基于 Karatsuba 和 Vedic 算法的快速单精度浮点乘法器

易清明[1,2*]，符清杆[1]，石 敏[1]，骆爱文[1]，陈嘉文[1]

(1. 暨南大学信息科学技术学院 广州 510632；2. 泰斗微电子科技有限公司 广州 510663)

【摘要】针对现有的单精度浮点乘法器存在运算速度慢的问题，该文设计了一种融合 Karatsuba 算法和 Vedic 算法两者优点的快速单精度浮点乘法器。该文利用 Karatsuba 算法减少单精度浮点乘法器的乘法运算次数，将 24 bit 尾数的乘法运算分解为少位数乘法运算，获得基于 3 bit 和 4 bit 的尾数乘法架构；进一步地，利用 Vedic 算法对单精度浮点乘法器的尾数乘法架构进行优化，利用复杂度低、速度快的加法器实现了 Karatsuba 算法分解后的 3 bit 和 4 bit 的两个基本乘法运算，提高了运算速度。仿真及 FPGA 验证结果表明，该文设计的单精度浮点乘法器相对于基于传统的 Karatsuba 算法的单精度浮点乘法器、基于 Vedic 算法的单精度浮点乘法器，其最大运行时钟频率分别提高了约 5 倍和 2 倍。

关 键 词 Karatsuba 算法； 乘法运算； 最大运行时钟频率； 单精度浮点乘法器； Vedic 算法

中图分类号 TP332 **文献标志码** A doi:10.12178/1001-0548.2020161

A Fast Single-Precision Floating-Point Multiplier Based on Karatsuba and Vedic Algorithms

YI Qing-ming[1,2*], FU Qing-gan[1], SHI Min[1], LUO Ai-wen[1], and CHEN Jia-wen[1]

(1. School of Information Science and Technology, Jinan University Guangzhou 510632;
2. Taidou Microelectronic Science and Technology Co., Ltd. Guangzhou 510663)

Abstract To deal with the slow operation speed in the existing single-precision floating-point multiplier, a fast Karatsuba-based single-precision floating-point multiplier which combines the advantages of Karatsuba algorithm with the Vedic algorithm is designed in this paper. The fast Karatsuba-based multiplier decreases the multiplication-operation times of the conventional single-precision floating-point multiplier by splitting the multiplication of 24-bit mantissa into that of fewer mantissa. An improved multiplication architecture composed of the 3-bit and 4-bit mantissa is constructed and further optimized by employing the Vedic algorithm. The 3-bit and 4-bit multipliers are respectively achieved by the corresponding adders with low complexity and fast speed, leading to faster processing speed. The results of simulation and FPGA verification imply that the designed single-precision floating-point multiplier achieves approximately 5 times and 2 times higher performance in the maximum operating clock frequency, comparing to the conventional Karatsuba-based and the Vedic-based single-precision floating-point multiplier, respectively.

Key words Karatsuba algorithm; multiplication-operation; maximum operating clock frequency; single-precision floating-point multiplier; Vedic algorithm

单精度浮点数乘法运算的耗时主要集中在 24 bit 的尾数相乘部分。为了提高单精度浮点数乘法运算的速度，研究者提出了各种单精度浮点乘法器的改进方法[1-8]。文献 [1] 提出了基于 Karatsuba 算法[9-10] 的改进设计，通过数学运算的公式变换，用加法器代替乘法器，相比于 24 bit 尾数直接相乘的单精度浮点乘法器的设计，该设计减少了 3 个乘法器，只使用了 6 个乘法器。但是乘法器的使用限制了单精度浮点乘法器运算速度的进一步提高。文献 [2] 提出了一种基于 Vedic 算法[3-4] 的单精度浮点乘法器设计方法，该设计沿用了 24 bit 尾数直接相乘的设计思路，通过使用 Vedic 算法设计 3 bit 的乘法器，迭代复用，实现 24 bit 的尾数相乘运算，从而避免乘法器的使用，提高单精度浮点乘法器的运算速度。但是简单使用 Vedic 算法设计单精度浮点乘法器，导致 Vedic 算法设计的 3 bit 乘法器

收稿日期：2020－04－01；修回日期：2020－10－26
基金项目：中央高校基本科研业务费专项 (21620353)
作者简介：易清明 (1965－)，女，博士，教授，主要从事信号处理、图像、SoC 开发等方面的研究. E-mail：tyqm@jnu.edu.cn

迭代次数过多，硬件资源增多。文献 [3] 沿用了 Vedic 算法的思想，利用流水线设计方法对单精度浮点乘法器结构进行优化，同时减少单精度浮点乘法器的组合逻辑延时，提高单精度浮点乘法器的运算速度，但是同样存在 Vedic 算法设计的 3 bit 乘法器迭代次数过多，硬件资源增多的问题；文献 [5] 在 Vedic 算法的基础上，通过全加器的复用来简化单精度浮点乘法器的设计结构，提高单精度浮点乘法器的运行速度，但是依然无法满足目前的运算需求。

以上文献提出的改进方法对单精度浮点乘法器的运算速度均有一定的提高，但是人工智能的兴起对浮点数乘法运算的速度提出了更高的要求，数以百万计的浮点数乘法运算造成深度学习的训练时间过长，限制了人工智能的研究与实用性。为了进一步提高单精度浮点乘法器的运算速度，本文利用 Vedic 算法改进了 Karatsuba 算法，并把该算法应用于单精度浮点乘法器设计。

1　浮点数乘法原理

本文设计的单精度浮点乘法器基于 IEEE754 标准[11]。单精度浮点数的表示格式由 4 字节组成，分为 3 个部分。第 1 部分为符号位，占 1 bit；第二部分为阶码部分，占 8 bit；第三部分为尾数部分，占 23 bit。一个实数 H 的浮点数表示形式为：

$$H=(-1)^{C}ZR^{E} \tag{1}$$

式中，C 表示符号位；Z 表示尾数部分，在浮点数运算时要补上隐藏的‘1’，转换成{1, Z}；E 表示阶码，在实数转换为浮点数格式时要减去相应的偏移量。

单精度浮点数乘法运算可以简单分为 4 个步骤。

1) 两个乘数的符号位进行异或运算。根据异或的结果决定最后计算结果的正负性。若符号位异或的结果为 1，则浮点数相乘的结果是负数，若异或的结果为 0，则浮点数相乘的结果是正数。

2) 两个乘数的阶码直接相加，其结果再减去偏移量 Bias 得到相乘后结果的阶码，Bias 的值为 127。

3) 尾数部分补上隐藏的‘1’，执行尾数相乘操作，得到相乘的结果 V。

4) 步骤 3) 相乘结果 V 的最高位的值决定标准化结果尾数的取值。若最高位为 1，则取 V[46:24] 位作为最后标准化结果的尾数；若最高位是 0，则取 V[45:23] 位作为最后标准化结果的尾数。联合符号位的值、阶码的值和尾数的值到两个浮点数相乘的结果。

浮点数乘法运算的流程如图 1 所示。Ha、Hb 表示 32 位的浮点数，Za、Zb 为 23 bit 的尾数。Ha 的符号位与 Hb 的符号位进行异或运算得到符号位部分运算后的结果；Ha 的阶码加上 Hb 的阶码再减去相应的偏移量 (Bias) 得到阶码部分运算后的结果；尾数在高位补上隐藏的‘1’，分别得到 24 bit 的乘数{1, Za}、{1, Zb}。24 bit 的乘数相乘得到 48 bit 的结果 V，其最高位的值决定浮点数相乘结果尾数的取值。联合符号位部分运算后的结果、阶码部分运算后的结果及尾数部分标准化的结果得到浮点数相乘的结果。由于尾数部分的标准化需要依据 V 最高位的值，所以尾数相乘的耗时决定了浮点乘法器的运算速度。

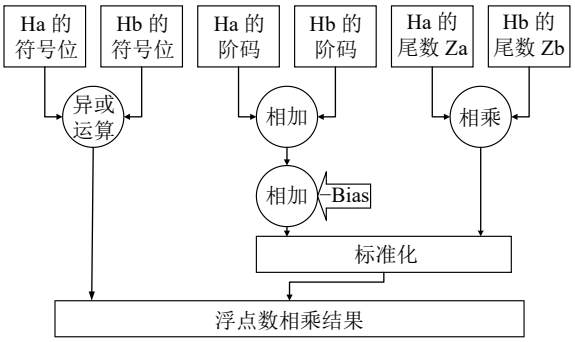

图 1　浮点数乘法运算的流程框图

2　Karatsuba 算法

传统单精度浮点乘法器中的尾数相乘实现方法如下所示[1]：

$$W=W_2 2^{2n}+W_1 2^{n}+W_0 \tag{2}$$

$$X=X_2 2^{2n}+X_1 2^{n}+X_0 \tag{3}$$

$$\begin{aligned}WX=&W_2X_2 2^{4n}+(W_2X_1+W_1X_2)2^{3n}+\\&(W_2X_0+W_0X_2+W_1X_1)2^{2n}+\\&(W_0X_1+W_1X_0)2^{n}+W_0X_0\end{aligned} \tag{4}$$

式中，n 取值为 8；W、X(W={1, Za}, X={1, Zb}) 为 24 bit 的尾数；W_2、W_1、W_0 分别表示 W 的高 8 位、中 8 位、低 8 位；X_2、X_1、X_0 分别表示 X 的高 8 位、中 8 位、低 8 位。传统的单精度浮点乘法器设计把尾数 W 和 X 直接相乘，由式 (4) 可知，得到 24 bit 尾数相乘的结果需要 9 个 8 位×8 位的乘法器和 8 个加法器。由于乘法运算比加法运算耗时长，且电路结构更加复杂，所以传统的单

精度浮点乘法器效率比较低且运算速度慢。Karatsuba 算法用加法器来替代乘法器，从而减少乘法器的使用，达到简化电路结构，提高运算速度的目的。基于 Karatsuba 算法的 24 bit 尾数相乘的实现方法具体如下所示[1]：

$$\alpha_2 = W_2X_2 \tag{5}$$

$$\alpha_1 = W_1X_1 \tag{6}$$

$$\alpha_0 = W_0X_0 \tag{7}$$

$$\beta_2 = (W_2 + W_1)(X_2 + X_1) - \alpha_2 - \alpha_1 \tag{8}$$

$$\beta_1 = (W_2 + W_0)(X_2 + X_0) - \alpha_2 - \alpha_0 \tag{9}$$

$$\beta_0 = (W_1 + W_0)(X_1 + X_0) - \alpha_1 - \alpha_0 \tag{10}$$

根据式 (5)～(10) 可以得到 24 bit 尾数相乘的结果，如下所示：

$$WX = \alpha_2 2^{4n} + \beta_2 2^{3n} + (\beta_1 + \alpha_1)2^{2n} + \beta_0 2^n + \alpha_0 \tag{11}$$

由式 (11) 可得，相比于传统方法实现的 24 bit 尾数相乘，基于 Karatsuba 算法实现的 24 bit 尾数相乘使用的乘法器数量更少，由之前的 9 个乘法器减少到 6 个乘法器，而仅增加了 9 个加法器。

3 本文设计

针对基于 Karatsuba 算法设计的单精度浮点数乘法器依然存在乘法器，限制了运算速度的提升，本文对原 Karatsuba 算法进行了改进，并将该改进算法应用于单精度浮点乘法器设计。具体地，本设计在将 24 bit 乘法分解为 8 bit 乘法和9 bit 乘法的基础上，再次复用 Karatsuba 算法，将 8 bit 和 9 bit 的乘法运算进一步分解，使得原有的 24 bit 乘法运算被分解为 3 bit 和 4 bit 的乘法运算，从而使用更少的低位宽乘法器来实现更高比特的乘法运算，可以减少乘法器的使用，进一步简化电路结构，达到提高运算速度的目的。另一方面，本设计同时结合 Vedic 算法设计一种基于加法运算的乘法器，替换 Karatsuba 尾数乘法架构中的 3 bit 和 4 bit 的乘法器，从而利用少量低比特加法器实现乘法运算，降低硬件资源消耗和降低关键延时，提高单精度浮点乘法器运算速度。如图 2 所示，单精度浮点乘法器由数据分割处理模块、符号位运算模块、阶码运算模块、尾数运算模块、标准化模块等 5 个模块组成。

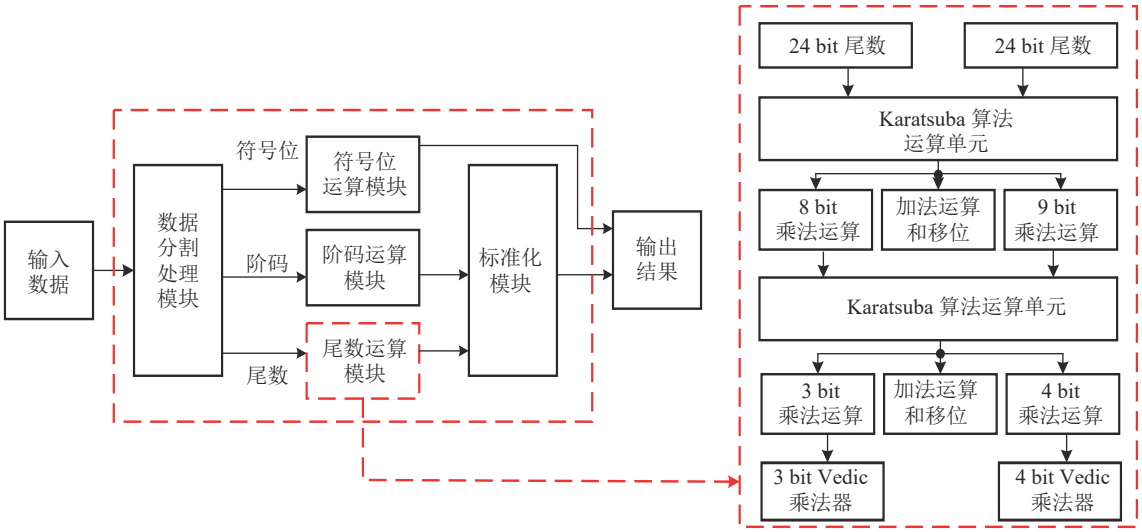

图 2 浮点数乘法器的整体结构

如图 2 所示，本文对 24 bit 的尾数运算 (尾数部分在最高位隐藏了一个“1”，在计算浮点数乘法时要补上“1”) 进行改进。本文的设计步骤主要分为 3 个阶段：

1) 将 24 bit 的乘法运算通过 Karatsuba 算法分解为 8 bit 的乘法器、9 bit 的乘法器、加法器和移位寄存器[1]；

2) 再次利用 Karatsuba 算法对 8 bit 乘法器和 9 bit 乘法器进行分解，分解成 3 bit 的乘法器和 4 bit 的乘法器、加法器和移位寄存器；

3) 使用 Vedic 算法设计的 3 bit Vedic乘法器和 4 bit Vedic 乘法器来代替阶段 2) 的 3 bit 乘法器和 4 bit 乘法器。

以下是各个阶段的具体实现方法。

阶段 1)：由式 (11) 可知，将 24 bit 的乘法运算分解为 8 bit 和 9 bit 乘法运算；阶段 2)：对式 (5)～(10) 中的乘法运算进行分解，拆分为 3 bit 和 4 bit 的乘法运算。式 (5)～(7) 中的乘法运算是 8 bit 的乘法运算，式 (8)～(10) 中的乘法运算是 9 bit 的乘法运算。9 bit 乘法运算的分解方法和 24 bit 乘法运算的分解方法一样，把输入进行划分，n 的取值变为 3。8 bit 乘法运算的分解方法为：

$$W_2 = W_{21}2^n + W_{22} \tag{12}$$

$$X_2 = X_{21}2^n + X_{22} \tag{13}$$

$$W_2X_2 = W_{21}X_{21}2^{2n} + (W_{21}X_{22} + W_{22}X_{21})2^n + W_{22}X_{22} \tag{14}$$

式中，W_{21}、W_{22}、X_{21}、X_{22} 分别是 W_2、X_2 的高 4 位和低 4 位，此时 n 的取值为 4。阶段 3)：为了进一步提高单精度浮点乘法器的运算速度，利用 Vedic 算法设计的 3 bit Vedic 乘法器和 4 bit Vedic 乘法器来代替 Karatsuba 算法分解后的 3 bit、4 bit 的乘法器。按照 Vedic 算法的思想使用加法器来搭建乘法器时使用了流水线设计方法。

Vedic 算法可通过加法器来实现乘法运算，其具有运算简单，实现的乘法运算速度快、效率高、硬件资源消耗少的特点[12]。基于 Vedic 算法设计 3 bit Vedic 乘法器流程如图 3 所示。

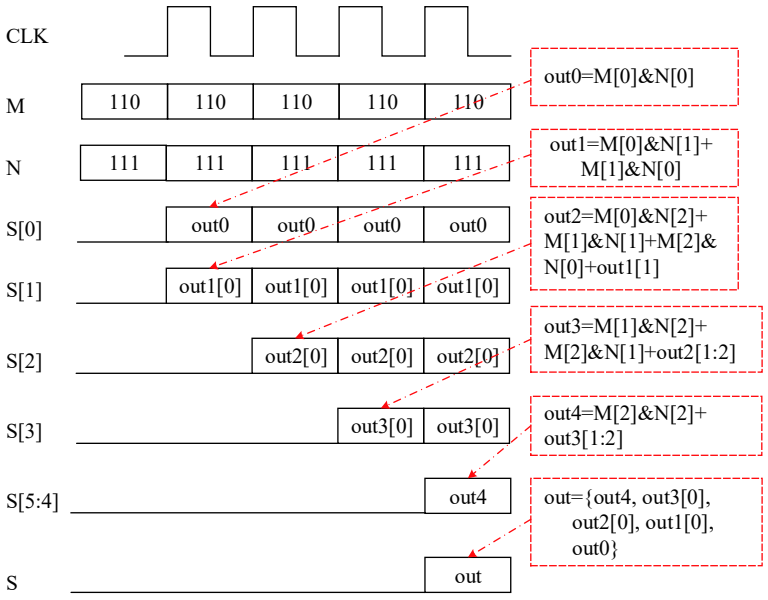

图 3　Vedic 算法设计 3 bit 乘法器的流程图

如图 3 所示，M 是二进制的 $(110)_2$，N 是二进制的 $(111)_2$。M_2、M_1、M_0 分别代表 M 的高 1 位、中间 1 位、低 1 位，N_2、N_1、N_0 分别代表 N 的高 1 位、中间 1 位、低 1 位。运算步骤如下：

1) M_0 与 N_0 相与的结果 out0 作为输出结果的最低位 S[0]；

2) M_1 与 N_0 相与的结果再加上 M_0 与 N_1 相与的结果得到值 out1，将 out1[0] 作为输出结果的次低位 S[1]；

3) out1[1] 作为加数与 M_0 和 N_2 相与的值、M_1 和 N_1 相与的值、M_2 和 N_0 相与的值相加得到结果 out2，将 out2[0] 作为输出结果的 S[2]；

4) out2[1:2] 作为加数与 M_1 和 N_2 相与的值、M_2 和 N_1 相与的值相加得到的结果 out3，将 out3[0] 作为输出结果的 S[3]；

5) out3[1:2] 作为加数与 M_2 和 N_2 相与的值相加得到结果 out4，将 out4 作为输出结果的 S[5:4]。拼接 S[5:4]、S[3]、S[2]、S[1]、S[0] 的值得到输出结果 S。

4 bit Vedic 乘法器设计流程与 3 bit Vedic 乘法器设计流程类似。如图 4 所示，把式 (8) 中的 W_2 加 W_1 的结果 A 分割为高 3 位 A_2、中间 3 位 A_1、低 3 位 A_0，X_2 加 X_1 的结果 B 分割为高 3 位 B_2、中间 3 位 B_1、低 3 位 B_0。按照 Karatsuba 算法的思想可以通过 3 bit 的 Vedic 乘法器 (VM 表示 Vedic multiplier) 和 4 bit 的 Vedic 乘法器实现 9 bit 的乘法运算。9 bit 乘法器的整体设计框如图 4 所示。

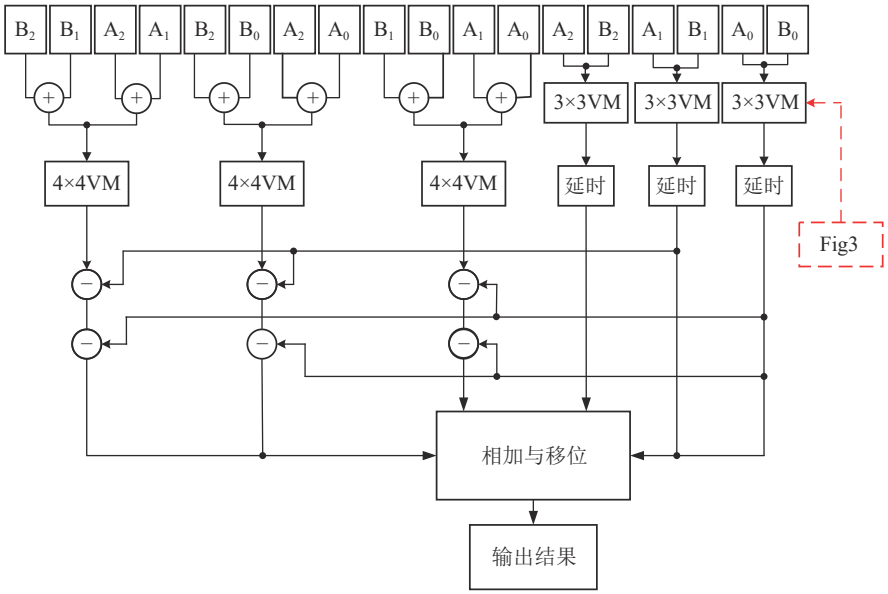

图 4　9 bit 乘法器的硬件设计框图

同理，按照 Karatsuba 算法的思想通过 8 bit 乘法器和 9 bit 乘法器实现 24 bit 尾数相乘运算。24 bit 乘法器的整体设计框如图 5 所示。

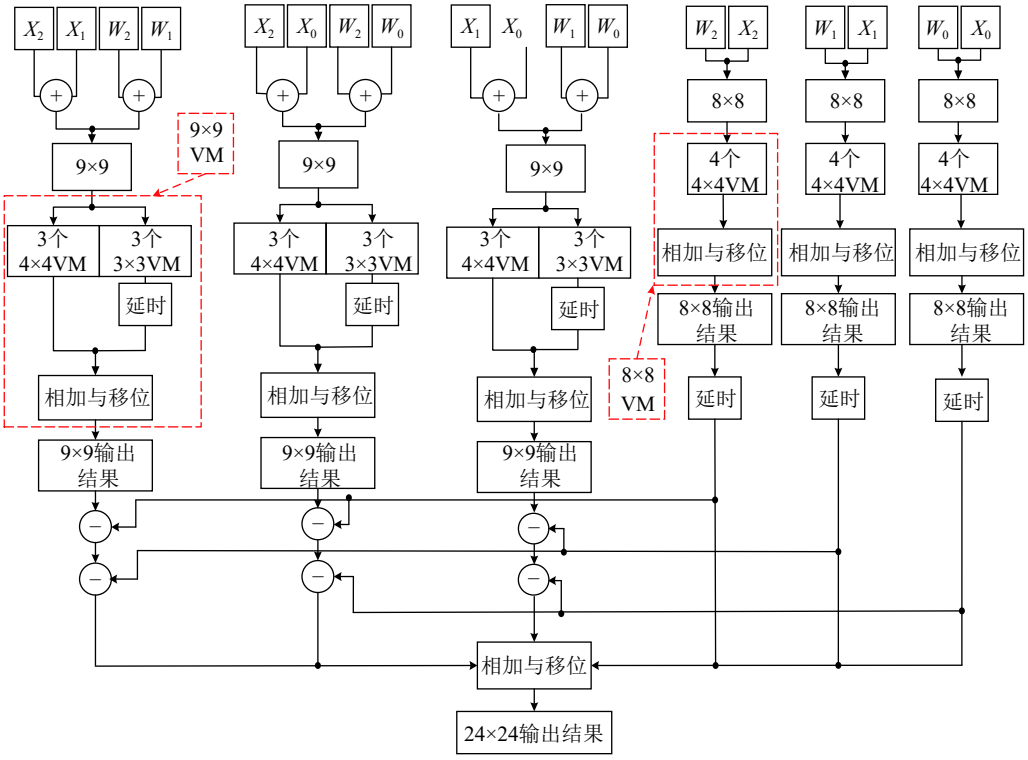

图 5　24 bit 乘法器的硬件设计框图

由图 5 可知，得到 24 bit 尾数相乘的结果仅需要 21 个 4 bit Vedic 乘法器和 9 个 3 bit Vedic 乘法器。相比于文献 [3] 的 64 个 3 bit Vedic 乘法器，本文设计减少了 Vedic 乘法器的迭代次数。

4　算法性能仿真与验证

本文设计的单精度浮点乘法器在 Modelsim 平台上进行仿真。随机产生 3000 组 32 位浮点数测试数据 ain 和 bin，执行完乘法运算后，在输出数据有效指示信号 dout_vld 为高电平时得到输出结果 dout。

图 6 给出了本文设计的单精度浮点乘法器的一个测试案例的仿真结果。由仿真结果可知，输入 ain 以十六进制表示为 3f000000 (即十进制实数

0.5)，输入 bin 以十六进制表示为 3f400000 (即十进制实数 0.75)，输出结果 dout 以十六进制表示为 3ec00000 (即十进制实数 0.375)。根据仿真波形可得，单精度浮点乘法器的输出结果需要延时 5 拍时钟，仿真结果证明了本文所提出的单精度浮点乘法器设计方法的正确性。

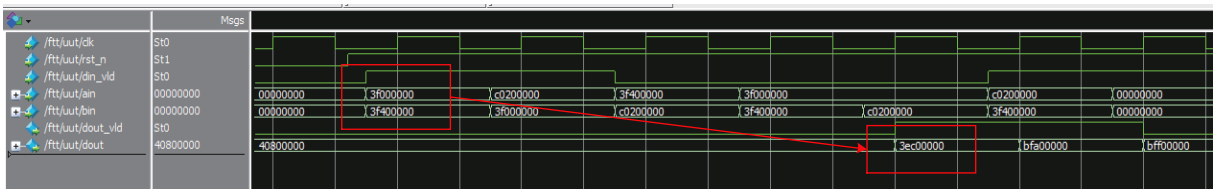

图 6　浮点数乘法器的仿真结果

为了进一步证明本文设计的有效性，利用 Quartus II 64-bit 13.0.1 平台在相同的芯片 Cyclone II EP2C5F256C6、Cyclone IV EP4CE6F17C6 上综合。表 1 给出了本文设计的单精度浮点乘法器与文献 [1] 算法、文献 [3] 算法设计的单精度浮点乘法器及 Altera 公司开发的浮点数乘法 IP 核的性能比较。

表 1　5 种浮点乘法器的性能参数对比

器件类型	实现方法	逻辑单元(LE)使用数/逻辑单元总数(使用率/%)	寄存器使用数/寄存器总数(使用率/%)	LUTs	功耗/mW	最大时钟频率/MHz	延时拍数
Cyclone II EP2C5F256C6	基于三级流水线Karatsuba算法[1]	1 147/4 608(25)	440/4 608(10)	1 079	41.06	59.11	8
	基于并行全加器Vedic算法[3]	1877/4 608(48)	1 145/4 608(25)	1 767	40.73	148.28	9
	浮点数乘法IP核	917/4 608(20)	530/4 608(12)	864	40.59	145.79	5
	二次复用Karatsuba算法	**1 682/4 608(37)**	**381/4 608(8)**	**1 592**	**40.88**	**73.86**	**6**
	Karatsuba算法+Vedic算法	**1 846/4 608(40)**	**815/4 608(18)**	**1 691**	**40.38**	**313.58**	**5**
Cyclone IV EEP4CE6F17C6	基于三级流水线Karatsuba算法[1]	1 157/6 272(18)	440/6 272(7)	1 091	84.30	62.31	8
	基于并行全加器Vedic算法[3]	1 883/6 272(30)	1 145/6 272(18)	1 776	84.48	159.06	9
	浮点数乘法IP核	922/6 272(15)	530/6 272(8)	868	77.27	185.74	5
	二次复用Karatsuba算法	**1 695/6 272(27)**	**381/6 272(6)**	**1 592**	**73.29**	**80.89**	**6**
	Karatsuba算法+Vedic算法	**1 870/6 272(30)**	**815/6 272(13)**	**1 597**	**75.27**	**331.35**	**5**

在相同的芯片 Cyclone II EP2C5F256C6、Cyclone IV EP4CE6F17C6 下，文献 [1] 设计的单精度浮点乘法器最大时钟频率分别为 59.11 MHz、62.31 MHz。文献 [1] 消耗的资源较少，是因为沿用了资源消耗少但工作频率低的乘法器。文献 [3] 设计的单精度浮点乘法器最大运行时钟频率分别为 148.28 MHz、159.06 MHz。相比于文献 [1]，文献 [3] 设计的单精度浮点乘法器的最大运行时钟频率提高了 2.5 倍，但是逻辑单元、寄存器及 LUTs 使用数量分别平均增多了约 730 个、700 个和 690 个。本文二次复用 Karatsuba 算法把文献 [1] 的 9 bit 和 8 bit 乘法运算分解为 3 bit 和 4 bit 乘法运算。本文二次复用 Karatsuba 算法实现的单精度浮点乘法器的最大时钟频率分别增大到 73.86 MHz、80.89 MHz。二次复用 Karatsuba 算法的实验结果证明，文献 [1] 设计的单精度浮点乘法器的最大时钟频率受限于乘法器。本文融合 Karatsuba 算法和 Vedic 算法两者优点设计的单精度浮点乘法器输出结果需要延时 5 拍时钟，相比于文献 [1] 的 8 拍时钟、文献 [3] 的 9 拍时钟，延时更短，响应时间更快。本文融合 Karatsuba 算法和 Vedic 算法优点实现的单精度浮点乘法器的最大时钟频率是相对最优的，分别达到 313.58 MHz、331.35 MHz。相比于文献 [1]、文献 [3]、浮点数乘法 IP 核，本文融合 Karatsuba 算法和 Vedic 算法两者优点设计的单精度浮点乘法器的最大时钟频率分别平均提高了 5.30 倍、2.10 倍、1.97 倍。本文融合 Karatsuba 算法和 Vedic 算法优点设计的单精度浮点乘法器比文献 [3] 的逻辑单元、寄存器及 LUTs 分别平均降低了 22 个、330 个、127 个，比文献 [1] 的逻辑单元、寄存器及 LUTs 分别平均增加了 706 个、375 个、559 个，比浮点数乘法 IP 核的逻辑单元、寄存器及 LUTs 分别平均增加了 938 个、285 个、778 个，但是在硬件资源可以接受的范围下获得了更高的最大时钟频率，降低了输出延时，提高了运算速度。同时本文融合 Karatsuba 算法和 Vedic 算法优点设计

的单精度浮点乘法器的功耗分别是 40.38 mW、75.27 mW。在芯片 Cyclone II EP2C5F256C6 下，本文设计的单精度浮点乘法器相比于文献 [1]、文献 [3]、浮点数乘法 IP 核分别降低了 0.68 mW、0.35mW、0.21mW。在芯片CycloneIVEP4CE6F17C6 下，本文设计的单精度浮点乘法器相比于文献 [1]、文献 [3]、浮点数乘法 IP 核分别降低了 9.03 mW、9.21 mW、3.0 mW。本文所提出的设计方法在需要计算大量数据的人工智能领域更具有优势，更能满足计算实时性的需求，具有一定的实用价值。

5 结 束 语

本文优化了 Karatsuba 算法并利用该算法设计了基于 IEEE754 标准的单精度浮点乘法器。设计充分利用了 Karatsuba 算法可以减少乘法器数量的优点和 Vedic 算法可以用简单的加法器实现乘法功能而门延迟和面积增加很缓慢的优点，通过二次复用 Karatsuba 算法来减少乘法器的数量，降低单精度浮点乘法器复杂度；再通过 Vedic 算法设计的 3 bit、4 bit Vedic 乘法器来替换本文设计中需要用到的 3 bit 乘法器和 4 bit 乘法器，并在 Vedic 算法设计的 3 bit 乘法器、4 bit 的乘法器中运用了流水线设计方法来优化结构，提高设计的运行时钟频率、运算效率及运算速度。在相同的平台和芯片型号下，本文设计的单精度浮点乘法器的最大时钟频率相比于文献 [1]、文献 [3]、浮点数乘法 IP 核分别平均提高了 5.30 倍、2.10 倍、1.97 倍。

本文研究工作得到羊城创新创业领军人才支持计划的资助 (2019019)，在此表示感谢。

参 考 文 献

[1] MEHTA A, BIDHUL C B, JOSEPH S, et al. Implementation of single precision floating point multiplier using Karatsuba algorithm[C]//2013 Int Conf on Green Computing, Communication and Conservation of Energy (ICGCE). [S.l.]: IEEE, 2013: 254-256.

[2] PALDURAI K, HARIHARAN K. FPGA implementation of delay optimized single precision floating point multiplier[C]//2015 Int Conf on Advanced Computing and Communication Systems. [S.l.]: IEEE, 2015: 1-5.

[3] HAVALDAR S, GURUMURTHY K S. Design of Vedic IEEE 754 floating point multiplier[C]//2016 IEEE Int Conf on Recent Trends in Electronics, Information & Communication Technology (RTEICT). [S.l.]: IEEE, 2016: 1131-1135.

[4] MAHAKALKAR S S, HARIDAS S L. Design of high performance IEEE754 floating point multiplier using Vedic mathematics[C]//2014 Int Conf on Computational Intelligence and Communication Networks. [S.l.]: IEEE, 2014: 985-988.

[5] GOWREESRINIVAS K V, SAMUNDISWARY P. Comparative performance analysis of multiplexer based single precision floating point multipliers[C]//2017 Int Conf of Electronics, Communication and Aerospace Technology (ICECA). [S.l.]: IEEE, 2017, 2: 430-435.

[6] KODALI R K, GUNDABATHULA S K, BOPPANA L. FPGA implementation of IEEE-754 floating point karatsuba multiplier[C]//2014 Int Conf on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). [S.l.]: IEEE, 2014: 300-304.

[7] PALEKAR S, NARKHEDE N. High speed and area efficient single precision floating point arithmetic unit[C]//2016 IEEE Int Conf on Recent Trends in Electronics, Information & Communication Technology (RTEICT). [S.l.]: IEEE, 2016: 1950-1954.

[8] JAIN A, DASH B, PANDA A K, et al. FPGA design of a fast 32-bit floating point multiplier unit[C]//2012 International Conf on Devices, Circuits and Systems (ICDCS). [S.l.]: IEEE, 2012: 545-547.

[9] MONTGOMERY P L. Five, six, and seven-term Karatsuba-like formulae[J]. IEEE Trans on Computers, 2005, 54(3): 362-369.

[10] WEIMERSKIRCH A, PAAR C. Generalizations of the Karatsuba algorithm for efficient implementations[J]. IACR Cryptology ePrint Archive, 2006, 2006: 224.

[11] ZURAS D, COWLISHAW M, AIKEN A, et al. IEEE standard for floating-point arithmetic[J]. The Institute of Electrical and Electronics Engineers, 2008, 754(2008): 1-70.

[12] POORNIMA M, PATIL S K, SHIVUKUMAR S K P, et al. Implementation of multiplier using vedic algorithm[J]. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 2013, 2(6): 219-223.

编 辑 刘飞阳