JAISWAL K B, KUMAR N, SESHADRI P, et al. Low power Wallace tree multiplier using modified full adder[C]//Proceedings of the 3rd International Conference on Signal Processing, Communication and Networking (ICSCN). Washington D. C., USA: IEEE Press, 2015: 1-4.
- [19] GUSTAFSON J. Posit arithmetic[EB/OL]. [2024-02-10]. <https://posithub.org/docs/Posits4.pdf>.
- [20] LU M. Arithmetic and logic in computer systems [M]. [S. l.]: John Wiley & Sons, 2005.
- [21] 张旭. 基于 FPGA 的定浮点一体 FPU 设计[D]. 济南: 山东大学, 2023.
- ZHANG X. Design of fixed-point and floating-point integrated FPU based on FPGA [D]. Jinan: Shandong University, 2023. (in Chinese)
- [22] SOLANKI V, DARJI A D, SINGAPURI H. Design of low-power Wallace tree multiplier architecture using modular approach [J]. Circuits, Systems, and Signal Processing, 2021, 40(9): 4407-4427.
- [23] SUREKA N, PORSELVI R, KUMUTHAPRIYA K. An efficient high speed Wallace tree multiplier[C]//Proceedings of 2013 International Conference on Information Communication and Embedded Systems (ICICES). Washington D. C., USA: IEEE Press, 2013: 1023-1026.
- [24] FADAVI-ARDEKANI J. M * N Booth encoded multiplier generator using optimized Wallace trees [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 1993, 1(2): 120-125.
- [25] ABED S, KHALIL Y, MODHAFFAR M, et al. High-performance low-power approximate Wallace tree multiplier [J]. International Journal of Circuit Theory and Applications, 2018, 46(12): 2334-2348.
- [26] 杨思捷, 陈俊奇, 王勇, 等. 基于 FPGA 的软硬件协同纠错编码加速方案[J]. 计算机工程, 2024, 50(2): 224-231.
- YANG S J, CHEN J Q, WANG Y, et al. FPGA-based software and hardware cooperative acceleration scheme of erasure code encoding [J]. Computer Engineering, 2024, 50(2): 224-231. (in Chinese)

文字编辑 金胡考
栏目编辑 宋 圆