分离通路浮点乘加器设计与实现

何 军 黄永勤 朱 英
(上海高性能集成电路设计中心 上海 201204)

摘 要 针对传统浮点融合乘加器会增加独立浮点加减法、乘法等运算延迟的缺点,首先设计并实现了一种分离通路浮点乘加器SPFMA,通过分离乘法和加法通路,在保持融合乘加运算延迟6拍延迟不变的情况下,将独立乘法和加法等运算延迟由6拍减为4拍,克服了传统融合乘加器的缺点。然后经专用工艺单元库逻辑综合评估,SPFMA可工作在1.2GHz以上,面积60779.44um^2。最后在硬件仿真加速器平台上运行SPEC CPU2000浮点测试课题对其进行性能评估,结果表明所有浮点课题性能均有所提高,最大提高5.25%,平均提高1.61%,证明SPFMA可进一步提高浮点性能。

关键词 浮点加法,浮点乘法,融合乘加,分离通路,浮点性能,运算延迟
中图法分类号 TP332.2 **文献标识码** A

Design and Implementation of Separated Path Floating-point Fused Multiply-Add Unit

HE Jun HUANG Yong-qin ZHU Ying
(Shanghai Hi-Performance IC Design Centre, Shanghai 201204, China)

Abstract Considering the shortcoming that the fused multiply-add(FMA) unit increases the latency of separate floating-point addition and multiplication operations, a separated path FMA(SPFMA) unit was designed and implemented firstly. The SPFMA unit can reduce the multiplication and addition latency from 6 cycles to 4 cycles while keeping the FMA operation latency to 6 cycles by separating the multiplication and addition path, overcoming the shortcoming of traditional FMA unit. Then utilizing the specific technology cell library, the SPFMA was logically synthesized and could work at 1.2GHz above with area about 60779.44um^2. Finally based on the hardware emulation accelerating platform, the performance of the SPFMA unit was estimated through running the SPEC CPU2000 floating-point benchmarks. It turned out that the performances of the benchmarks are all improved, 5.25% at most and 1.61% on average, which proves that the SPFMA unit helps to promote floating-point performance further.

Keywords Floating-point add, Floating-point multiply, Fused multiply-add, Separated path, Floating-point performance, Operation latency

1 引言

浮点FMA(Fused Multiply-Add, FMA)运算自从1990年在IBM RS/6000[1]上首次实现以来,由于其对计算性能的显著提升,已经在很多商业处理器上实现,包括IBM POWER6[2]、POWER7[3]、Intel Itanium[4]、Fujitsu SPARC64 VIIIfx[5]、NVIDIA Fermi[6]等。通过AVX(Advance Vector Extension)指令集扩展,x86处理器也开始支持FMA,在新的IEEE 754-2008浮点标准[7]中也增加了对FMA运算的描述和规定。支持FMA运算已经成为主流选择。

浮点FMA运算将独立的浮点乘法和加法运算融合为一种运算,既能实现浮点乘法、加减法运算,还能实现各种融合乘加运算,如乘加、乘减、负乘加、负乘减等。跟离散乘法、加法相比,浮点FMA运算的优点比较明显:1)通过乘法和加法运算的部分重叠和并行,有利于减少运算延迟;2)乘法中间结果不舍入,有利于减少舍入误差,提高运算精度;3)通过共享部分硬件,减少硬件开销。

正是由于上述优点,FMA能提高科学计算应用中点积、矩阵乘法和多项式求值等核心计算的性能和精度,并在主流高端处理器中广泛实现。但是,其缺点也比较明显:由于采用统一的FMA算法结构来实现多种运算,独立的浮点加法和乘法运算延迟变得跟FMA运算延迟一样,这对浮点加法和乘法实际上是不利的。

由于FMA运算的显著优点,一直以来针对如何克服FMA运算缺点的研究比较少。最近有不少研究开始反思融合FMA结构的缺点[8,9]。文献[8]研究表明,将FMA分离为独立"乘法器+加法器"有利于提高浮点性能。该研究选择的主要是6个多媒体应用的测试课题,而没有采用更常用的测试课题,如SPEC测试课题[10]。如果采用独立的乘法器和加法器的设计,显然完全失去了FMA结构的优点。

到稿日期:2012-10-19 返修日期:2013-01-28

何 军(1980—),男,博士生,工程师,主要研究方向为微处理器设计,E-mail:joyhejun@126.com;黄永勤(1955—),女,高级工程师,博士生导师,主要研究方向为计算机系统结构、高性能计算;朱 英(1964—),女,硕士,高级工程师,主要研究方向为微处理器设计与验证。

为了在保持 FMA 运算的优点的同时，克服其缺点，通过对浮点加法和 FMA 算法进行研究，我们提出了一种分离通路乘加(Separated Path FMA，SPFMA)算法，设计并实现了一个分离通路乘法器，在保持 FMA 运算延迟(6 拍)不变的情况下，减少了独立乘法和加法运算的延迟(减为 4 拍)。利用 Synopsys Design Compiler(DC)和专用单元库进行逻辑综合评估，SPFMA 频率可以达到 1.2GHz 以上，面积60779.44um^2，相对于原基本的 FMA 设计面积增加了约 31.30%。利用硬件仿真加速器平台，我们对 SPFMA 进行了性能评估，结果表明，所有浮点课题性能均有所提高，最大提高 5.25%，平均提高 1.61%。

本文第 2 节概述 FMA 相关的研究工作，介绍基本 FMA 算法、对 FMA 算法的改进和反思；第 3 节详细介绍 SPFMA 算法的设计思想、结构实现及其逻辑综合实现评估；第 4 节介绍 SPFMA 性能评估的方法和结果；最后总结全文。

2 FMA 相关工作

2.1 基本 FMA 运算算法

假设 FMA 运算：$F=A\times B+C$，其中 A、B 为乘数和被乘数，C 为加数。令 $C=0$ 时，则 $F=A\times B+0$，相当于实现了乘法运算；令 $B=1.0$，则 $F=A\times 1.0+C$，相当于实现了加法运算。此外，还可以对 FMA 运算进行扩展，实现乘减($F=A\times B-C$)、负乘加($F=-A\times B+C$)、负乘减($F=-A\times B-C$)等运算。

基本 FMA 算法最早在 IBM RS/6000 上实现，其主要流程概述如下(主要是尾数部分处理，见图 1)：

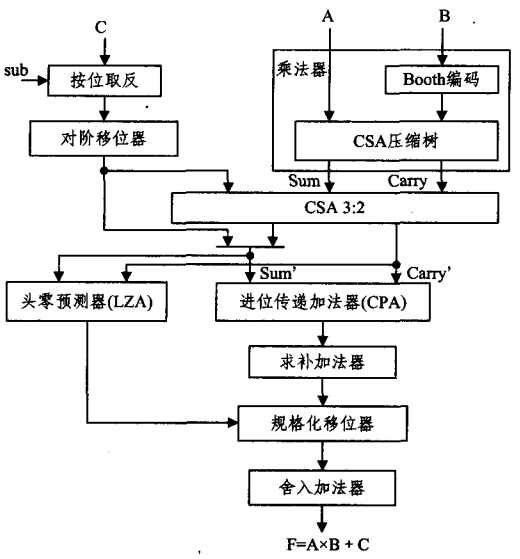

图 1 基本 FMA 算法结构

1)尾数相乘：被乘数与乘数尾数相乘得到保留进位形式乘积(Carry，Sum)；

2)加数对阶：在尾数相乘的同时，对加数进行对阶移位(如果是减法运算需要对加数按位取反)，使加数阶码和乘积阶码相等；

3)乘积与加数求和：将移位后的加数与乘积，通过 CSA(Carry Save Adder，CSA)3:2 压缩和拼接得到新的(Carry′，Sum′)形式和，然后利用进位传递加法器(Carry Propagate Adder，CPA)求和(如果结果为负数，还需要对负数结果求补码得到正数结果)；

4)结果规格化：在乘积与加数求和的同时，利用头零预测器(Leading Zero Anticipation，LZA)预测结果中首 1 的位置，然后根据 LZA 预测结果进行规格化移位，使结果的最高位为 1；

5)结果舍入：根据舍入模式，对规格化的结果进行舍入得到最终结果(如果结果溢出，还需要进行再规格化)。

2.2 FMA 算法改进和研究

自从基本 FMA 算法提出以来，有不少改进 FMA 算法的研究，归纳起来主要体现在如下几点：一是采用环回进位(End-Around-Carry，EAC)加法器代替普通的 CPA 去掉对负数结果的求补，二是对 LZA 算法的改进，三是结合双通路浮点加法器算法设计思想[11]，实现双/多通路 FMA 算法，减少运算延迟。

在基本 FMA 算法中，如果 CPA 对新的(Carry′，Sum′)求和(实为相减)得到的结果是负数，需要再进行求补，这意味着还需要一个 CPA 求补，而这里的 CPA 一般是 161 位宽的，延迟比较长。利用 EAC 加法器[12]可以同时计算出两个操作数绝对值，避免得到负数结果，从而节省了一个 CPA 的延迟，有利于减少 FMA 运算延迟。

LZA 是 FMA 的重要模块之一，文献[13]提出的 LZA 算法中，预编码具有统一的形式，不需要根据结果的正负采用不同的编码，且延迟比较短。文献[14]对 LZA 算法进行了进一步改进，提出了根据三操作数进行首 1 预测的算法，即直接根据 CSA3:2 的输入而不是输出预测结果首 1 的位置，使 LZA 路径上节省了一级 CSA3:2 的延迟。

减少 FMA 运算延迟是主要的改进目标之一，文献[15]将进位传递加法器和最后的舍入加法器合并以减少延迟，同时将规格化提到加法求和操作之前，该算法可以减少 15%～20%的延迟[15]。在文献[15]的基础上，结合双通路浮点加法器的思想[11](即根据实际运算类型是加法运算还是减法运算、两操作数阶码的差值(简称“阶差 d”)的情况，将算法分为 Close Path 和 Far Path 两条通路，前者负责实际为减法运算且$|d|\leqslant 1$的处理，其它情况由后者处理，在关键路径上可以节省一个移位器的延迟)，文献[16]根据实际运算类型和乘积与加数的阶差，将 FMA 算法的加法部分也分为两条通路，以减少加法运算的延迟。该研究表明，其乘加运算延迟相对于基本乘加算法能减少 10%左右，相对于文献[15]的算法有所增加，但是加法运算延迟能减少 30%～40%左右[16]。

文献[17]根据实际运算类型、乘积与加数的阶差情况，以及乘积和加数的相对大小程度，进一步细分了 5 种情况，提出了一种多通路 FMA 算法。文献[18]提出了一种三通路 FMA 算法，除了相同的 Close Path 外，根据乘积和加数的相对大小，增加了两条通路：Product Anchor Path 和 Adder Anchor Path，前者负责进行乘积大于加数的处理，后者负责加数大于乘积的处理。该结构采用 65nm SOI 工艺实现，相对于基本 FMA 结构，增加了 38.6%的面积，减少了约 12%的延迟。文献[18]还提出了一种桥接 FMA 结构，亦即将独立的浮点乘法和加法部件通过 Bridge 部件桥接起来，相对于基本 FMA 结构，桥接 FMA 结构延迟增加了 18.8%，面积增加了 51.7%，唯一的优点在于能并行执行浮点乘法和加法。

国内的西北工业大学和国防科技大学进行了 FMA 算法相关的研究。文献[19]中西北工业大学针对 PowerPC603e 微处理器系统，采用全定制的方式实现了一个 64 位双精度浮

点乘加器。文献[20]中，国防科技大学基于文献[15]的乘加结构，设计了一种5级流水的SIMD(Single Instruction Multiple Data)浮点乘加器，用以支持双精度和双单精度SIMD浮点乘法和乘加运算。

2.3 对FMA结构的反思

虽然FMA结构有明显的优点，但是最近有不少研究开始反思融合FMA结构的缺点，主要是由于采用统一的FMA结构，独立的浮点加法和乘法运算延迟会有所增加。文献[8]认为将FMA分离为独立"乘法器+加法器"有利于提高浮点性能，该研究测试课题主要选择的是6个多媒体应用的测试课题，而没有采用更常用的SPEC测试课题。文献[9]也针对FMA结构的缺点对浮点性能的影响进行了研究，提出了一种可变延迟(即加法、乘法比FMA延迟短)的FMA结构设想，通过运行SPEC CPU2000测试课题，发现其能获得4%～6%的性能改进。

3 分离通路FMA算法设计与实现

3.1 SPFMA算法设计思想

通过对浮点FMA算法结构的分析和研究，结合双通路浮点加法器的思想，我们提出了一种分离通路FMA算法(见图2)，将FMA算法中浮点乘法部分和浮点加法部分解耦合，实现两种运算通路的分离，从而将浮点乘法运算延迟和浮点加减法(含比较)运算延迟由6拍减为4拍，克服了传统FMA结构的缺点。在分离通路FMA算法中，相对于基本FMA算法，主要改进包括：

1)为乘法设置独立的求和、规格化和舍入模块，实现乘法结果的旁路输出；

2)为实现通路分离，将加数对阶移位器下移到乘法器MUL之后，不再与乘法器MUL并行；

3)加法部分采用双通路设计，分为NPath和FPath两条通路，使关键路径上只需要进行对阶移位或者规格化移位，减少一个移位器的延迟。

这里，NPath负责进行实际为减法运算且加数阶码E_C和乘积阶码E_{AB}阶码差值d(简称阶差，即$d=E_C-E_{AB}$)满足$-1\leqslant d\leqslant 2$的处理，这条通路上由于阶差d比较小，对阶移位可以简单实现，不需要复杂长位宽的对阶移位器，但由于操作数比较接近，容易出现高若干位相消的情况，因此需要进行头零预测，并进行长位宽的规格化移位器。FPath负责进行实际为加法运算，或者实际为减法运算，但是$d<-1$或$d>2$的处理，这条通路上由于阶差d比较大，需要复杂的长位宽对阶移位器，但是不会出现高若干位相消的情况，结果首1的位置容易确定，不需要LZA和长位宽的规格化移位器。

在分离通路的FMA算法中(见图2)，FMA运算为6级流水线FMA_ST0～ST5，乘法运算为4级流水线MUL_ST0～ST3，加减法(含比较)运算为4级流水线ADD_ST0～ST3。其中，独立乘法运算和加减法运算的通路是分离的，可以独立进行处理。但是，需要注意的是，由于这几种运算在结果写回寄存器文件时是共享同一个写端口的，因此需要发射控制部件在指令发射时进行控制，以避免这几种运算对写端口的冲突。

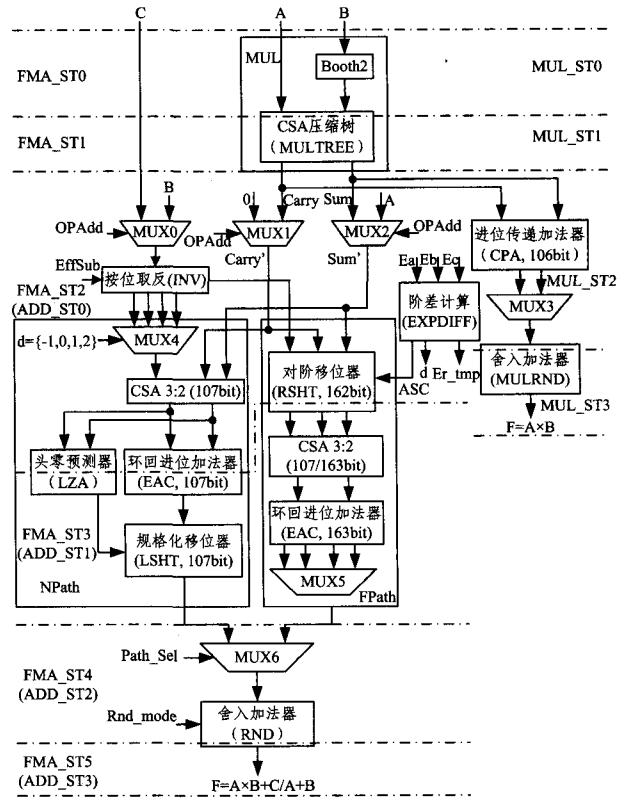

图2 分离通路FMA算法总体结构

3.2 SPFMA算法结构详述

3.2.1 乘法通路

乘法通路主要实现乘法运算功能，即计算尾数的乘积，主要包括乘法器(MUL)、106位CPA和二路选择器MUX3和舍入加法器(MULRND)(见图2)。乘法器MUL采用基数为4的Booth编码(Booth2)和Wallace树形部分积压缩阵列(MULTREE)，其中MULTREE主要由CSA4:2压缩器组成，将27个部分积经过4级Wallace树形压缩阵列压缩为2个部分积(Carry，Sum)，参见图3。MUL划分为两级流水线站台MUL_ST0和MUL_ST1。

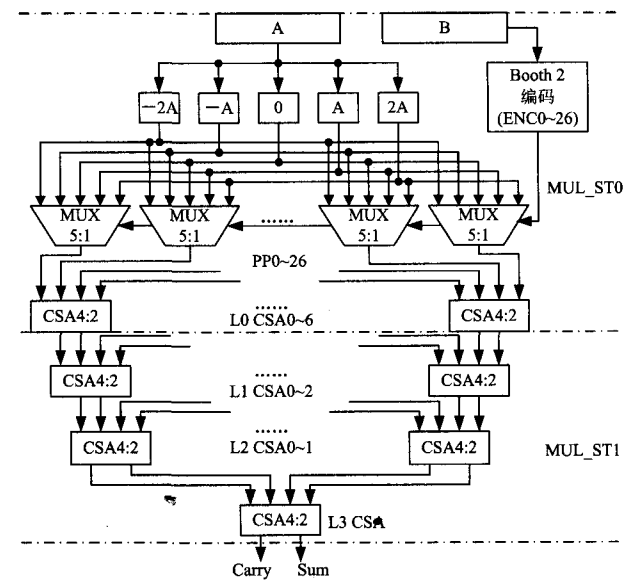

图3 乘法器(MUL)结构

106位CPA实现对Carry和Sum乘法运算结果求和，将其分解为高低53位两个部分，低53位是Kogge-Stone树形

加法器，高 53 位是两个 Kogge-Stone 树形加法器，进位输入分别为 0 和 1，然后根据低 53 位加法器的进位输出选择二者之一作为高 53 位的结果。这样 106 位加法器的延迟相当于一个 53 位加法器加上一个二路选择器的延迟。

二路选择器 MUX3 实现乘法运算结果的规格化，因为乘法运算结果只有两种情况：乘积 AB 尾数最高位为 0 或 1，为 0 时，需要将结果左移 1 位，乘积阶码不变；为 1 时，结果已经是规格化的，乘积阶码加 1。

舍入加法器(MULRND)实现乘法运算结果的舍入，主要根据舍入模式判断是否需要对乘积尾数加 1。这里还需要检测结果是否发生上下溢出，并给出相应的结果。

106 位 CPA、二路选择器 MUX3 和 MULRND 组成了乘法通路的两级流水线站台 MUL_ST2 和 MUL_ST3。

3.2.2 *加法通路*

加法通路主要实现加法运算功能，包括独立加法运算 $A+B$ 和 FMA 的加法运算 $AB+C$，主要包括数据选择(二路选择器 MUX0～2)、数据取反(INV)、阶差计算(EXPDIFF)、NPath、FPath、通路选择器(MUX6)和舍入加法器(RND)。

二路选择 MUX0～2 实现乘加运算的加法通路操作数(源操作数 C、部分积 Carry 和 Sum)和独立加法运算的操作数(源操作数 A 和 B)的选择，得到新的乘积(Carry′，Sum′)。当为加法、减法运算时，选择控制信号 OPAdd 有效，选择源操作数 A 和 B 作为加法通路的输入，否则 OPAdd 无效，选择源操作数 C、部分积 Carry 和 Sum 作为加法通路的输入。当实际为减法运算时，EffSub 有效，数据取反(INV)将加数按位取反。

阶差计算(EXPDIFF)实现加数阶码 E_C 和乘积阶码 E_{AB} 阶差 $d=E_C-E_{AB}$ 的计算(含加法两操作数阶码阶差的计算)，还实现 FPath 对阶移位量 ASC(即阶差 d 的绝对值)的计算和临时结果阶码 Er_tmp 的计算：对于 NPath，$Er\_tmp=E_{AB}+2$；对于 FPath 选择较大者作为临时结果阶码，即 $d<0$ 时，$Er\_tmp=E_{AB}$，否则 $Er\_tmp=E_C$。

通路选择器(MUX6)根据 EffSub 和阶差 d 形成通路选择信号 Path_Sel，选择 NPath 或者 FPath 通路的结果作为最后的结果，输出到舍入加法器(RND)进行舍入。当 EffSub 有效且 $-1\leqslant d\leqslant 2$ 时，选择 NPath，否则选择 FPath。舍入加法器(RND)主要根据舍入模式 Rnd_mode 对结果进行舍入，还需要检测结果是否发生上下溢出，并给出相应的结果。

下面主要介绍 NPath 和 FPath 的结构。

3.2.2.1 *加法通路 NPath*

NPath 主要负责实际为减法且 $-1\leqslant d\leqslant 2$ 的处理，主要包括四路选择器(MUX4)、CSA3：2 压缩器、头零预测器(LZA)、环回进位加法器(EAC)和规格化移位器(LSHT)，参见图 2。NPath 的处理跟基本 FMA 算法类似，只是对阶移位实现简化了。

由于阶差 d 只有{−1，0，1，2} 4 种情况，对阶移位可用一个四路选择器简单实现。经过对阶移位后，乘积 AB 和加数 C 的相对位置参见图 4，最多有 107 位宽。乘积(Carry′，Sum′)和对阶后的加数经过 CSA3：2 压缩器后得到两个操作数，并行输出到 LZA 和 EAC，分别进行首 1 预测和求和，然后 LSHT 根据 LZA 的输出对求和结果进行规格化左移，最多需要左移 107 位，因为首 1 的位置可能在 107 位的任意位置。同时需要根据左移位数相应调整临时结果阶码 Er_tmp 得到结果的阶码。

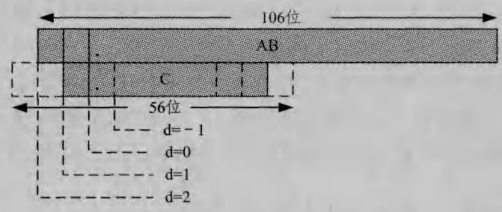

图 4 NPath 对阶移位示意图

这里再简单介绍一下 EAC 的原理，假设乘积为 $P=AB$，C 为加数，

1)若 $P-C\geqslant 0$，则 $P-C=P+\sim C+1$(“～”表示按位取反)；

2)若 $P-C<0$，则 $C-P=-(P-C)=-(P+\sim C+1)=-(P+\sim C)-1=\sim(P+\sim C)+1-1=\sim(P+\sim C+0)$

若 $P-C=P+2n-C=2n+(P-C)\geqslant 0$，则进位输出 $Cout=1$；若 $P-C<0$，则进位输出 $Cout=0$。因此，可以根据 $P-C$ 是否有进位输出来判断 $P-C$ 的符号，当 $Cout=1$ 时，$P-C\geqslant 0$，否则 $P-C<0$。据此可以计算 $P-C$ 的绝对值：并行计算 $S1=P+\sim C+1$ 和 $S0=P+\sim C+0$，若前者的进位输出 $Cout=1$，则 $|P-C|=S1$，否则 $|P-C|=\sim S0$(见图 5)。

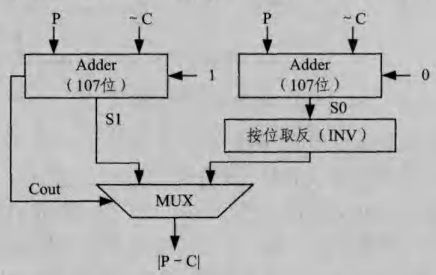

图 5 EAC 示意图

3.2.2.2 *加法通路 FPath*

FPath 主要负责实际为加法运算，或者实际为减法运算，但是 $d<-1$ 或 $d>2$ 的处理，主要包括对阶移位器(RSHT)、CSA3：2 压缩器、环回进位加法器(EAC)和多路选择器 MUX5，参见图 2。由于其他情况已经由 NPath 处理，FPath 处理的情况相当于 $d<0$ 和 $d\geqslant 0$ 两种情况，实际运算可能为加法或者减法。

对于 $d<0$，即 $C<AB$，对阶时将 C 右移向 AB 对齐，取 $Er\_tmp=E_{AB}+2$，C 最多右移 105 位，如果对阶移位量 $ASC>105$，则 AB 远大于 C，C 仅参与舍入粘贴位(sticky bit，st)的计算，最终结果近似于 AB，因此不需要再右移了(参见图 6(a))。最终结果的首 1 的位置由乘积 AB 首 1 的位置决定(图 6 中带“*”的位置即为可能的首 1 位置)，不需要进行预测。不论实际运算为加法或者减法，最终结果必为正值。

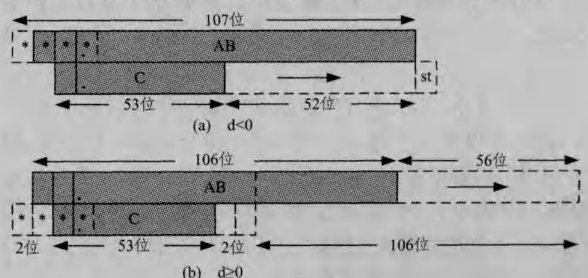

图 6 FPath 对阶移位示意图

对于 $d \geqslant 0$，即 $C \geqslant AB$，对阶时将 AB 右移向 C 对齐，取 $Er\_tmp = E_C$，AB 最多右移 56 位，如果对阶移位量 $ASC > 56$，则 C 远大于 AB，AB 仅参与舍入粘贴位 *st* (*sticky bit*) 的计算，最终结果近似于 C，因此不需要再右移了(见图 6(b))。最终结果的首 1 的位置由加数 C 首 1 的位置决定，不需要进行预测。当实际为加法运算时，最终结果必为正值，否则必为负值。

综合上述两种情况，考虑到乘积 AB 实际上是由(Carry′, Sum′)表示，对阶移位器 RSHT 实际主要由两个 162 位宽的右移器(移位量为 64 位)组成，一个用于 Carry′和加数 C 的右移，另一个用于 Sum′的右移，以节省硬件开销。由于加数 C 的移位量最多为 105，需要一级选择器先进行 64 位的移位，再由 RSHT64 完成 64 位以内的移位，参见图 7。

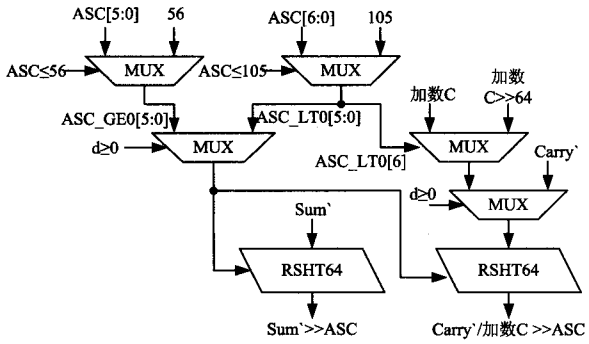

图 7　对阶移位器 RSHT

经过对阶移位后，分别利用一个 107 位的 CSA3:2 压缩器和一个 163 位的 CSA3:2 压缩器实现 $d<0$ 和 $d \geqslant 0$ 两种情况下部分积和加数的压缩，然后利用一个 163 位的环回进位加法器(EAC)实现结果求和。由于最终结果首 1 的位置只有 4 种情况，可利用四选一多路选择器 MUX5 实现结果规格化。

3.3　SPFMA 实现评估

我们用 Verilog 硬件描述语言完成 SPFMA 的 RTL (Register Transfer Level, RTL)逻辑设计，并对相关设计进行修改，完成了 RTL 逻辑验证，通过了我们内部测试验证集合的所有验证。

利用 Synopsys Design Compiler(DC)和专用单元库，我们对 SPFMA 进行了逻辑综合评估，SPFMA 频率可以达到 1.2GHz以上，其关键路径(CTP)主要在于 NPath 和 FPath 的 FMA_ST2、FMA_ST3 两拍，CTP 延迟为 0.698ns，面积为 60779.44um^2，其中组合逻辑 40896.34um^2，时序逻辑 19883.10um^2。

基于基本 FMA 算法，我们实现了一个乘加器(6 拍流水线)，并用 EAC 替换了其中的进位传递加法器 CPA 和求补加法器，以减少延迟。在相同的工艺条件和约束条件下，利用 DC 进行了逻辑综合，其关键路径延迟为 0.721ns，面积为 46290.18um^2，其中组合逻辑 28837.64um^2，时序逻辑 17452.54um^2。

相对基本 FMA，CTP 逻辑级数减少 3 级，延迟减少 3.19%，面积增加了约 31.30%，其中组合逻辑增幅较大，增加 41.81%，参见表 1。相对于基本 FMA，SPFMA 为乘法通路专门增加的 106 位 CPA、二路选择器 MUX3 和 MULRND (即乘法通路的两级流水线站台 MUL_ST2 和 MUL_ST3)的面积为 5828.96um^2，相当于增加了 12.56%的面积，如果去掉这一部分，相当于只增加了 18.74%的面积。

表 1　SPFMA 与基本 FMA 逻辑综合评估比较

	SPFMA	基本 FMA	SPFMA 增幅
CTP 逻辑级数	16	19	−3 级
CTP 延迟(ns)	0.698	0.721	−3.19%
面积(um^2)	60779.44	46290.18	+31.30%
组合逻辑(um^2)	40896.34	28837.64	+41.81%
时序逻辑(um^2)	19883.10	17452.54	+13.93%

总的来说，SPFMA 增加了 31.30%的面积，关键路径的延迟减少了 3.19%，独立乘法和加法运算延迟由 6 拍减为了 4 拍，减少了 1/3。

4　SPFMA 性能评估

4.1　评估方法

硬件仿真加速器验证是一种适于大规模和复杂 IC 设计的验证方法，兼有传统软件模拟验证和 FPGA 原型验证的优点，运行速度介于软件模拟验证和 FPGA 原型验证之间(大约是软件模拟验证的 1000 倍)，且可以比较方便地观测信号波形，便于错误定位和调试。目前不少 EDA 厂商都推出了硬件仿真加速器产品，比如 Cadence 公司的 Palladium 系列、Mentor Graphics 公司的 Veloce 系列等。

为了评估 SPFMA 的性能，我们利用 Cadence 公司的硬件仿真加速器运行 SPEC CPU2000 浮点测试课题的方法进行评估，该硬件仿真加速器验证系统可以支持运行轨迹和处理器软件模拟参考模型实时比较，适于运行海量伪随机测试激励，还可以运行操作系统和大量应用课题。我们对某国产处理器的 RTL 级设计代码进行了修改，用 SPFMA 替换了原有的 FMA 部件，并相应地修改了发射控制部件，并通过了正确性验证；然后对 RTL 级设计代码综合编译，映射到硬件仿真加速器上；最后利用硬件仿真加速器验证系统，运行 SPEC CPU2000 浮点测试课题，统计每道课题运行的时钟周期数。

4.2　评估结果

我们共运行了 14 道 SPEC CPU2000 浮点测试课题，统计了每道课题运行的核心时钟周期数 C，然后可以根据时钟周期数计算出课题实际运行的时间 $T(T=C/f, f$ 为核心时钟频率，这里为 1.2GHz)。根据 FMA 部件延迟优化前后的时钟周期数，可以计算出优化改进比例，见表 2。结果表明，所有测试课题浮点性能都有一定提升，其中课题 183.equake 改进比例最大，为 5.25%，其次是 171.swim，改进比例为 4.64%，另外 189.lucas、188.ammp、200.sixtrack 和 301.apsi 改进比例也比较明显，平均改进比例为 1.61%。

表 2　浮点课题测试结果

课题名称	FMA 部件延迟未优化		FMA 部件延迟优化后		改进比例
	C	T(s)	C	T(s)	
168.wupwise	7,692,377,985	6.41	7,649,180,986	6.37	0.56%
171.swim	876,761,579	0.73	836,043,654	0.70	4.64%
172.mgrid	18,143,934,032	15.12	18,037,191,595	15.03	0.59%
173.applu	323,208,079	0.27	320,306,561	0.27	0.90%
177.mesa	5,368,916,973	4.47	5,338,640,941	4.45	0.56%
178.galgel	4,011,677,405	3.34	3,993,486,731	3.33	0.45%
179.art	7,036,317,864	5.86	7,017,839,360	5.85	0.26%
183.equake	1,896,461,636	1.58	1,796,899,764	1.50	5.25%
187.facerec	7,305,659,307	6.09	7,250,245,493	6.04	0.76%
188.ammp	24,345,094,400	20.29	23,934,955,342	19.95	1.68%
189.lucas	6,360,700,306	5.30	6,153,189,118	5.13	3.26%
191.fma3d	25,836,270	0.02	25,596,416	0.02	0.93%
200.sixtrack	14,874,472,885	12.40	14,671,485,067	12.23	1.36%
301.apsi	9,297,664,627	7.75	9,174,491,924	7.65	1.32%

结束语 针对浮点 FMA 部件对独立乘法和加法运算延迟不利的缺点，我们设计并实现了一种分离通路的乘加器(SPFMA)，其在保持 FMA 运算延迟 6 拍不变的情况下，独立乘法、加法运算延迟由 6 拍减为 4 拍。经过 DC 逻辑综合，结果表明该乘加器相对于基本 FMA 乘加器面积增加了 31.30%，CTP 延迟减少了 3.19%。利用硬件仿真加速器验证系统运行 SPEC CPU2000 浮点测试课题的结果表明，所有浮点课题性能均有所提高，最大提高 5.54%，平均提高 1.66%。本文的主要贡献是，提出了一种新的乘加结构 SPFMA，它可以在保持 FMA 运算优点的同时克服其缺点，减小独立乘法和加法等运算的延迟，有利于浮点性能的进一步提升。此外，SPFMA 结构还可以支持独立乘法和加法运算的并行执行，只要将乘法和加法运算分配在独立的执行流水线上。相对于桥接 FMA 结构，SPFMA 结构硬件开销更小，而且不增加 FMA 运算延迟。下一步还可以对 SPFMA 的面积、延迟和流水线站台划分进行优化，实现更优化的设计。

参 考 文 献

[1] Montoye R K, Hokenek E, Runyon S L. Design of the IBM RISC System/6000 Floating-Point Execution Unit[J]. IBM Journal of Research and Development, 1990, 34: 61-62

[2] Eisen L, III J W W, Tast H-W, et al. IBM POWER6 Accelerators: VMX and DFU [J]. IBM Journal of Research and Development, 2007, 51: 663-684

[3] Boersma M, Kroener M, Layer C, et al. The POWER7 Binary Floating-Point Unit[C]// Proceedings of IEEE Symposium on Computer Arithmetic. Tübingen, Germany, IEEE Computer Society, 2011

[4] Sharangpani H, Arora K. Itanium Processor Microarchitecture [J]. IEEE Micro Magazine, 2000, 20(5): 24-43

[5] Maruyama T, Yoshida T, Kan R, et al. SPARC 6 4 VIIIfx: A New-generation Octocore Processor for Petascale Computing [J]. IEEE Micro, March-April 2010: 30-40

[6] Glaskowsky P N. NVIDIA's Fermi: The First Complete GPU Computing Architecture, Nvidia Fermi Whitepaper [EB/OL]. http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIAFermiComputeArchitectureWhitepaper.pdf, 2012-09-27

[7] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic[S]. IEEE Standard 754-2008. New York, USA, August 2008

[8] Lutz D. Fused Multiply-Add Microarchitecture Comprising Separate Early-Normalizing Multiply and Add Pipelines[C]// Proceedings of IEEE Symposium on Computer Arithmetic. Tübingen, Germany, IEEE Computer Society, 2011

[9] Galal S, Horowitz M. Latency Sensitive FMA Design[C]// Proceedings of IEEE Symposium on Computer Arithmetic. Tübingen, Germany, IEEE Computer Society, 2011

[10] SPEC. CPF2000(Floating Point Component of SPEC CPU2000) [EB/OL]. http://www.spec.org/cpu2000/CFP2000, 2012-09-27

[11] Quach N, Flynn M J. An Improved Algorithm for High-Speed Floating Point Addition[R]. CSL-TR-90-442. Computer Systems Laboratory, Stanford University, Aug. 1990

[12] Schwarz E M, Floating-Point B. Unit Design: the fused multiply-add dataflow, High-Performance Energy-Efficient Microprocessor Design [M]// Oklobdzija V G, Krishnamurthy R K, eds. Springer, Printed in the Netherlands, 2006: 199-201

[13] Schmookler M S, Nowka K J. Leading Zero Anticipation and Detection A Comparison of Methods[C]// Proceedings of IEEE Symposium on Computer Arithmetic. Vail, CO, USA, IEEE Computer Society, June 2001: 11-17

[14] 梅小露. 浮点乘加部件中三操作数前导 1 预测算法的设计[J]. 微电子学与计算机, 2005, 22(12): 16-20

[15] Lang T, Bruguera J. Floating-Point Fused Multiply-Add with Reduced Latency [J]. IEEE Transactions on Computer, 2004, 53(8): 088-1003

[16] Bruguera J D, Lang T. Floating-point fused multipy-add: reduced latency for floating-point addition[C]// Proc. 17th IEEE Symp. Computer Arithmetic. Hyannis, June 2005: 27-29

[17] Seidel P M. Multiple path IEEE floating-point fused multiply-add[C]// Proc. 46th Int. IEEE Midwest Symp. Circuits and Systems(MWS-CAS). 2003

[18] Quinnell E. Floating-Point Fused Multiply-Add Architectures [D]. University of Texas at Austin, 2007

[19] 靳战鹏，白永强，沈绪榜. 一种 64 位浮点乘加器的设计与实现[J]. 计算机工程与应用, 2006, 28(18): 95-98

[20] 吴铁彬，刘衡竹，杨惠，等. 一种快速 SIMD 浮点乘加器的设计与实现[J]. 计算机工程与科学, 2012, 34(1): 69-73

(上接第 27 页)

[2] 冯煌. GPU 图像处理的 FFT 和卷积算法及性能分析[J]. 计算机工程与应用, 2008, 44(2): 120-129

[3] Wang Xiang, Ding Yong. Efficient implementation of a cubic-convolution based image scaling engine[J]. Zhejiang Univ-SciC (Compute & Electron), 2011, 12: 743-753

[4] 肖汉. 利用 GPU 计算的双线性插值并行算法[J]. 小型微型计算机系统, 2010, 32(11): 2241-2245

[5] 郭一汉，史美萍，吴涛. 基于 GPU 的实时图像拼接[J]. 计算机科学, 2012, 39(7): 257-261

[6] 卢凤顺. CPU/GPU 协同并行计算研究综述[J]. 计算机科学, 2011, 38(3): 5-10

[7] Jason S, Edward K. CUDA by example: an introduction to general-purpose GPU programming[M]. Beijing: Tsinghua University Press, 2010: 64-74

[8] 刘建明. 基于全局优化的图像修复及其在 GPU 上实现[J]. 浙江大学学报：工学版, 2011, 45(2): 247-252

[9] 杜刘革. 基于多 GPU 的 FDTD 并行算法机器在电磁仿真中的应用[D]. 济南：山东大学, 2011

[10] 杨靖宇. 遥感影像 GPU 并行化处理技术与实现方法[D]. 郑州：解放军信息工程大学测绘学院, 2008