论文原创性和授权使用声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

本人授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

保密的学位论文在解密后也遵守此规定。

作者签名：张鑫

2008年5月30日

摘 要

浮点运算是高精度的运算方式，主要应用在科学和多媒体计算中。浮点运算能力是关系到 CPU 的多媒体、3D 图形处理的一个重要指标。相对于定点运算来说，不具备浮点运算单元的主 CPU 所从事的浮点运算，都是在许可范围内尽可能逼近的近似值。随着多媒体和互联网的高速发展，更高性能的精确计算对嵌入式 CPU 的浮点运算能力提出更高的要求。虽然一些软件库被开发出来暂时解决浮点计算问题，但是软件级别的模拟速度较慢，不能满足一些嵌入式系统的实时性要求，因此需要设计一种硬件结构来满足嵌入式领域的浮点运算需求。

本文给出一种兼容 IEEE754 标准的嵌入式高性能浮点协处理器——VFP 的设计与实现方法。该协处理器通过主处理器提供的外部协处理器接口同主处理器连接，支持浮点标量和向量操作，能够通过硬件高速执行符合 IEEE754 标准单精度、双精度的加、减、乘、除、乘加、平方根等运算，支持从浮点到整字的转换，具有分立的 64 位高带宽的 load/store 总线。

使用该协处理器的嵌入式协处理器可以得到多方面的性能提高：VFP 可以在浮点运算方面提高汽车的性能，包括在对精确性和可预测性要求较高的机械传动和车体控制应用，汽车中的机械传动、ABS 系统、牵引控制、灵活性背负系统等关键应用；图像应用如比例缩放、印刷中的字形产生、3D 转换、FFT、图形过滤等；下一代消费产品如网络应用产品、网关和机顶盒可以直接受益于 VFP，等等。

该协处理器使用自顶向下的基于系统级算法的快速成型设计流程。首先确定系统的设计目标和关键性能参数。然后在系统级设计阶段确定系统流水线划分和各运算实现算法，并进行有效性评估及优化。进一步在 RTL 级设计通过改变底层实现进行性能优化。最终得到符合要求的设计。

本设计着眼于嵌入式的应用领域，力求做到性能和功耗、面积代价的权衡。该设计的技术特点如下：

数据通路上，使用改进的浮点乘累加数据通路和浮点单/双精度乘法的舍入方法，提供完全真正意义上的符合 IEEE754 舍入标准的乘累加运算，缩短了流

水线关键路径，减少芯片面积代价。使用改进的恒定周期的迭代算法实现了除法和开方的复用设计，减少了面积代价，降低了系统功耗。使用分立的 Load/Store 总线和主处理器交换数据，去除了影响系统数据吞吐率的性能瓶颈。

流水线技术上，使用共发射的两条独立数据处理流水线。使用适合嵌入式设计的简化的记分牌技术来解决系统的数据和资源冲突，实现不同流水线指令的乱序执行。使用提交队列保证指令的顺序提交。使用缓冲队列减少寄存器堆的端口，使用分立迭代单元的方法实现浮点向量迭代运算操作。使用预测技术实现适合嵌入式的非精确浮点异常处理，给出完全符合 IEEE754 标准规定的异常处理结果。

本文的创新之处在于使用改进的浮点乘累加数据通路和浮点单/双精度乘法的舍入方法，提供完全真正意义上的符合 IEEE754 舍入标准的乘累加运算，缩短了流水线关键路径，减少芯片面积代价。使用改进的恒定周期的迭代算法实现了除法和开方的复用设计，减少了面积代价，降低了系统功耗。使用缓冲队列减少寄存器堆的端口。

设计采用 TSMC.13 工艺进行 RTL 综合，系统时钟达到 300Mhz，面积约为 10 万门，满足预定的设计要求。可以作为独立的 IP 用于 SOC 设计。该设计被国内知名公司采用，具有很高的实际商用价值。

关键词：向量协处理器；IEEE754；记分牌；浮点运算；浮点异常；乘累加；浮点除法；浮点开方；VLSI

Abstract

The high-precision Floating-point processing, mainly used in the filed of scientific and multimedia calculations, which is one of the most important indication of the central processing unit's capacity to processing Multimedia and 3D graphics. As to fix point processing, all the floating-point calculation in the CPU without the floating-point processing unit is an approximately simulation. As the rapid development of multimedia and international networks, more and more calculations call for a more precise and powerful capacity in embedded floating-point processing. Though, some software library was developed to carry on the floating-point calculation on fix point system, the terrible real-time response could not be tolerated by many embedded system. So, a hardware implement of floating-point calculations is badly in need in embedded application environments.

A design and implement of a high speed embedded floating-point coprocessor, VFP fully compliance with the IEEE754 floating-point arithmetic standard was presented in this thesis. This coprocessor attached with the main processor by a external coprocessor interface. It can performs single or double floating-point add, subtract, multiply, division, multiply and accumulation and square root operations. It supports conversion between floating-point number and integer. It has separated 64bits width load and store bus.

This floating-point coprocessor can provide benefit to the following several fields. It can improve accuracy and predictability of the automotive applications for engine management and power train computations. It can improve the performance of graphic processing such as image scaling, font generation in printing, 3D transform, FFT, image filtering etc. It can also benefit to the next generation consumer electronics such as networking applications, gateways and set-top boxes etc.

This coprocessor was designed with a top-down fast design flow. The design goal and critical parameter was set up in the first step. Then, the system partition and pipeline was designed, the algorithm of different calculation was worked out and their validity was evaluated and optimized. Further, the full design was optimized in RTL level to gain a higher performance. Finally, an expected implement was completed.

This design which was optimized with the trade-off between performance and power area cost for embedded fields has several technical improvements. The optimized floating-point multiply and accumulate data path and single/double precise floating-point round algorithm was applied in this design and pervaded a full compliance with IEEE754 round algorithm in multiply and accumulate operation. This improvement helps to shorten the critical path and reduce the area cost. The fix cycle iteration and logic resource sharing algorithm applied on division and square root operation can remarkably reduced the area and power cost of the whole system. The separated load/store bus used to exchanging data with main processor can remove

the bottle-neck of the system throughput.

Several pipeline technologies were applied in this design which contains two separated data processing pipeline. The simplified scoreboard algorithm was used in this embedded coprocessor to solve the system's data and resource hazard and carry out an out of order instructions operation in different pipeline. The retire queue was implemented to ensure the out of order instruction can retire in order. Buffer queue was used to reduce the write ports of register file. Separated iteration unit was implemented to realize the operation of vector instructions. Detections were made before executing and construct an IEEE754 compliant non-precise floating-point exception processing mechanism which is suitable for embedded in resource cost.

The novel improvements in this thesis include the optimized floating-point multiply and accumulate data path and single/double precise floating-point round algorithm which provided a full compliant with IEEE754 standard and help to shorten the critical path as well as reduced the area cost. It also include the fix cycle iteration and logic resource sharing algorithm applied on division and square root operation and the buffer queue used to reduce the write ports of register file.

The RTL decryption of the implement was synthesized into net-list with TSMC .13 technical libraries. The system clock turns out to be 300 Mhz and the area cost was approximately 100,000 gates which were achieved the expected design goal. It can be used in SOC design as a integrate IP and was applied by a Chinese well-known corporation, which proved to have a great potential in commercial applications.

Key Words: Vector coprocessor, IEEE754, Scoreboard, floating-point arithmetic, floating-point exception, multiply and Accumulate, floating-point division, floating-point square root, VLSI

第一章 绪论

本章介绍了浮点协处理器的背景，讨论了嵌入式浮点协处理器实际的应用价值和意义，以及其设计目标和设计方法学。指出文章的创新之处和结构。

1.1 背景

浮点运算和定点运算是处理器的常用运算方式。浮点运算是指随着计算过程和结果不同，指数改变，小数点会随之移动的运算。定点运算是指计算中及结果小数点位置固定不变，小数位长度不变的运算，其中包括整数运算。从两者的运算方式上不难得出浮点运算的精度将大大超过定点运算。通常情况下的运算不需要太高的精度，比如排序、数据库等操作，使用定点运算就可以满足要求。而浮点运算是一种高精度的运算方式，主要运用在科学和多媒体计算中。相对于定点运算来说，不具备浮点运算单元的主 CPU 所从事的浮点运算，都是在许可范围内尽可能逼近的近似值。随着多媒体和互联网的高速发展，更高性能的精确计算对嵌入式 CPU 的浮点运算能力提出更高的要求。浮点运算能力是关系到 CPU 的多媒体、3D 图形处理的一个重要指标。虽然一些软件库被开发出来暂时解决浮点计算问题，但是软件级别的模拟速度较慢，浮点运算软件模拟和硬件执行速度的差别在几个数量级。不能满足一些嵌入式系统的实时性要求。因此需要设计一种硬件结构来满足嵌入式领域的浮点运算需求。本文正是着眼于这一需求，给出一种嵌入式浮点协处理器的硬件实现。

1.2 设计目标和应用价值

本文给出一种兼容 IEEE754 标准[8]的嵌入式高性能浮点协处理器——VFP 的设计与实现方法。该协处理器通过 ARM 外部协处理器接口同 ARM11 系列处理器连接。能够通过硬件高速执行符合 IEEE754 标准单精度、双精度的加、减、乘、除、乘加、平方根等运算。支持从浮点到 ARM 整字的转换。通过特殊操作提供支持高级语言的 Round-towards-Zero 模式转换。具有分立的 64 位高带宽的

load/store 总线。除法、平方根运算和其他运算并行执行，可以减小除法、平方根运算的延迟。在特定运行模式下可以预测运行时间。支持浮点标量和向量操作。使用该协处理器的嵌入式协处理器可以得到如下等方面的性能提高：VFP 可以在浮点运算方面提高汽车的性能，包括在对精确性和可预测性要求较高的机械传动和车体控制应用、汽车中的机械传动、ABS 系统、牵引控制、灵活性背负系统等关键应用；图像应用如比例缩放，印刷中的字形产生、3D 转换、FFT、图形过滤等；下一代消费产品如网络应用产品、机顶盒和网关可以直接受益于该协处理器等等。

1.3 设计流程和技术特点

该协处理器使用自顶向下的基于系统级算法的快速成型设计流程[1]。首先确定系统的设计目标和关键性能参数。然后在系统级设计阶段确定系统流水线划分和各运算实现算法，并进行有效性评估及优化。进一步在 RTL 级设计通过改变底层实现进行性能优化。最终得到符合要求的设计。

本设计着眼于嵌入式的应用领域，力求做到性能和功耗、面积代价的权衡。该设计的技术特点如下：

数据通路上，使用改进的浮点乘累加数据通路和浮点单/双精度乘法的舍入方法，提供完全真正意义上的符合 IEEE754 舍入标准的乘累加运算，缩短了流水线关键路径，减少芯片面积代价。使用改进的恒定周期的迭代算法实现了除法和开方的复用设计，减少了面积代价，降低了系统功耗。使用分立的 Load/Store 总线和主处理器交换数据，去除了影响系统数据吞吐率的性能瓶颈。提供三种可以切换的运行模式，分别对应完全兼容 IEEE754 标准或者对运算的精确预测以及他们的折中情况。

流水线技术上，使用共发射的三条分立流水线。使用适合嵌入式设计的简化的记分牌技术来解决系统的数据和资源冲突，实现不同流水线指令的乱序执行。使用完成队列保证指令的顺序提交。使用数据前推减少部分发生数据冲突指令的写回延时。使用寄存器分组和分立迭代单元的方法实现浮点向量迭代运算操作。使用指令标记实现分支跳转指令造成的流水线冲洗，并结合条件码实现指令的有条件执行。使用预测技术实现适合嵌入式的非精确浮点异常处理，给出完全符合

IEEE754 标准规定的异常处理结果。使用独立的队列结构同主处理器交换控制信息，实现主从处理器流水线的松散同步。

1.4 创新之处

本设计的创新之处在于使用改进的浮点乘累加数据通路和浮点单/双精度乘法的舍入方法，提供完全真正意义上的符合 IEEE754 舍入标准的乘累加运算，缩短了流水线关键路径，减少芯片面积代价。使用改进的恒定周期的迭代算法实现了除法和开方的复用设计，减少了面积代价，降低了系统功耗。

1.5 文章结构

第 1 章是背景介绍和全文的概述。

第 2 章介绍了浮点协处理器设计中所涉及的相关理论基础。为设计提供理论依据。

第 3 章介绍了浮点协处理器的架构设计。确定了整个协处理器的主要性能结构参数，完成了自顶向下设计中的顶层设计。

第 4 章分为两节。第一节着重介绍决定浮点协处理器性能的部件——乘累加器的关键技术。第二节介绍了完整的乘累加流水线的结构设计。

第 5 章介绍了除法开方流水线的设计。首先介绍合并的除法开方迭代算法。进一步给出除法开方迭代单元的实现方法。

第 6 章给出协处理器的异常处理机制。

第 7 章分为三节。第一节介绍了设计结果的仿真和验证。第二节在利用第 2 章给出了的公式进行了系统的浮点舍入误差分析。第三节给出设计 RTL 描述的综合方法及结果。

第 8 章给出结论并指出有待进一步研究之处。

第二章 相关理论基础阐述

本章介绍了浮点协处理器设计中所涉及的相关理论基础。介绍了浮点协处理器所遵守的通用标准——IEEE754 浮点运算标准的基本内容，并分析了标准中的规定对应的具体实现。给出了浮点计算误差分析相关定理。

2.1 IEEE754 标准简介

任何一款通用浮点处理器都必须满足某种通用的浮点运算标准才能兼容不同的系统内容各异的运算。而 IEEE-754 浮点运算标准是当今计算机上最为通用的浮点运算标准。无论是 Intel 公司用于个人计算机的 CISCO 架构[13]的 CPU 还是 IBM 用于工作站的 Power 系列 RISC 架构 CPU[14]，主流浮点处理器均兼容该标准。

IEEE-754 浮点运算标准主要从以下几个方面进行了规定：

(1)基本的和扩展后的浮点数格式。

(2)浮点加、减、乘、除、开方、求余和比较等运算。

(3)浮点运算的舍入模式和舍入结果。

(4)整数和浮点数格式间转换和不同的浮点数格式间转换。

(5)浮点异常和它们的处理方式。

这几项内容是设计通用浮点处理器的重要依据。其中：

第 1 项规定了通用的浮点数据格式，这是各协处理器之间、协处理器和相关软件之间互相兼容的基础。

第 2 项规定了各种浮点运算的输入、运算过程、运算结果需要满足的条件。这是设计协处理功能和结构的依据。

第 3 项给出了浮点运算不同舍入模式的条件和异同以及舍入结果需要满足的格式得到该结果需要满足的条件，这是保证浮点运算精度的必要条件。

第 4 项规定了各种数据格式之间的转换。其中整数和浮点数之间的转换是整数主处理器和浮点协处理器、浮点协处理器和整数模拟的浮点库之间兼容的依

据。

第 5 项给出了浮点异常分类和相应的处理方法。浮点异常的分类是协处理器异常检测的依据，协处理器处理异常的方法必须满足该标准，这样才能在不同的情况下得到可重复的精确运算结果。

2.2.1 浮点数格式

IEEE 浮点数由三个部分组成：符号，指数，和尾数。尾数由小数部分和隐含位构成。

表 2.1 显示出了 32 位单精度和 64 位双精度浮点数的格式。

表 2.1 IEEE754 浮点数格式

	符号	指数	尾数	偏移量
单精度	[31]	[30:23]	[22:0]	127
双精度	[63]	[62:52]	[51:0]	1023

其中：符号位(Sign)：0 代表正数，1 代表负数。

指数(Exponent): 指数部分既要能够代表正指数，又要能够代表负指数。为了做到这一点，指数部分实际存储的是真实的指数加上一个偏移量。对于 IEEE 单精度浮点数，这个值是 127。因此，指数为 0 意味着指数部分的值将是 127。如果指数部分的值是 200，则意味着实际的指数值应该为 200 – 127 = 73。另外，指数值-127 和+128 是被保留作为特殊数来使用的。对于双精度浮点数，指数部分共占据 11 位，偏移量为 1023。

尾数(Mantissa): 尾数部分代表着浮点数的精确度，它包含着一个隐含位和小数部分。为了说明隐含位的值，考虑任何一个数用科学记数法有很多种表示方法。例如，5 可以被表示成下面几种形式：

5.00×10^{0}

0.05×10^{2}

5000×10^{-3}

为了最大化可表示数的数量，浮点数通常是以规格化的形式出现的。这要求将小数点放在第一个非 0 数的后面。在这种规则下，5 将会被表示成 5.0×10^{0}。

这样，一个小小的优化就能够被用在二进制数上，因为第一个非 0 的数字只能为 1。这样我们就可以假设领头的数字为 1，从而不需要将其显式表示出来。在这个条件下，尾数就能够以 23 个比特位来表示 24 个比特的信息。

浮点数的表示范围：正浮点数能被分成规格化的数和非规格化的数。具体数值范围如表 2.2 所示：

表 2.2 IEEE754 浮点数表示范围

	过小数	正常数	近似十进制整数
单精度	$\pm 2^{-149} \rightarrow (1-2^{-23})\cdot 2^{-126}$	$\pm 2^{-126} \rightarrow (2-2^{-23})\cdot 2^{127}$	$\pm \sim 10^{-44.85} \rightarrow \sim 10^{38.53}$
双精度	$\pm 2^{-1074} \rightarrow (1-2^{-52})\cdot 2^{-1022}$	$\pm 2^{-1022} \rightarrow (2-2^{-52})\cdot 2^{1023}$	$\pm \sim 10^{-323.3} \rightarrow \sim 10^{308.3}$

有五个数值范围是单精度浮点数所不能表示的。

小于 -(2-2^-23) x 2^127 的负数。

大于 -2^-149 的负数。

0

小于 2^-149 的正数。

大于 (2-2^-23) x 2^127 的正数。

2.2.2 特殊值

IEEE-754 保留了指数部分全为 0 和全为 1 的数来表示一些特殊值。

1. 0

0 是无法用浮点形式直接表达出来的，因为尾数部分隐含了一位 1。0 值是用指数部分全 0，尾数部分全 0 来表示的。注意+0 和 -0 是有区别的，虽然它们的值相同。

2. 低于正常数

当指数部分全为 0，而尾数部分不全为 0，则该数为一个非规格化数。它不含有隐含位 1。这样非规格化的数的表达式为(-1)s x 0.f x 2^-126。这里的 s 是符号位，f 是尾数部分。你可以将 0 的特殊表达方式作为非规格化数的一个特例。

3. 无穷大数

当指数部分全为 1 而尾数部分全为 0 时，该浮点数表示无穷大。由于符号位任意，因此有正无穷大和负无穷大。

4. Not A Number

NAN 用来代表一个非真实的数字。当浮点数的指数部分全为 1，而尾数部分不全为 0 时，该浮点数代表了一个 NAN。

非数又分为告警非数（SNAN）和静态非数（QNAN）。告警非数尾数首位为 0，静态非数首位为 1。浮点处理对两者区别对待。

表 2.3 IEEE 定义的特殊数值

Sign(s)	Exponent(e)	Fraction	Value
0	00000000	00000000000000000000000	+0 (positive zero)
1	00000000	00000000000000000000000	-0 (negative zero)
1	00000000	10000000000000000000000	-2^{-126} x0.(2^{-1})= -2^{-126} x 0.5
0	00000000	00000000000000000000001	$+2^{-126}$ x0.(2^{-23}) (smallest value)
0	00000001	01000000000000000000000	$+2^{1-127}$x1.(2^{-2})= $+2^{1-127}$x1.25
0	10000001	00000000000000000000000	$+2^{129-127}$x1.0= 4
0	11111111	00000000000000000000000	+ infinity
1	11111111	00000000000000000000000	- infinity
0	11111111	10000000000000000000000	Not a Number(NaN)
1	11111111	10000100010000000001100	Not a Number(NaN)

2.2.3 异常

IEEE-754 标准定义了五种异常，分别是：无效操作异常、被零除异常、非精确异常、下溢异常、上溢异常。

1. 无效操作异常

计算中有一些操作是无效的，例如负数开方。无效操作的结果应该是一个静态非数。

下面给出了一些无效操作的例子：

任何对 NAN 的操作

加或减：$\infty+(-\infty)$

乘：$\pm 0 \times \pm\infty$

除：$\pm 0/\pm 0$ 或 $\pm\infty/\pm\infty$

开方：开方数小于 0

2. 被零除异常

除了 0 以外的任何数被 0 除都等于无穷大，将给出被零除异常。

3. 非精确异常

当计算的结果由于受到指数或精度范围限制时，将给出非精确异常。

4. 下溢异常

当极微小的数被发现以及损失了精度时，都会发生下溢异常。当计算结果被位于 $\pm 2^{E_{min}}$ 内，在舍入前后极微小的数会被发现；当计算结果非精确或非规格化时会损失精度。当 VFP 的计算结果在舍入后发现了极微小的数，与此同时计算结果也是非精确的，这时下溢异常将会发生。

5. 上溢异常

上溢异常将会在计算结果超过了浮点数所能表达的最大指数范围时发生。如果仅有一个操作数是无穷大，上溢异常是不会发生的，因为无穷大是精确的。被零除也不会触发这个异常。

表 2.4 IEEE754 定义默认结果

异常类型	默认正结果	默认负结果
无效操作	QNaN	QNaN
除零	正无穷	负无穷
上溢	舍入到最近，舍入到正无穷： 正无穷 舍入到零，舍入到负无穷： 正的最大值	舍入到最近，舍入到负无穷： 负无穷 舍入到零，舍入到正无穷： 负的最大值
下溢	正常舍入结果	正常舍入结果
不精确	正常舍入结果	正常舍入结果

2.2.4 舍入模式

IEEE-754 定义了四种舍入模式：舍入到最近的数、舍入到零、向上舍入、

向下舍入。舍入模式影响除比较之外的所有数据处理操作。

舍入到最近的数：这是标准的默认舍入方式。即向上或向下舍入到最近的精确的整数。

舍入到零：一般在这种模式下，结果都不会舍入，多余的 bits 会被截掉。例如：3.47 将会被截成 3.4。

向上舍入：结果将会朝+∞方向舍入，例如：3.2 将舍入到 4，-3.2 舍入到-3。

向下舍入：结果朝－∞方向舍入。例如：3.2 将舍入到 3，-3.2 舍入到-4。

上溢默认结果：

Round to nearest 模式：与中间结果同号的±∞

Round to zero 模式：与中间结果同号的目的精度最大数

Round towards +∞模式：正溢出时为 +∞，负溢出时为负的目的精度最大数

Round towards -∞模式：正溢出时为正的目的精度最大数，负溢出时为 -∞

下溢默认结果：

在舍入和 Flush 前写入+0 代替。

在介绍各种舍入模式的具体实现之前先介绍保护位的概念。保护位是在保证初始操作数和运算结果尾数限制在规定尾数基础上，在运算中间步骤上保留一些附加位，通常称为保护位。目的是使结果具有更高的精度。生成最终结果时去除保护位，完成对最终结果的舍入。

1. Round to nearest 模式

这种最复杂的模式是 IEEE754 默认舍入方式，得到与中间结果同号的目的精度最大数。并且是无偏近似。

过程如下：如果去除位的 MSB（最高有效位）位置上为 1，则在保护位的 LSB（最低有效位）位置上加 1。去除位的 MSB（最高有效位）位置上为 0，则去除保护位。这样三位有效数字的 $0.b_{-1}b_{-2}b_{-3}1\cdots$ 就被舍入成 $0.b_{-1}b_{-2}b_{-3}+0.001$，而 $0.b_{-1}b_{-2}b_{-3}0\cdots$ 则被舍入成 $0.b_{-1}b_{-2}b_{-3}$。当去除位为 $10\cdots0$ 的情况时，待截取的数字位于两个最近的截取表示正中间，是一种中间状态。对此采取舍入到最近的偶数的方法。这样三位有效数字的 $0.b_{-1}b_{-2}0100\cdots00$ 就被舍入成 $0.b_{-1}b_{-2}0$，而 $0.b_{-1}b_{-2}1100\cdots00$ 则

被舍入成$0.b_{-1}b_{-2}1+0.001$。误差的区间大致在保护位 LSB 位置的-1/2 到+1/2。如图 2.1 所示。

中间状态的判断方法：在执行操作的中间步骤设置 2 个保护位。其中第 1 位是将被去除的尾数部分的最高有效位。第 2 位是尾数的完整表示中除去最高有效位的其余各位的逻辑或。对于串行操作此位在操作中初始化为 0，如果有一个 1 从该位置移出，则将该位置 1，并一直保持。通常称为黏着位。保护位为 10 则处于中间状态。

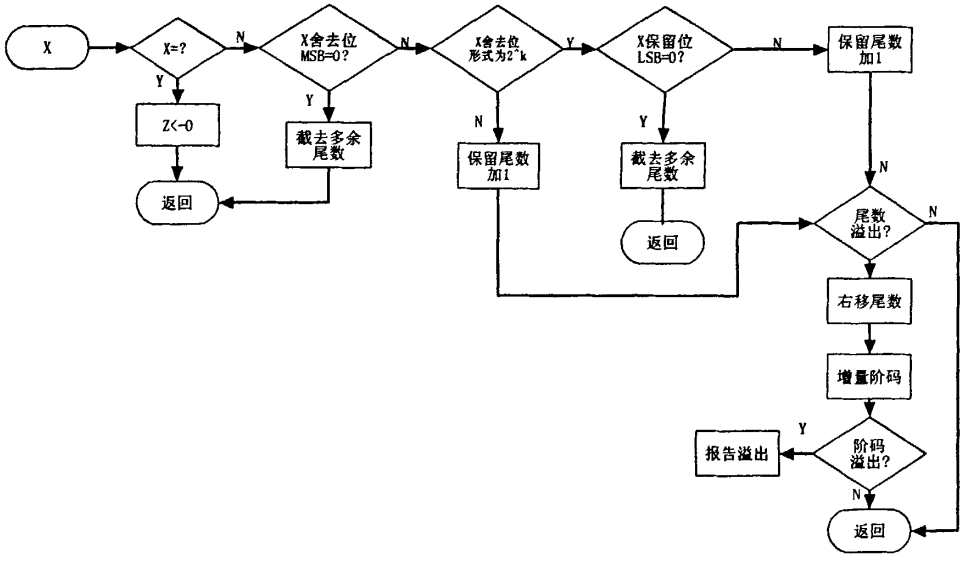

图 2.1 Round to nearest 模式流程图

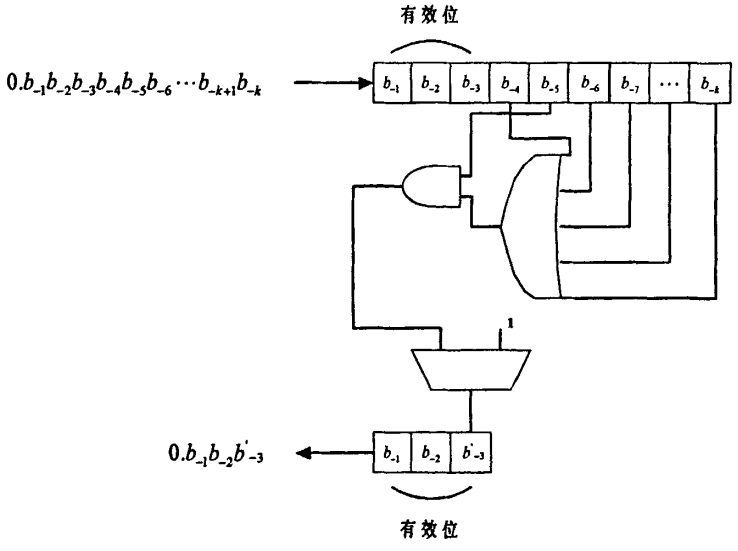

图 2.2　Round to nearest 模式逻辑结构示意图

2. Round to zero 模式

在这种模式下，直接将保护位去除，剩余各位不加改变。是有偏近似。三位有效数字的六位数字 $0.b_{-1}b_{-2}b_{-3}000$ 到 $0.b_{-1}b_{-2}b_{-3}111$ 内的所有小数都将截取为 $0.b_{-1}b_{-2}b_{-3}$，三位结果误差为从 0 到 0.000111。截断误差为从 0 到接近有效数字的最低位的 1。

3. Round towards +∞/-∞模式

结果将会朝+∞/-∞方向舍入，在实现间隔算法（Interval Arithmetic）时有用，间隔算法是指在一系列浮点运算结束时，由于硬件的限制必须导致舍入，使得不能得到精确答案。因此采用对系列中每一次运算做两次，一次使用 Round towards +∞模式，一次使用 Round towards -∞模式，则结果一定保持在正确答案的上下界之间。可以用于分析结果。

2.2.5 有关浮点误差的规定

(1)IEEE 标准要求在 10^{-44} 和 10^{+44} 数量级范围内的典型数字进行正确地舍入，但允许对更大的指数进行略有误差的舍入。

(2)一种测量误差大小的单位称为最后一位中的单位（Unit in the Last Place or Unit of Least Precision，简写为 ulp）[11]，表示两个最接近的浮点数的差值[4]。在标准表示中，浮点数尾数的最低有效位就是其最后一位。该位表示的值（例如，其表示除此位外均相同的两个数的绝对差值）就是该数的最后一位的单位。如果计算结果是通过将真实结果舍入到最接近的可表示数得出的，则无疑舍入误差不会大于计算结果最后一位中的单位的一半。换句话说，在舍入模式为舍入到最接近值的 IEEE 算法中，计算结果的舍入值为：

$$0 \le |\text{舍入值}| \le \frac{1}{2} ulp$$

浮点数的 ulp 取决于表示该数的精度。

表 2.5 列出四种不同精度的 ulp:

精度	值
单精度	ulp(1) = 2^-23 ~ 1.192093e-07
双精度	ulp(1) = 2^-52 ~ 2.220446e-16
双精度扩展 (x86)	ulp(1) = 2^-63 ~ 1.084202e-19
四倍精度 (SPARC)	ulp(1) = 2^-112 ~ 1.925930e-34

(1)IEEE 标准要求精确舍入加、减、乘、除的结果。也就是说，必须先精确计算结果，然后舍入为最接近的浮点数（或者舍入为偶数）。当两个浮点数的指数有很大差异时，精确计算这两个浮点数的差或和的开销会非常大。我们引入保护位，它提供了一种在保证相对误差很小的同时计算差值的实用方法。

(2)许多问题（如数值积分和微分方程的数值解）涉及计算多个项的和。因为每个加法运算都有可能引入大至 0.5 ulp 的误差，所以产生数千项的求和会具有相当大的舍入误差。纠正这一点的简单方法是将部分被加数存储在双精度变量中，并使用双精度执行每个加法运算。

(3)IEEE 标准并不保证同一程序在所有符合该标准的系统上都将提供完全相同的结果。实际上，由于种种原因，大多数程序都会在不同的系统上产生不同的结果。其中一个原因是，大多数程序都涉及十进制格式和二进制格式之间的数字转换，而 IEEE 标准没有完全指定执行这样的转换必须使用的准确度。IEEE 标准要求将每个结果都正确舍入到它的目标的精度，但是标准不要求由用户程序确定该目标精度。因此，不同的系统可能将其结果提供给不同精度的目标，使同一程序产生不同的结果（有时差异很大），即使那些系统都符合标准亦是如此。

(4)按照 IEEE754 标准规定，每一步运算的结果都必须舍入到目标精度，然后才能参与到下一步运算。这是为了避免累积误差，便于科学运算的误差分析[5]。然而一些单精度/ 双精度系统提供单个指令将两个数相乘并与第三个数相加，只进行一次最终舍入。此运算称为合并的乘-加，会导致同一程序在不同的单精度/双精度系统上产生不同的结果；与扩展精度一样，它甚至会导致同一程序在同一系统上产生不同的结果，这取决于是否使用它和何时使用它。在本处理器设计中中没有这种情况，我们将乘累加的结果分两步运算，首先得到精确舍入的乘法的结果，再进行累加操作，避免了非精确的误差累积。

2.2 误差分析的数学基础

计算机中的浮点数 F 可以表示为：

$$f = \pm w \times \beta^J \qquad L \le J \le U$$

这里 β 是机器的基数，J 表示阶码，w 是尾数，尾数表示为：

$$w = 0.d_1 d_2 \ldots d_t$$

其中 t 是字长，$0 \le d_i < \beta$，若 $d_1 \neq 0$，则称该浮点数为规格化的浮点数。我们用 F 表示一个系统的浮点数的全体所构成的集合，则有：

$$F = \{0\} \cup \{f : f = \pm 0.d_1 \ldots d_t \times \beta^J, 0 \le d_i < \beta, d_1 \neq 0, L \le J \le U\}$$

上述的集合用四维数组来刻画 (β, t, L, U)。集合 F 是一个包含 $2(\beta-1)\beta^{t-1}(U-L+1)+1$ 这么多个元素的有限数集，这些数对称地分布在区间[m,M]和[-M,-m]中，其中：

$$m = \beta^{L-1}, M = \beta^U(1-\beta^{-t}) \qquad (1)$$

1. 实数转换成浮点数时的相对误差

记实数 x 的浮点数表示为 fl(x)：若 $x = 0$，则 fl(x)取为零；若 $m \le |x| \le M$，则当使用舍入法时，取 fl(x)为 F 中最接近于 x 的 f；则当使用截断法时，取 fl(x)为 F 中满足 $|f| \le |x|$ 且最接近于 x 的 f。例如，对 (β, t, L, U)=(10，3，0，2)和实数 x=5.45627，若用舍入法，则得 fl(x)=0.546*10；若用截断法，则有 fl(x)=0.545*10。有了实数的浮点数表示以后，我们就可确定它的相对误差，这是进行误差分析的基础。

下面的基本定理给出了一个实数表示成浮点数所引起的相对误差。

定理 1 设 x=0 或 $m \le |x| \le M$，其中 m 和 M 由（1）式定义，则

$$fl(x) = x(1+\delta), |\delta| \le u \qquad (2)$$

其中 u 为机器精度，即

$$u = \begin{cases} \frac{1}{2}\beta^{1-t}, \text{用舍入法} \\ \beta^{1-t}, \text{用截法} \end{cases}$$

证明 若 x=0，则 fl(x)=0，则(2)自然成立。

现在假定 x>0（若 x<0，证明完全类似），即 $m \le x \le M$，则 x 可以表示为：

$$x = 0.d_1 \ldots d_t d_{t+1} \ldots \times \beta^{\alpha}, d_1 \ne 0, L \le \alpha \le U$$

则：

$$\beta^{\alpha-1} \le x \le \beta^{\alpha} \qquad (3)$$

对于舍入法，有

$$fl(x) = \begin{cases} (0.d_1 \ldots d_t + \beta^{-t}) \times \beta^{\alpha}, d_{t+1} \ge \frac{1}{2}\beta \\ 0.d_1 \ldots d_t \times \beta^{\alpha}, d_{t+1} < \frac{1}{2}\beta \end{cases}$$

即：

$$|fl(x) - x| \le \frac{1}{2}\beta^{\alpha-1} = \frac{1}{2}\beta^{\alpha-1}\beta^{1-t} \le \frac{1}{2}x\beta^{1-t}$$

$$\frac{|fl(x) - x|}{x} \le \frac{1}{2}\beta^{1-t} \qquad (4)$$

对于截断法，有

$$fl(x) = 0.d_1 \ldots d_t \times \beta^{\alpha}$$

所以

$$|fl(x) - x| \le x\beta^{1-t}$$

$$\frac{|fl(x) - x|}{x} \le \beta^{1-t} \qquad (5)$$

由(4)式和(5)式从而得知定理 1 成立，定理 1 规定了在浮点数和实数转换时的舍入误差。下面研究在浮点数运算时的舍入误差定理和分析具体的情况。另外为了研究方便，有时也会把定理 1 中的相对误差公式写成下式：

$$fl(x) = \frac{x}{(1+\delta)}, |\delta| \le u \qquad (6)$$

2. 浮点数基本运算相对误差

考虑基本运算的舍入误差。设 $a,b\in F$ 是两个给定的浮点数，我们用 $\circ$ 表示 $+-\times/$ 中任意一种运算。$fl(a\circ b)$ 的意义是先进行运算，得到精确的实数，再按舍入规则表示成浮点数。在运算中，若出现 $|a\circ b|<m$ 或 $|a\circ b|>m$，则就是发生了下溢或上溢。在不发生溢出的情况下，由定理 1 得到：

定理 2 $fl(a\circ b)=(a\circ b)(1+\delta),|\delta|\le u$

定理 3 若 $|\delta_i|\le u$ 且 $nu\le 0.01$,那么

$$1-nu\le\prod_{i=1}^{n}(1+\delta_i)\le 1+1.01nu \tag{7}$$

或写成

$$\prod_{i=1}^{n}(1+\delta_i)\le 1+\delta,|\delta|\le 1.01nu \tag{8}$$

证明 因为 $|\delta_i|\le u$，故有

$$(1-u)^n\le\prod_{i=1}^{n}(1+\delta_i)\le(1+u)^n \tag{9}$$

为证明定理结论，我们只需证明以下两个不等式

$$(1-u)^n\ge 1-nu$$
$$(1+u)^n\le 1+1.01nu$$

我们考虑考虑函数 $(1-x)^n(0<x<1)$ 的 Taylor 展开：

$$(1-x)^n=1-nx+\frac{n(n-1)}{2!}(1-\xi)^{n-2}x^2,\xi\in(0,x)$$

所以 $1-nx\le(1-x)^n$

$$1-1.01nu\le 1-nu\le(1-u)^n \tag{10}$$

又由 e^x 的幂级数

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^2}{3!} + \ldots$$
$$= 1 + x + \frac{x}{2} x \left(1 + \frac{x}{3} + \frac{2x^2}{4!} + \ldots \right)$$

级数 $1 + \frac{x}{3} + \frac{2x^2}{4!} + \ldots$ 收敛于：

$$1 + \frac{x}{3} + \frac{2x^2}{4!} + \ldots < e^x$$

所以 $1 + x \le e^x \le 1 + x + \frac{x}{2} x e^x$，又因为 $e^{0.01} < 2$，所以当 $0 \le x \le 0.01$ 时：

$$1 + x \le e^x \le 1 + 1.01x$$

由假设： $nu \le 0.01$

从而得到： $(1+u)^n \le e^{nu} \le 1 + 1.01nu$ (11)

综合(10) 和(11)便可得到定理的结论。

由上述的定理还能够退推导出在矩阵运算时的误差和浮点数上下界等的问题，这里不再进行详述，由上面的三个定理就能对浮点数的舍入运算进行误差分析了。

第三章 架构设计

本章介绍了浮点协处理器的架构设计。包括设计目标的制定，主要数据通路的结构，向量运算的实现方法，以及冲突处理和顺序提交的实现方法。

3.1 设计目标和基本架构

在 Top-Down 的设计流程中首先确定设计目标，这是整个设计实现的依据，架构的选择和部件的优化都是朝着设计目标进行努力。

为了制定出合理的设计目标，首先分析浮点运算的特点。为达到协处理的硬件加速目的，主要运算应该在一个周期内完成。根据统计，通常的浮点应用程序中除去数据传输指令，70%以上的的运算指令都是乘累加指令，只含有少量除法开方指令。嵌入式运算要求精度不是太高，为权衡面积因素，可以将单精度的乘累加指令设计为一个时钟周期完成，双精度的乘累加运算可以使用两个时钟周期迭代完成，这可以通过改造乘法器的结构实现，具体分析参见后续章节。除法开方运算在嵌入式等用途的浮点应用程序中出现频率较低，计算较为复杂，因此使用迭代算法串行实现，这样既不会对整体性能产生较大的影响，又可以节省大量资源。以上两个关键架构确定后，再加入其他指令和功能。在系统控制处理上，采用 32 个 32 位寄存器组成寄存器堆。乘累加流水线和除法开方流水线采用共用译码级和发射级结构，发射级设迭代单元。指令在发射级顺序发射，指令发射后乱续执行。向量指令的尾指令留在迭代单元中继续迭代直到结束。采用特殊结构处理异常、冲突和乱续执行等多流水线相关问题。使用提交队列记录指令的发射顺序，完成指令的顺序提交。系统结构图如图 3.1 所示。

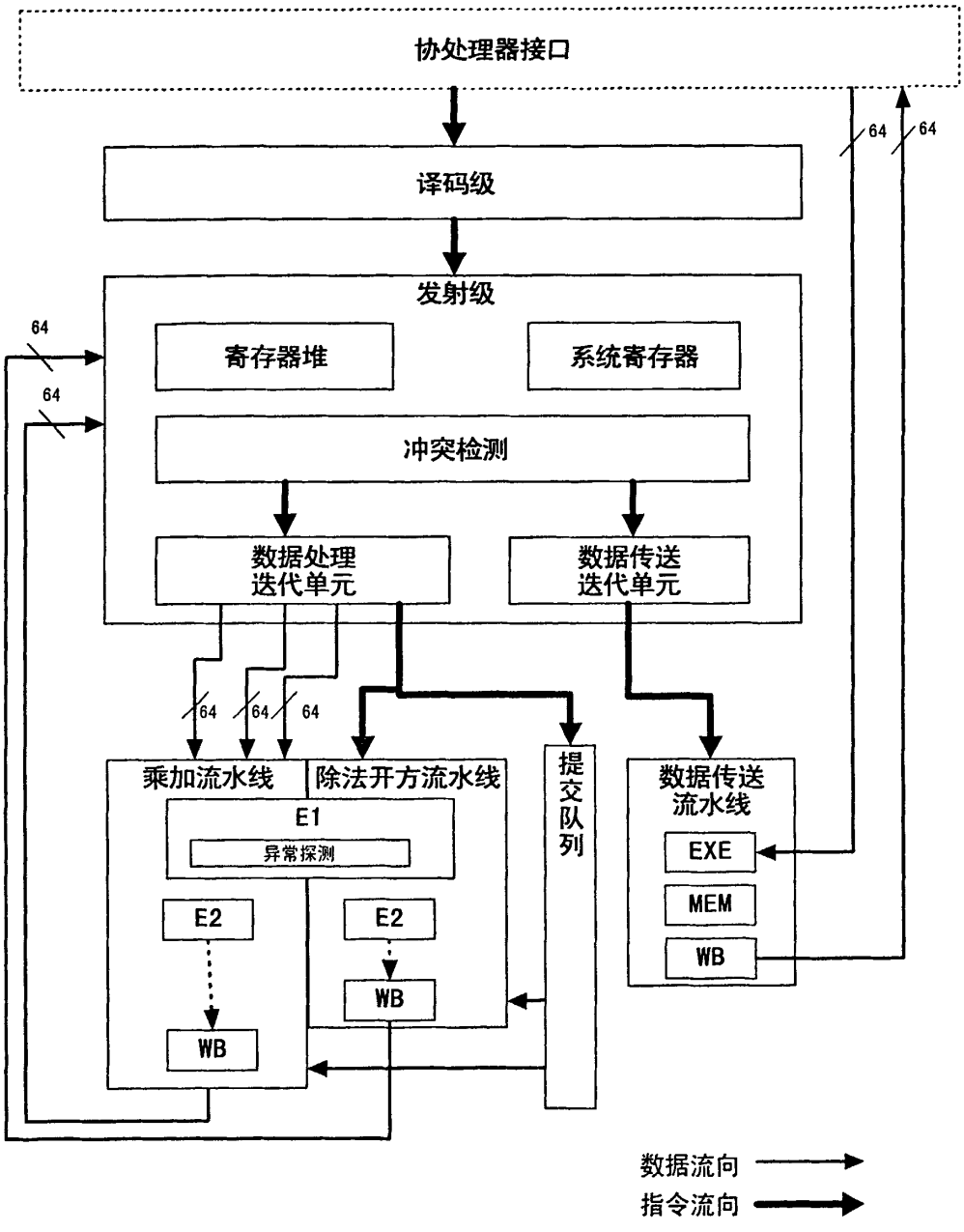

图 3.1 系统结构图

3.2 主要数据通路

通过初步的架构分析，可以确定处理器支持的运算类型，以及主要的数据通路。列举如下：

乘累加运算： ，其中 Fm 为 0 时表示乘累加运算的特例——加法运算，Fd

为 0 时表示特例——乘法运算。通过乘累加流水线实现。

比较运算：大于、等于、小于，无法比较，通过改写系统标志位表示比较结果。

单双精度转换：双精度转成单精度通过直接舍入实现，单精度转成双精度通过指数调整和尾数补齐实现。

浮点数转整数：通过指数运算和尾数舍入和移位实现。

整数转浮点数：通过指数运算和尾数移位实现。

以及取反、取负和取绝对值运算。

以上几种运算通过多路器合并到乘累加流水线的数据通路上，以实现资源共享。

除法运算：Fn/Fm。

开方运算：Fm^(1/2)。

除法和开方运算采用合并的数据通路，以实现资源共享。具体的分析详见后续章节。

3.3 控制逻辑

1. 向量运算

设计向量运算采用迭代串行完成。是涉及到多个目标寄存器的操作。并且在计算的不同阶段涉及到不同的的源寄存器。向量运算是一个完整的循环，每一次迭代计算产生结果的一个元素，然后修改下标，继续计算下一个元素。

向量操作的特点：

(1)每一个元素结果的运算与前面无关，这样允许很深的流水线操作而不产生于流水线相关（Dependence）的问题。

(2)循环指令由向量运算代替，减少了循环转移引起的控制冲突（Hazard）。

(3)单个向量指令指定了大量的工作。

(4)向量存取指令存取相邻的向量元的操作在高度交叉的存储器中能顺利进行。

向量运算译码级将向量指令译码，产生头指令和头指令所需操作数地址。发射时，头指令进入发射级迭代单元并修改操作数寄存器地址，产生尾指令。尾指

令在发射级迭代单元依次进行迭代直到结束。

2. 寄存器堆设计

在浮点处理器中设计三条独立工作的流水线：浮点乘加流水线（MAC），进行除了除法和开方运算的所有算数和逻辑运算；除法开方流水线（DS），用于浮点的除法和开方运算；Load/Store 流水线（LS），完成寄存器堆与浮点处理器外部的数据交换。这三条流水线对寄存器堆的写操作具有随机的特点，即三条流水线可能同时写寄存器堆。在没有冲突的情况下要保证三条流水线互不影响的同时写寄存器堆就需要考虑三条流水线将三个 64 位双精度浮点数写入寄存器堆的情况，这时就需要 6 个 32 位寄存器堆写端口，这将占用很大的面积并使功耗大大增加。

考虑到三条流水线同时写寄存器堆的机会很小，尤其是除法和开方流水线需要十几个或几十个时钟周期的迭代才能完成一次完整的运算，所以除法和开方流水线两次写寄存器堆之间要间隔多个时钟周期，也就是说写寄存器堆的概率比较小。所以我们将三条流水线分配优先级，当有两条或三条流水线同时写寄存器堆时优先级高的流水线先写寄存器堆，同时为了不影响优先级低的流水线中后续指令继续执行，将优先级低的流水线要写入寄存器堆中的数据暂时存入到一个缓冲队列中。三条流水线优先级按照它们使用频率和重要性进分配，即 Load/Store 流水线优先级最高，乘加流水线次之，除法开方流水线的优先级最低。

3. 顺序提交队列

为了使算术运算指令顺序提交，需要记录进入流水线的顺序[9]。当算术运算指令从公用的执行 1 级进入执行 2 级时同时将其指令编号存入顺序提交队列。队列设计可以存储下乘加流水线和除法开方流水线执行 2 级到最后执行级的所有指令的编号。该队列是一个先进先出的队列，每个新进入的信息写入最低的空单元，并由一个指针来指示最低的空单元。队列中的信息始终从最低的单元中读出，每读出一次信息，队列中其余的有效信息按序传递到下一级，否则滞留在本级。

4. 处理器冲突处理机制

为提高处理器的处理速度，引入流水线技术。多条流水线并行处理可以进一步提高处理器的处理能力，同时由于指令之间的相关也引入了冲突这一矛盾。

解决冲突的方法：使用记分牌。记分牌是使资源充足且没有相关性指令可以乱序执行的一种手段。它负责：①检查冲突，②控制指令发射和执行。

协处理器在发射级探测所有的数据冲突条件，使用源和目的记分牌来保证一条指令所有的源寄存器和目标寄存器中保存有效数据并且能供读写：目标记分牌锁定当前操作的每个目标寄存器。源记分牌锁定当前操作的每个源目标寄存器。

在译码级先决定操作需要使用那些源寄存器和目标寄存器，在发射级检查并更新源、目标记分牌。如果它探测到在发射级的指令和前面的指令有冲突，记分牌就不会更新，指令被滞留在发射级。

3.4 浮点处理器异常处理机制

在遇到可能会出现异常的浮点运算时，为保证系统的正确和持续运行，处理器必须提供一种处理这些异常的机制。

首先我们简单介绍异常引起中断的过程和精确异常的概念。

精确异常处理是指在发生异常时，仅仅对发生异常的指令或其后面的指令进行异常处理；而其前面的指令要保证能够正常结束。所谓“精确”，是指受到异常处理影响的只有产生异常条件的那条指令，所有在此之前的指令在异常被处理前都将被执行完成。异常处理结束后仍将从发生异常的指令开始继续执行。

在一个用于指令并行执行的体系中，系统的顺序执行只是一种抽象。中断将使我们看到程序不是顺序执行的。当一个异常发生，系统的顺序执行被中断时，将会有几条指令出于流水线的不同阶段。为了使中断处理不破坏程序的正常执行，对于没有执行完的指令，我们必需记住它们执行到哪一个阶段，以便在中断处理之后能恢复程序执行。如果处理器是精确异常的，那么用于异常的软件处理就会非常简单。对于一个精确异常的处理器，在异常发生时，会有一个引起异常的指令。该指令前面的所有指令都以执行完，该指令以及该指令以后的指令在异常处理结束前都不会被执行。所以软件作异常处理时，只需找到一起异常的指令，可以完全忽略指令的非顺序执行。例如对于常见的 MIPS 体系结构，几乎所有的异常都是精确异常，即能准确定位恢复程序执行的正确位置。在任何异常产生后，处理器的控制寄存器 EPC 指向中断处理后恢复程序执行的正确位置。在绝大多数情况下，它是产生异常的指令。找出产生异常的指令看似容易，但在有些级数

很多的流水线 CPU 上这件事并不容易。在流水线的 CPU 上，异常可能会发生在指令执行的不同阶段。如果一个指令产生一个异常,这个异常一直要到读写存储器阶段才产生。如果它的后一条指令在取指阶段就产生错误，则后一条指令将先产生异常。从而破坏异常发生的顺序跟指令执行的顺序相同这个约定。为了避免这个问题，被发现的异常情况一直要等到前面的所有指令都不产生异常时才产生异常。在发现异常情况时，该情况只是被记下来沿着流水线传递下去直到某一级。如果在这个过程中以前的指令产生了异常，该异常情况仅仅被简单的忽略掉。

精确异常代价很大。因为它限制了可以流水线执行的可能性。这种限制对于浮点运算尤为严重，因为浮点运算需要很多级才能完成。只有系统确认已发射指令不会产生异常时才会让浮点指令进入算术逻辑单元。

结合冲突处理引入的记分牌，在本协处理器设计用一种近似精确的方法处理异常。在运算流水线的执行 1 级采用异常探测单元检查浮点算术运算指令是否有潜在异常发生。如果有潜在异常则发生中断，调用软件处理该条指令。中断返回后再继续执行后续指令。前一条浮点指令未进行异常探测时，后一条指令不允许发射。

第四章 乘加流水线设计

本章分为两节。第一节着重介绍决定浮点协处理器性能的部件——乘累加器的关键技术。包括加法器数据通路的优化设计，浮点乘法器采用树形结构，舍入合并技术，快速并行加法器结构，组合加法器等内容。第二节介绍了完整的乘累加流水线的结构设计。

4.1 乘累加器关键技术

4.1.1 概述

乘累加器是决定浮点处理器性能的主要运算部件。每秒钟完成的双精度浮点乘累加运算是衡量一个系统浮点性能的标准之一。为提高系统时钟频率，提高流水线吞吐率，乘累加器设计成多级流水线结构。乘累加器，特别是乘法器中的部分积压缩逻辑也占用了浮点处理器运算单元的大部分面积芯片面积。因此，优化和改进乘累加器的性能和面积是浮点协处理器设计的主要任务之一。在实际设计中，为提高运行速度，尽量将串行逻辑并行化，尽量发掘各相关步骤之间并行的可能性。为优化面积，乘累加器内部数据通路尽量做到资源共享。下面首先分析浮点加法的数据通路，提出改进的方案。进而引出整个乘累加器的的数据通路。

4.1.2 单通路浮点加法器结构

浮点数的加法在算法上需要经过对阶、尾数运算、规格化、舍入操作和判断结果正确性等几个步骤[15]。

求指数差 expdiff，并根据指数差较大的数。

(1)根据指数差对较小的操作数进行移位，完成对阶。

(2)根据两操作数符号，确定进行那种运算。尾数相加减。

(3)尾数结果是负数的求出结果绝对值，修正符号位。

(4)求出前导零个数，对结果规格化移位，使之满足 IEEE754 标准格式。

(5)按照舍入模式对结果舍入（+1 或+0）。

调整指数，并进行尾数溢出的结果进行移位修正。检查指数是否溢出，判断

结果正确性，输出最终结果。

图 4.1 给出 IEEE 单通路浮点加法器关键路径。

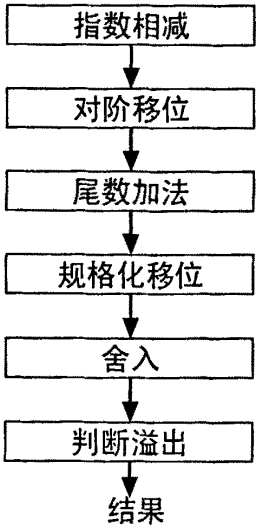

图 4.1 IEEE 单通路浮点加法器关键路径

以上过程每个步骤之间相互依赖，必须顺序执行。其中对阶移位 2，尾数加减 3，规格化移位 5，舍入 6 这些步骤的复杂度都和操作数尾数长度成正比。直接按照该算法实现电路将浪费大量资源，且速度低下。

为提高系统的运行速度，有必要重点研究浮点加法器个运算步骤之间的关系，提高它们之间的并行度。并进一步实现乘累加器需要考虑将乘法器和乘法舍入结合到浮点加法的数据通路上，尽量做到数据通路和资源共享。

4.1.3 双通路浮点加法器结构

传统的单通路浮点加法器功耗大、速度慢，占用资源较大。为提高浮点加法器的性能，使用双通路技术实现浮点加法首先由 Farmwald 提出[16]。Sit 等人、Benschneider 等人和 Birman 等人完成了符合 IEEE 标准的双通路实现[17]。通过对上述简单的浮点加法分析。我们可以发现：

(1)当操作数的指数差绝对值小于等于 1 时，其尾数进行齐移位时至多只需右移一位。当这两个尾数进行减法运算时会产生多个前导零，在规格化时需要等于尾数宽度的左移桶形移位器。

(2)当指数差大于 1 时，两尾数进行对阶操作要进行等于尾数宽度的右移桶形移位。而结果前导零的个数至多为 1，规格化时至多只需左移或右移 1 位，

不需要桶形移位器。

两种情况均需一个尾数宽度的桶形移位电路。因此可根据操作数指数差和将要进行的操作将加法器分成两个独立的数据通路来实现，即双通路加法器[19]。利用这两条数据通路同时算出结果最后再根据划分路径的标准来选择最终的操作结果这种结构示意如下图 4.2 所示。

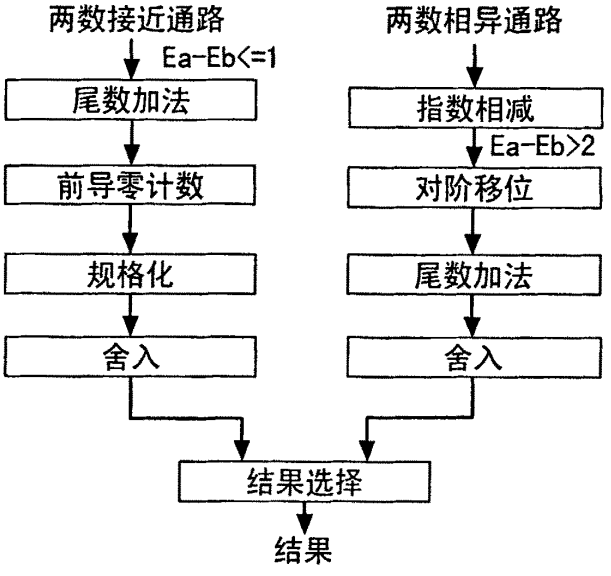

图 4.2 双通路浮点加法关键路径

4.1.4 前导零预测技术

通过分析“两数接近通路”发现左侧尾数加法、前导零计数和规格化的复杂度都和尾数长度成正比，而右侧只有对阶移位和尾数加法的复杂度都和尾数长度成正比。显然左侧路径逻辑延时大于右侧路径。关键路径较大延时通过左侧路径。为减少延时可以将前导零计数同尾数加法并行起来，在尾数进行加法的同时根据两操作数尾数预测结果的前导零个数就会大大减少延迟。如图 4.3 所示。前导零预测电路的原理是通过简单的逻辑电路预测出两个尾数加/减结果的首个“1”的位置，再通过首零计数器算出需要左移的位数，从而在结果计算出来后即可立即通过桶形移位器进行左移操作，使首个“1”移到最左端。

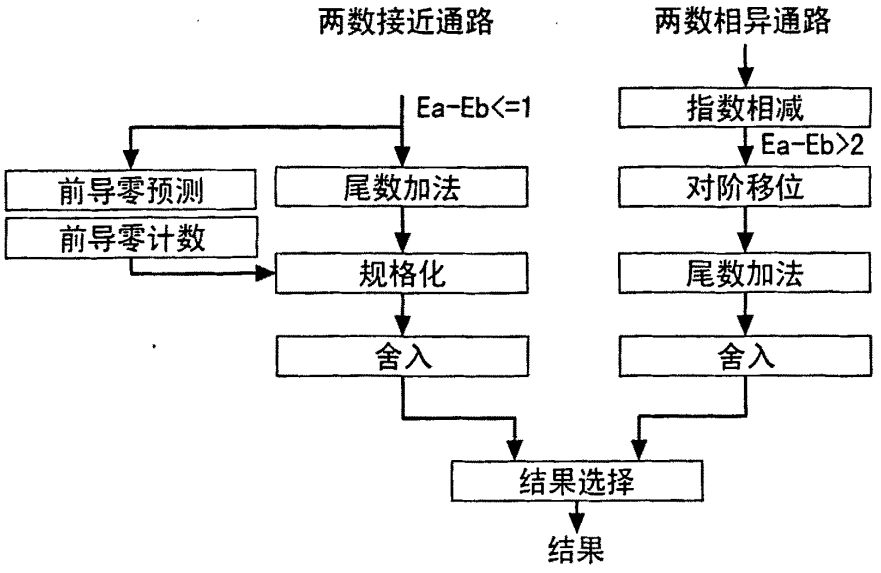

图 4.3 前导零逻辑预测示意图

4.1.5 浮点乘法器

高性能乘法器中乘累加器是关键部件，是实时图像处理和数字信号处理的核心。同时乘法器也处于乘累加流水线数据通路的关键路径上。浮点处理器完成一次乘法操作所需时间往往决定了处理器的系统主频率。

浮点乘法器的主要结构由部分积的译码，部分积压缩，进位传播加法器和舍入逻辑几部分构成。其中进行部分积压缩和舍入是决定乘法器性能的关键。

4.1.6 树形乘法器

树形乘法器是将部分积采用并行的树形结构的进位保留加法器进行压缩，其完成部分积压缩所需要的时间与操作数位数的对数成正比，且同线形阵列结构的压缩器相比，树形结构消耗资源更少。代价是互连的复杂度提高了。典型的树形乘法器结构由 wallace 提出[24]，使用全加器构成进位保留加法器。

在前人基础上，Dadda 提出了(n,m)并行计数器，进一步减少了乘法器的延迟时间[25]。一个(n,m)并行进位计数器就是有 n 个输入，m 个输出的组合逻辑网络。输出表示输入信号为 1 的个数。全加器就是(3,2)计数器。为了便于描述逻辑结构，(n,m)并行计数器也可以表示为压缩器。一个(n,m)压缩器就是上级有 n 个输入，到下一级有 m 个输出的组合逻辑网络。全加器就是(2,2)压缩器。常用的压缩器是(4,2)压缩器。使用(4,2)压缩器可以得到比 wallace 结构更少的逻辑级数。

更为复杂的压缩器也已经被提出来，但是连线比较复杂，不便于 ASIC 实现。

树形乘法器具有很高的性能，但是大数乘法器如双精度浮点乘法器占用资源较大，功耗较高。针对嵌入式领域领域进行优化，可以将乘法器的树形压缩逻辑拆分成两部分，在第一个周期中完成双精度尾数的高位部分积压缩，第二个周期完成双精度的低位部分积压缩。单精度尾数相乘仍在一个周期完成。这样就实现了逻辑复用，减少了资源消耗。代价是双精度乘法需要两周期完成。

4.1.7 舍入合并技术

处于进一步提高系统速度的目的继续深入挖掘加法器可以并行化的逻辑资源。通过对双通路加法器数据通路分析可以看出，舍入操作对加法结果进行+0或+1 操作。可以预先计算出尾数加法所有的可能结果，再根据具体舍入情况选择正确结果。对于于 IEEE 舍入到最近的数的舍入模式，仅有 A+B 和 A+B+1(ulp)两种可选结果。将这个优化方法加入双路径浮点加法算法要求尾数加法器能够同时计算出 A+B 和 A+B+1(ulp)两种结果，这可以通过使用复合加法器（Compound Adder）得以实现。复合加法器是一种特殊的加法器，它能够同时计算出 A+B 和 A+B+1(ulp)两种结果并共享一些内部的硬件，减小了芯片的面积。正确结果的选择要分析舍入位的情况，舍入位包括符号位、最低有效位、警戒位（Guard bit）和粘接位（Stikcy bit）。使用舍入合并技术可以减少位于关键路径上的舍入尾数加法。对于双通路加法，这要求在两个路径上都要使用复合加法器。

对于舍入到负无穷和舍入到 0 的两种 IEEE 舍入模式，加法 A+B 和 A+B+1(ulp) 的结果可能会产生最高有效位的进位溢出，这就需要一位右移，此时移位前的最低有效位现在成为了警戒位。还需要 A+B+2(ulp)这种运算结果。对于 IEEE 舍入模式中舍入到最近的模式，由于仅在原有的最低有效位(现为警戒位)为 1 时才会有舍入加 1(ulp)(加在原有的最低有效位)，因此隐含完成了 A+B+2(ulp)的操作。然而对于舍入到负无穷和舍入到 0 模式，原有的最低有效位(现为警戒位)不为 1 时也会有舍入加 1(ulp)(加在原有的最低有效位)，因此必须显式完成 A+B+2(ulp)操作。为了解决这个舍入的问题，需要在相异路径的尾数加法器前面加入一级半加器。这两个加法器合起来可以通过选择的预加完成 A+B+2(ulp)。

4.1.8 快速加法器结构

浮点加法和乘法都需要较大位宽的快速进位传播加法器。这些加法器往往在关键路径上，因此需要研究快速的进位传播加法器架构。

常见的快速加法器为并行进位结构的超前进位加法器（Carry-Look-Ahead Adder，简称 CLA）。其 16 位简单 CLA 结构如图 4.4 所示。其中每个全加器的进位均可由递推关系通过加数 A 和 B 快速得到。这样每位加法结果可以并行的很快得到。更复杂的多级CLA可以通过将图中的全加器替换成长度短的CLA实现。如 64 位 CLA 可以通过 4 个 16 位 CLA 构成。其逻辑延时为 1 + 4Log4 (n)。

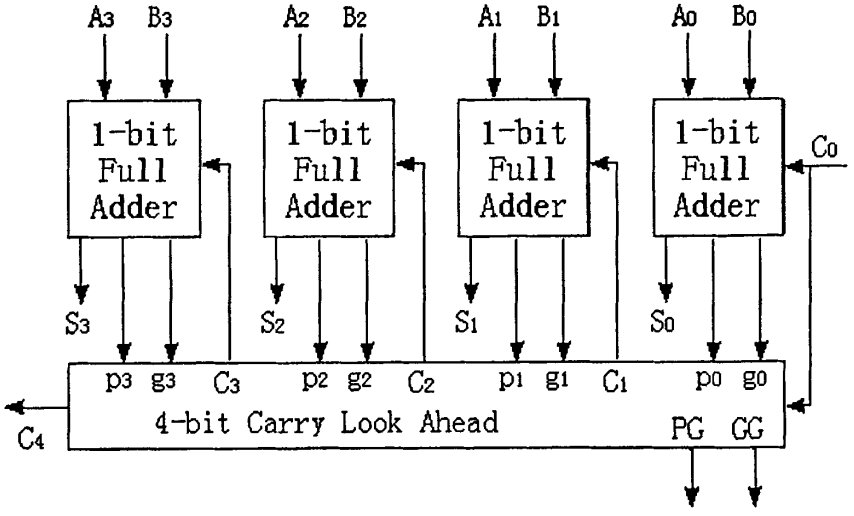

图 4.4 16 位 CLA 示意图

但是随着加法器位宽变宽，随着 ASIC 工艺向深亚微米发展，连线复杂度已经成为制约电路的主要因素。更为规则的版图有利于提高电路速度。对于高速加法器来说，P.M. Kogge 和 H.S. Stone，R. P. Brent 和 H. T. Kung 等人提出的并行前缀加法器[32]（Paralle Prefix Adder，简称 PPA）是一种并行结构的具有规则版图的加法器，它的逻辑延时为 O(log(n))。这种 PPA 主要由并行前缀电路构成，如图 4.5 所示。

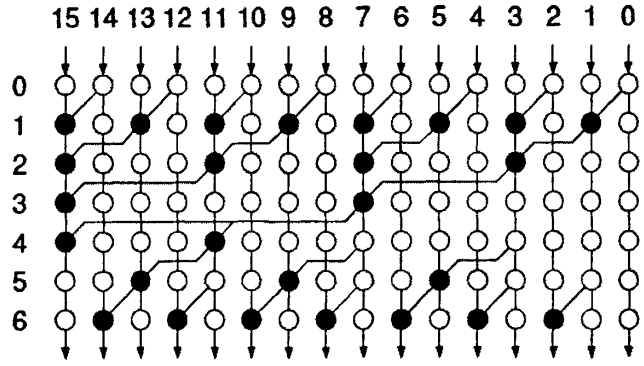

图 4.5 并行前缀电路示意图

图 4.5 中的前缀算符•定义为(g , p)•(g' , p') = (g|(p&g') , p|p') = (G , P)。

4.1.9 组合加法器

复合加法器[34]是一种基于并行前缀加法器的快速加法器。其最大的特点就是同时计算并输出 A+B 与 A+B+1 的结果。可以在上述并行前缀加法器并行前缀逻辑的基础上再添加一行简单的或逻辑实现组合加法器，如图 4.6 所示。

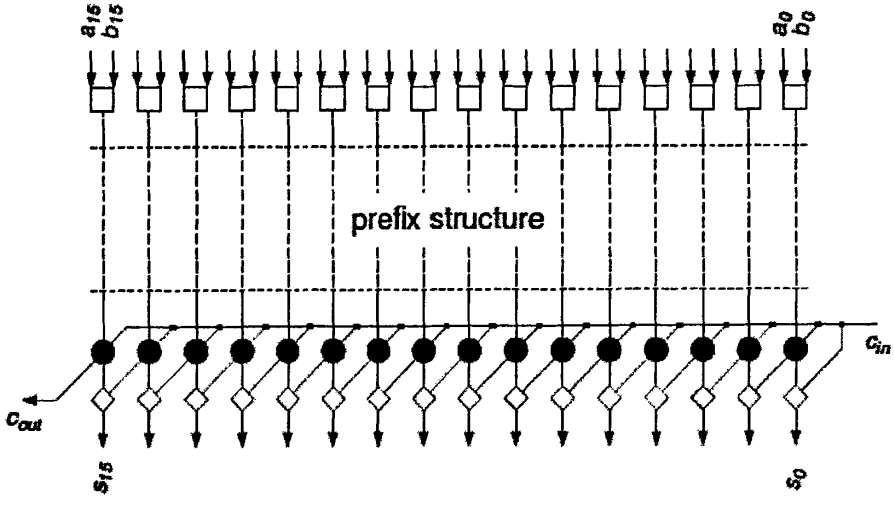

图 4.6 使用并行前缀加法器构成组合加法器示意图

即：

sum0[i]=A+B=A[i]⊕B[i]⊕GC[i];

sum1[i]=A+B+1= A[i]⊕B[i]⊕(GC[i]+PC[i]))

其中 GC 为进位生成逻辑，PC 为进位传播逻辑。

4.2 流水线设计

4.2.1 流水线概述

为提高系统时钟频率，提高流水线吞吐率，乘加流水线设计成多级流水线结构。可以完成单精度或双精度的乘累加操作。向量运算的最终结果等同于顺序完成的一系列等价操作。乘加流水线的指令可分为四类：运算类、比较类、转换类和部分简单指令。其中运算类包括加（ADD）、减（SUB）、乘（MUL）、乘加（MAC）、乘减（MSC）、负乘加（NMAC）和负乘减（NMSC）等指令。比较类包括两个浮点数的比较（CMP）和浮点数和零的比较（CMPZ）。转换类包括单双精度浮点数的转换和浮点数与有无符号整数之间的转换。部分简单指令包括浮点数取反（NEG）浮点数取绝对值（ABS）。

4.2.2 总体结构

乘加流水线分为 8 级流水线，如图 4.7 所示。

第一流水线中进行指数比较、浮点异常探测和符号预测。划分这一级流水线主要是考虑乘累加的异常必须在较大指数确定后才能给出异常的信息，需要较长的延时。

第二级流水线进行乘法运算，包括 Booth 译码和改进的(4,2)压缩器 Wallace 树形压缩。其中树形压缩中同时进行舍入到无穷的增量注入，详见下面章节。

第三级流水线进行乘法的求和及舍入。浮点比较的尾数作差也在这一级的加法器中完成。

第四级流水线选出较小指数的尾数进行对阶移位，并从移位及未移位的尾数中选出加法操作数供下一级求和。同时根据乘法的结果进行指数调整。浮点转整数的移位也在这一级的移位器中进行。

第五级流水线进行求和及前导零预测和前导零计数。

第六级流水线进行结果规格化，并根据求和结果得出最终符号和最终指数。整数转浮点数的移位也在规格化移位器中进行。

第七级流水线进行结果舍入，并选择出最终结果。

第八级是写回级。

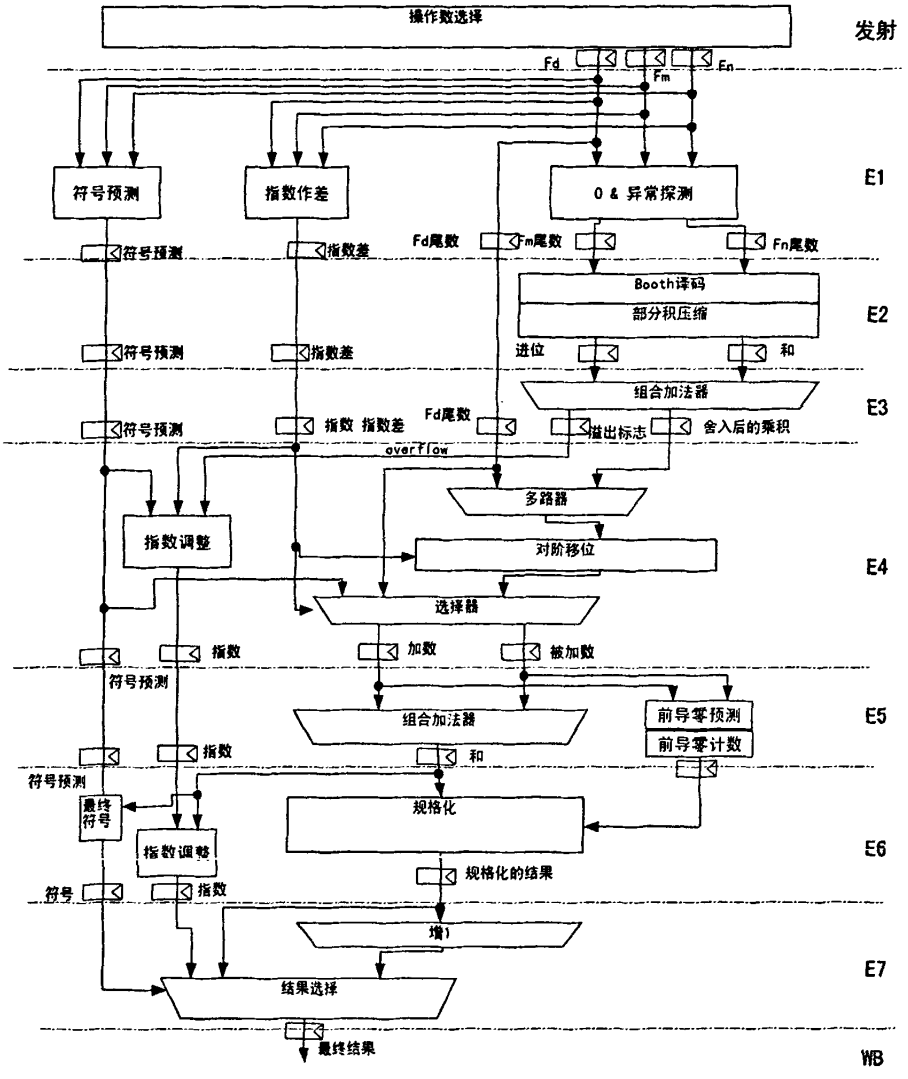

图 4.7 乘加流水线结构图

4.2.3 乘法器结构

将乘数扩展为 54 位。结果是一个乘法阵列。其中 x 为 53 位，y 为 54 位。使用 booth 译码得到的部分积，使用 wallace 树压缩乘积项，最后使用 108 位快速加法器计算和与进位项。示意图如 4.8 所示。

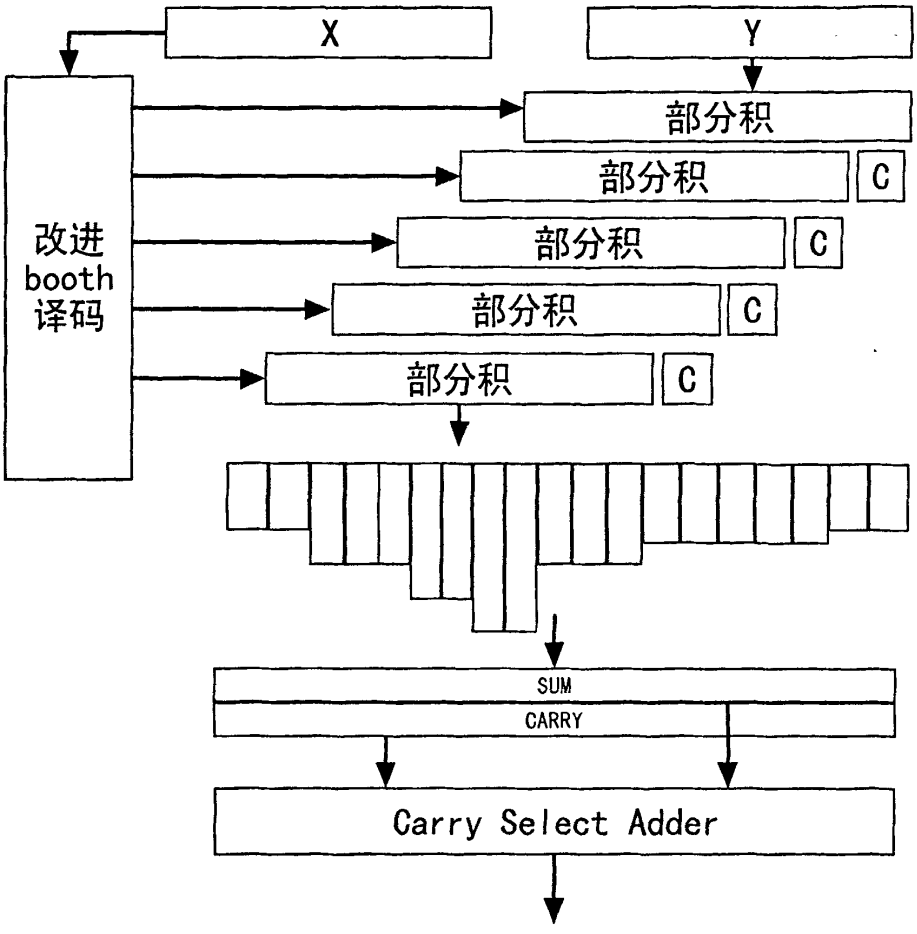

图 4.8 乘法器结构图

在第一个单元，被乘数跟乘数逐位相乘，来产生部分积，在这一步骤中，我们采用 Booth 译码[23]，从而使部分积的个数减少为原来的一半。第二个单元则是最重要的，它不仅是三个单元中最复杂的，而且它决定了整个乘法器的速度。在高速设计中，华莱士树[24]的方法则是最经常被采用来计算部分积，最后产生两列部分积，SUM 和 CARRY，它们则在最后第三单元进行相加，通常我们采用快速加法器，如 CLA，CSA，从而输出最后结果，当然，所乘的数的位数则决定了乘法器的延时。位数越多，则延时越长。

4.2.4 Booth 译码

改进后的 Booth 译码采用并行译码。采用二进制补码表示的 Y 为：

$$Y = -Y_{n-1}2^{n-1} + \sum_{i=0}^{n-2} Y_i 2^i \quad (1)$$

Y 可以被改写成：

$$Y = \sum_{i=0}^{n/2-1} (Y_{2i-1} + Y_{2i} - 2Y_{2i+1})2^{2i} \quad (2)$$

在公式(2)中，括号中表达式的值可以是{-2，-1，0，+1，+2}。如表 6 所示乘数 B 按照 3 位一组被分成重叠的组。每组对应相应的对被乘数 A 进行的并行乘法操作{-2X，-X，0，X，2X}。得到的部分积如图 8 所示。所有这些乘法操作都可以通过移位和求补运算得到。（其中 b-1=0）对部分积求和时，为避免浪费资源，对以二进制补码表示的各部分积符号扩展时采取符号补偿的方法：

对每一个部分积最高位取反；

在最高位左面补 1；

按照原被乘数 A 的最高位给部分积的和对应位置加 1。

其中加 1 可以用对应位加法器进位输入为 1 来实现。

表 4.1 Booth 译码列表

Ym+1	Ym	Ym-1	Booth Op	Dir	Shift	Add
0	0	0	0x	0	0	0
0	0	1	1x	0	-	1
0	1	0	1x	0	-	1
0	1	1	2x	0	1	0
1	0	0	-2x	1	1	0
1	0	1	-1x	1	-	1
1	1	0	-1x	1	-	1
1	1	1	-0x	1	0	0

BOOTH 译码涉及到的操作有：'direction','shift','addition'，direction 操作数决定被乘数究竟是取其本身 X,还是要取反~X，举个例子来说，如果乘数 Y 有一个三个一组分成的数' 101 '，我们则需要取被乘数 X 的补码。但是，这很难实现，因此我们的部分积只要求将 X 取反，然后再加一即可达到要求。Shift 操作数意味着把操作数向左移动一个位置，如果 Y 中有个待译码的数组'011','100'，我们则都需要将 X 左移一位。Addition 操作数则意味着在部分积的最低位加上 1。其中'direction','shift','addition'跟 Ym-1, Ym, Ym+1 的关系为：

Direction = Ym+1 ;

Shift = Ym-1· (Ym+1⊕ Ym) + Ym-1_bar· (Ym+1⊕ Ym) ;

Addition = Ym-1⊕ Ym ;

其中对被乘数 X 的操作主要如下所说：{-2X，-X，0，X，2X}

-2X:　将 X 左移一位，再取反，并在其最低位加 1。

-X:　将 X 取反，在其最低位加 1。

0 :　将 X 的各位全都用 0 代替。

X:　X 的各位保持不变。

2X:　将 X 左移一位。

这样即可得到各个部分积。

1. 关于 BOOTH 译码中的符号扩展问题

以 16×16Bit 的乘法加以说明，部分积生成电路根据 Booth 编码结果生成部分积，当被乘数是 16 位时，部分积是 17 位。由于 Booth 算法所产生的部分积是有符号数，所以在进入 Wallace 树开始运算前需要进行符号扩展，符号扩展本身需要占用电路，耗费时间，而且符号扩展还增加了 Wallace 树的规模。所以应对其进行改进。经 Booth 编码后生成的部分积为 8 个，假设 8 个部分积的符号位分别为：S7 ，S6 ，S5 ..... S0 。对由于符号扩展所引起的附加值 S 进行化简：

$$S=\sum_{i=0}^{7} S_i \sum_{t=2i+16}^{31} 2^t = \sum_{i=0}^{7} (1-\overline{S_i})(2^{32}-2^{2i+16})$$

$$=8\times 2^{32} - \sum_{i=0}^{7} \overline{S}_i 2^{32} - \sum_{i=0}^{7} 2^{2i+16} + \sum_{i=0}^{7} \overline{S}_i 2^{2i+16}$$

$$=7\times 2^{32} - \sum_{i=0}^{7} \overline{S}_i 2^{32} + (2^{32} - \sum_{i=0}^{7} 2^{2i+16}) + \sum_{i=0}^{7} \overline{S}_i 2^{2i+16}$$

$$=7\times 2^{32} - \sum_{i=0}^{7} \overline{S}_i 2^{32} + (\sum_{i=0}^{31} 2^i - \sum_{i=0}^{7} 2^{2i+16} + 1) + \sum_{i=0}^{7} \overline{S}_i 2^{2i+16}$$

$$=\sum_{i=0}^{6} 2^{32} - \sum_{i=0}^{7} \overline{S}_i 2^{32} + \sum_{i=0}^{7} \overline{S}_i 2^{2i+16} + (2^{29}+2^{27}+2^{25}+\cdots\cdots+2^{19}+2^{17}) + 2^{16} \quad \cdots\cdots(1)$$

由一表达式即可看出，我们仅需要做到以下几步：

将每个部分积的符号位取反；

将 1 加在每个部分积的左边；

按照原被乘数 X 的最高位给部分积的和对应位置加 1。

4.2.5 改进的华莱士树压缩逻辑

快速乘法器由 booth 译码得到部分积后，这些部分积对齐形式如图 4.9 左的平行四边形。

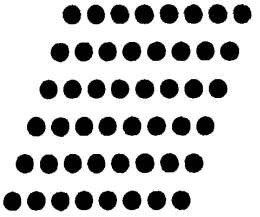

图 4.9 部分积示意图

使用华莱士树方法对部分积压缩，最终得到 Carry 和 Sum，将两者用 CLA 高速加法器相加的最终乘积。54 位（经符号扩展后）尾数相乘，得到 28 个 54 位部分积。示意图如图 4.10 所示。

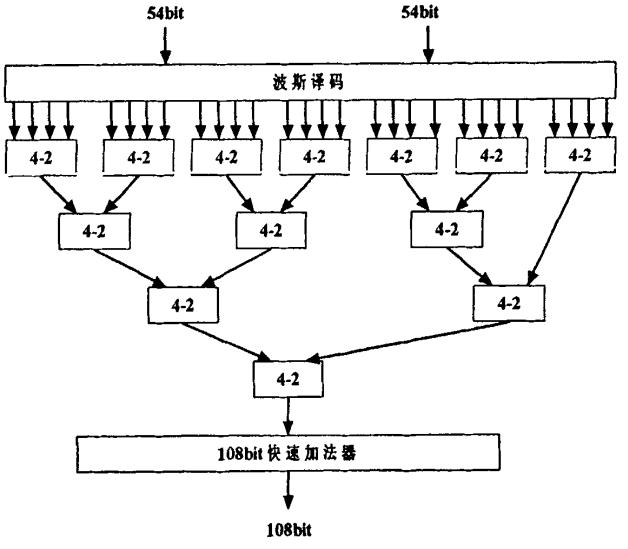

图 4.10 54bit 尾数乘法示意图

1. 压缩器

偶数位部分积乘法树常用 4-2 压缩器和 3-2 压缩器压缩构成：

3-2 压缩器在逻辑结构上就是一个 CSA 类的全加器。

$$Sum[j] = P_0^j \otimes P_1^j \otimes Carry[j-1]$$

$$Carry[j] = P_0^j \cdot P_1^j + P_0^j \cdot Carry[j-1] + P_1^j \cdot Carry[j-1]$$

示意图如图 4.11 所示。

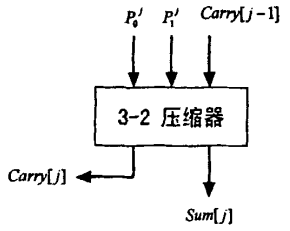

图 4.11 3-2 压缩器结构图

传统 4-2 压缩器在逻辑结构上是有两个 CSA 类的全加器构成。

$$Sum[j] = P_0^j \otimes P_1^j \otimes P_2^j \otimes P_3^j \otimes C_{out}[j-1]$$

$$Carry[j] = (P_0^j \otimes P_1^j \otimes P_2^j \otimes P_3^j) \cdot C_{out}[j-1] + \overline{(P_0^j \otimes P_1^j \otimes P_2^j \otimes P_3^j)} \cdot P_3^j$$

$$C_{out}[j] = (P_0^j \otimes P_1^j) \cdot P_2^j + \overline{(P_0^j \otimes P_1^j)} \cdot P_0^j$$

示意图如 4.12 所示。

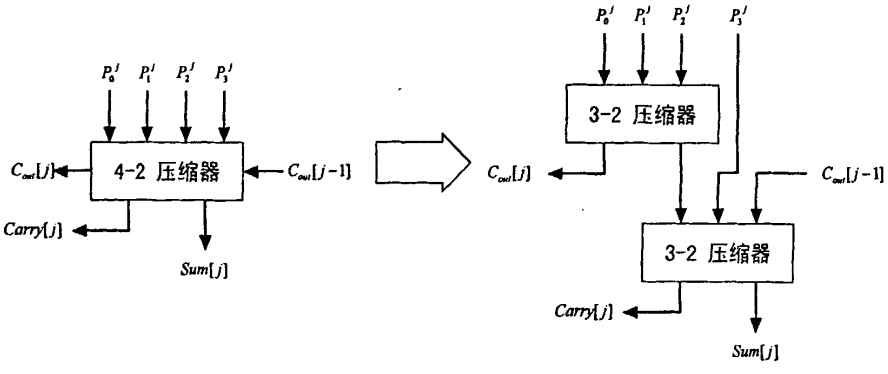

图 4.12 4-2 压缩器结构图

我们实际的乘法器是一个 54×54bit 乘法器，这样的话，经过 BOOTH 译码后，54 位的乘数被译成 27 位的数据，这样相承后的部分积应该是有 27 个，在加上最高位补一，还有符号位左边补一的话，总共部分积应该是有 28 个部分积。部分积相加单元采用压缩器，如上面所说相加，而在实际运算中，我们并不是按 28 行的来进行压缩，而是把这个过程分成了两个部分分两次完成的。第一周期中压缩部分积的后半部分，对于单精度，前两行置 0；对于双精度则直接压缩后面十五个部分积和为了舍入而加入的一行。第二个周期再把上一个周期压缩的结果作为部分积中的两行，完成全部的压缩。第一周期的结果，作为第二周期的输入与之对齐并进行压缩。其结果示意图如图 4.13 所示。

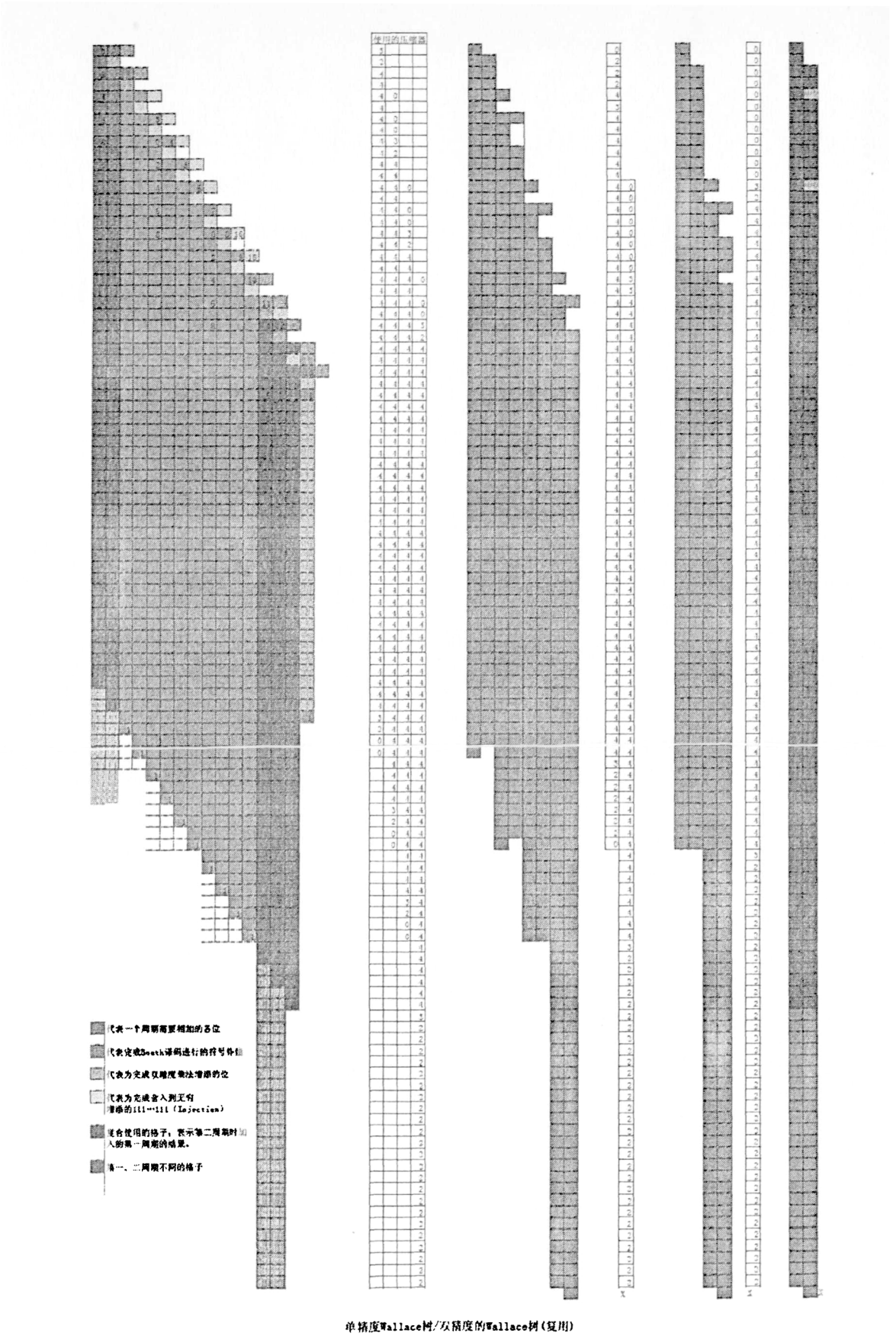

图 4.13 乘法器 Wallce 树压缩示意图

4.2.6 乘积部分积舍入实现

IEEE754 规定的舍入模式中较复杂的的是舍入到最近模式，它可以通过转化为先做舍入到最近的上限(round to nearest up)，再根据截去位是否为 1000…00 将保留数位的最低位置零来实现。由于符号不参与浮点运算，因此舍入到正无穷和舍入到负无穷均可通过在尾数的截去部分每一位加 1 来实现。

符合 IEEE754 标准的单精度数据二进制表示为 $01.x_{-1}x_{-2}\cdots x_{-22}x_{-23}$，双精度表示为 $01.x_{-1}x_{-2}\cdots x_{-51}x_{-52}$，其范围都属于(2, 1]。经过浮点乘法的部分积压缩后得到进位保留形式的结果如果直接相加后得到的最终结果的绝对值的范围是(4, 1]，。假设舍入需要截去的数位最高位为 R 位，保留的数位的最低位为 L 位。对于（2, 1]范围内的双精度结果，如果截去的数值 $x_{-53}x_{-54}x_{-55}\cdots$ 不全为零，则在保留的数位的最低位 L 上加 1。就是 $01.x_{-1}x_{-2}\cdots x_{-51}x_{-52}+00.00\cdots01_2$。对于范围在(4, 2]内的双精度结果，由于已经超出了 IEEE754 规定的标准双精度数据的范围，称之为发生了溢出。为将其装化为 IEEE 标准形式需要将尾数结果右移一位，并对指数加 1。因此 $10.x_{-1}x_{-2}\cdots x_{-51}x_{-52}$ 或 $11.x_{-1}x_{-2}\cdots x_{-51}x_{-52}$ 经右移后，实际需要截去的是原来的 $x_{-52}x_{-53}x_{-54}x_{-55}\cdots$ 位，舍入所加的 1 同未溢出相比也加在了原结果的更高一位。就是 $1.0x_{-1}x_{-2}\cdots x_{-51}x_{-51}+00.00\cdots01_2$ 或 $1.1x_{-1}x_{-2}\cdots x_{-51}x_{-51}+00.00\cdots01_2$。溢出结果的舍入也可以等价为在原来的 L 位的左边一位上加 1，并将结果右移。同时，如果原来在（2, 1]范围内的结果经舍入加 1 后结果落在了(4, 2]的范围内，也转化为了溢出的情况进行处理。如图 1 所示，以上都是在进位保留形式的结果加出最终结果后得到的结论。如果想不加出最终结果，直接通过进位保留形式的结果得到舍入的结果，则必须对溢出情况进行预测。舍入过程见图 4.14 所示。

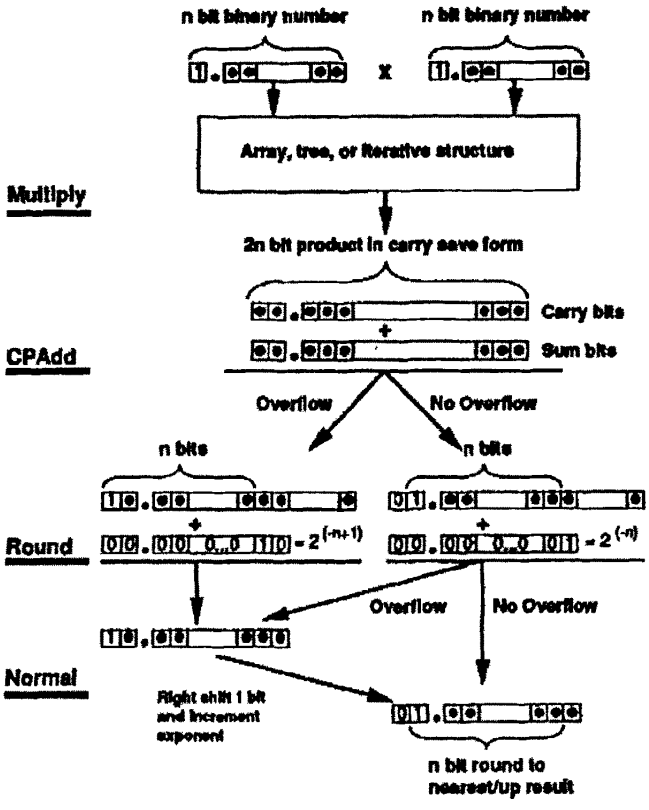

图 4.14 舍入过程示意图

现有的技术一是由 Guy Even 和 Peter-M. Seidel 提出的基于注入（injection）的 ES 舍入算法[30]。即在部分积压缩时将注入位同时相加。现有技术一的缺点是没有提供同时支持舍入到双精度和同时舍入到单精度的解决方案。当单精度和双精度舍入同时进行时，部分积压缩时的注入位选择较为复杂，容易成为系统瓶颈。

现有的技术二是由 Santoro 提出的基于注入预测位来减少备选舍入结果的算法[31]，以及 Quach 等在此基础上提出的改进算法。现有技术二的缺点是主要涉及舍入到最近的模式，没有完整集成各种舍入模式，更没有涉及到单双精度同时舍入。

具体设计中我们在以上两种方法的基础上，针对浮点协处理器的特点提出了自己的舍入方案。基本思路是通过舍入到最近的上限（Round to nearest up）再修正最低位来实现舍入到最近。

设按目标精度保留的数值最低位为 L(Low)，截去数值最高位为 R(Round)。经观察，舍入到最近总是可以通过在 R 位加 1，并在 R 右侧数均为 0 时将 L 位

置零得到。如图 4.15 所示，乘法结果高位可能是二进制 01.xx,10.xx,11.xx 三种情况。称后两种情况为溢出（overflow），最高位溢出用 Rv 表示。不溢出的情况下，舍入是在 R 位加 1 得到。溢出时，在原来不溢出情况下的 R 位左侧加 1。然后将结果右移一位，舍入后溢出的情况也包括在内。

对溢出和不溢出两种情况加以位置不同可以转化为选择在不溢情况 R 位加 1 或加 102 来实现。R 位所加的 1 称为 Rin。R 位右侧进位称为 Cin。待舍入结果为进位保留形式的 sum、carry，权重为 n 的 sum[n]和 carry[n]对应相加。R 位需要相加的项是：Rsum，Rcarry，Rin，Rv，Cin。如图 4.15 所示。

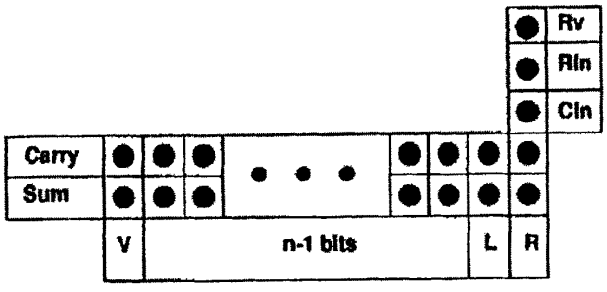

图 4.15 R 位需要相加项示意图

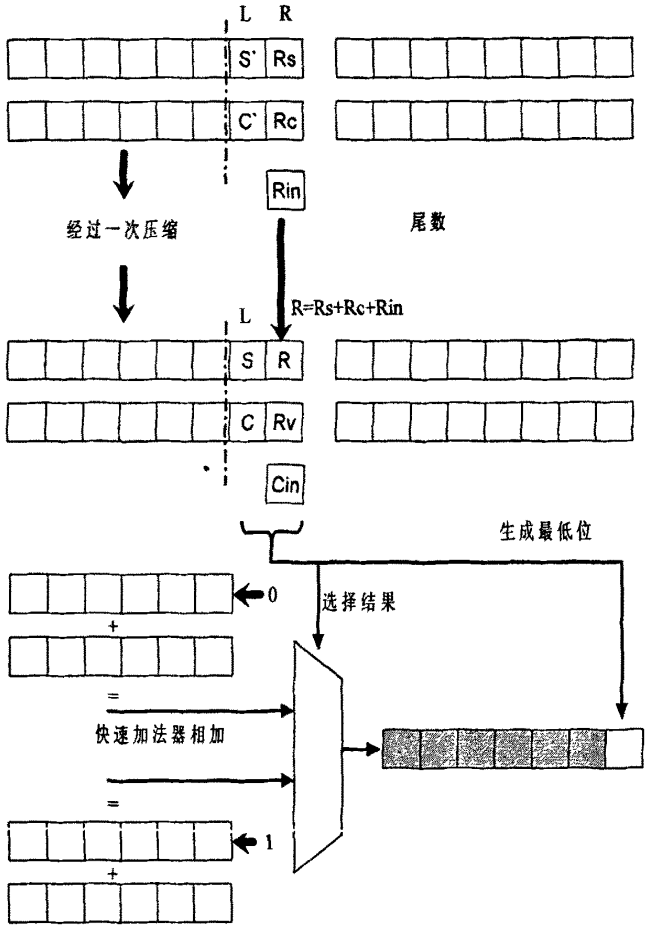

图 4.16　舍入到最近结构示意图

我们首先说明只舍入双精度的情况、再说明双精度和单精度的结合情况。对于双精度，将进位保留形式的待舍入数据分为四段并行处理。即结果高位（sum[1:-51]、carry[1:-51]）、L 位（sum[-52]、carry[-52]）、R 位（sum[-53]、carry[-53]）黏着位（sum[-53:-107]、carry[-53:-107]）四段。高位通过一行半加器压缩。L 位通过一个半加器压缩。R 位上的三个数 Rs、Rc、Rin 通过一个全加器压缩成一位。黏着位通过超前进位链生成向 R 位的进位 Cin。使用超前进位链生成 Cin 的进位的目的是提高速度，避免生成 Cin 的逻辑成为关键路径，同时为集成单精度及舍入到无穷模式提供便利。L 位压缩得到 S、C。R 位压缩后得到 R。结果高位压缩后得到 sum1 和 carry1。将 sum1 和 carry1 送入一改进的并行前缀加法器（Parallel-Prefix Adder）相加。该加法器可以一次性得到 sum1+carry1 和 sum1+carry1+1。再用 S，C，R，Rv,Cin 的不同情况，生成 L 位向高位的不同进

位(0 或者 1)。根据此进位选择高位相加的 sum1+carry1 或 sum1+carry1+1 结果来做为舍入结果的高位部分。同时经过逻辑单独算出最后一位，即最终结果的 L 位。

舍入到最近需要判断 tie 情况。tie 情况指的是待舍入结果中丢弃的部分表示的数值是其等位长数值上限和下限的平均值。此时待舍入数值位于两个最接近数值的中间。如二进制 110.1000 舍入到整数。1000 就等于(0 + 1111)的二分之一。IEEE754 标准规定这种情况下结果舍入到最近的偶数。由于我们舍入到最近是通过舍入到最近的上限实现的。因此仅需要将舍入到最近的上限得到结果的最低位置 0 即可得到舍入到最近的结果。设 sum[-54:-107]+carry[-54:-107]为零的标志是 Z。判断 tie 情况需要依据 Rsum、Rcarry、Cin 和 Z。分为溢出和不溢出的情况。不溢出时 Rsum、Rcarry 为 sum[-53]、carry[-53]，Cin 为 R 位的 carry in 即 Cin[-53]，Z 为 sum[-54:-107]+carry[-54:-107]为零标志；溢出时，Rsum、Rcarry 为 sum[-52]、carry[-52] ，Cin 为 R 位的 carry in 即 Cin[-52]，Z 为 sum[-53:-107]+carry[-53:-107]为零标志。Z 可以通过进位保留形式的 sum[-53:-107]+carry[-53:-107]的结果快速预测得到。对结果各位是否全为零的快速预测逻辑表达式如下：

z[i]=~((a[i]^b[i])^(a[i-1]|b[i-1])) -107<i<-53

再对结果各位的快速预测相与即可得到 Z。如 $Z_{-54} = \&z[-54:-107]$，$Z_{-53} = \&z[-53:-107]$。

tie 情况判断信号 tie_cond 如下：

$$\text{tie\_cond} = \begin{cases} \sim((\text{sum}[-53]\text{^carry}[-53]\text{^Cin}[-53])\&Z_{-54}) & (\text{Rv} = 0) \\ \sim((\text{sum}[-52]\text{^carry}[-52]\text{^Cinc}[-52)\&Z_{-53}) & (\text{Rv} = 1) \end{cases}$$

1. 舍入到无穷

舍入到无穷模式通过对舍入到最近模式的逻辑结构改进得到。

舍入到无穷和结果符号相关，正数舍入到正无穷，负数舍入到负无穷均需要判断是否在保留数位的最低位加 1。下面文中均假设是正数舍入到正无穷，负数舍入到负无穷的情况。

在前面乘法器部分积压缩阵列求 sum 和 carry 对过程中对舍入将要截去的各位加上注入位 1111...1111。对于双精度 $01.x_{-1}x_{-2}\cdots x_{-51}x_{-52}$ 其注入位为 $2^{-52}-2^{-104}$，

对于单精度$01.x_{-1}x_{-2}\cdots x_{-22}x_{-23}$，由于其在乘法阵列中是采用小数点对齐运算，因此其注入位为$2^{-23}-2^{-104}$。

然后在上述的舍入到最近逻辑结构基础上增添处理舍入到无穷溢出情况的逻辑。最高位溢出仍用 Rv 表示。此时只需在 L 位再加 Rv 即可。因此 L 位需要相加的项是：Lsum，Lcarry，Rv，Cin。

我们还是首先说明只舍入双精度的情况、再说明双精度和单精度的结合情况。对于双精度舍入到无穷，将进位保留形式的待舍入数据分为三段并行处理。即结果高位（sum[1:-51]、carry[1:-51]）、L 位（sum[-52]、carry[-52]）、黏着位（sum[-53:-107]、carry[-53:-107]）三段。高位通过一行半加器压缩。L 位通过一个半加器压缩，得到一位 L。压缩舍入到最近模式 R 位的全加器的输入为零，不会产生进位。黏着位通过超前进位链生成向 L 位的进位 Cin。L 位压缩得到 S、C。结果高位压缩后得到 sum1 和 carry1。将 sum1 和 carry1 送入改进的并行前缀加法器（Parallel-Prefix Adder）相加。得到 sum1+carry1 和 sum1+carry1+1。再用 L，Rv,Cin 的不同情况，生成 L 位向高位的不同进位(0 或者 1)。根据此进位选择高位相加的 sum1+carry1 或 sum1+carry1+1 结果来做为舍入结果的高位部分。同时经过逻辑单独算出最后一位，即最终结果的 L 位。

2. 舍入到零

舍入到零通过直接对待舍入结果截取得到。

在上述舍入到最近和舍入到无穷的结构基础上，仅需要把舍入到最近的 Rin 置零，把舍入到无穷的注入位置零，且生成结果的最低位不判断 tie 情况即可。

单精度数据舍入和双精舍入不同的地方是结果高位、L 位、R 位和黏着位位置不同。分别为结果高位（sum[1:-22]、carry[1:-22]）、L 位（sum[-23]、carry[-23]）、R 位（sum[-24]、carry[-24]）黏着位（sum[-25:-107]、carry[-25:-107]）四段。需要修改上面双精度舍入第一阶段使用的半加器、全加器构成的压缩器。即该级压缩器按如下几段构成：半加器（sum[1:-22]、carry[1:-22]），半加器（sum[-23]、carry[-23]），全加器（sum[-24]、carry[-24]），半加器（sum[-25:-51]、carry[-25:-51]），半加器（sum[-52]、carry[-52]），全加器（sum[-53]、carry[-53]）。

同时需要生成双精度、单精度两套进位 Cin。用超前进位链由（sum[-25:-107]、carry[-25:-107]）产生进位 Cin_sp 和 Cin_dp。

单精度和双精度被加数以小数点对齐方式进入并行前缀加法器。

3. 并行前缀加法器

使用改进的并行前缀加法器同时生成 sum1+carry1 和 sum1+carry1+1。并行前缀加法器的基本原理是定义一对进位生成（carry generate）和进位传播（carry propagate）运算符。然后采用这对运算符构成树形结构排列的进位阵列，缩小每个进位生成节点的扇出，同时为版图设计优化排列结构。在这种加法器的在最后一级进位生成传播结点上可以很容易的再加上一级同进位相关的逻辑，实现结果再加 1，并且不会构成组合回路。

运算符对中的进位生成运算符可以定义为逻辑与，传播运算符可以定义为逻辑或。设并行前缀加法器输入为 a，b，最后一级生成的进位生成消息为 g，进位传播消息为 p。通过 g 和 p 可以得到 a+b。而令 g'=g or p，则通过 g'和 p 可以得到 a+b+1。

4. 结果最低位生成

对于双精度：

$$dp\_lsb = \begin{cases} S \wedge C \wedge (R \,\&\, Rv \mid Cin[-53] \,\&\, Rv \mid R \,\&\, Cin[-52] & \text{(舍入到最近)} \\ S \wedge Cin[-53] \wedge Rv & \text{(舍入到无穷)} \\ S \wedge Cin[-53] & \text{(舍入到0)} \end{cases}$$

对于单精度：

$$sp\_lsb = \begin{cases} S \wedge C \wedge (R \,\&\, Rv \mid Cin[-24] \,\&\, Rv \mid R \,\&\, Cin[-23] & \text{(舍入到最近)} \\ S \wedge Cin[-24] \wedge Rv & \text{(舍入到无穷)} \\ S \wedge Cin[-24] & \text{(舍入到0)} \end{cases}$$

5. 舍入结果生成

设改进并行前缀加法器生成的 sum1+carry1 的结果为 A。sum1+carry1+1 的结果为 B。

舍入到最近的结果选择 A 或 B 与 S，C，R，Rv,Cin 有关。其中 R、Rv、Cin 是同权重的，他们的进位和 S、C 是同权重的。下面的表 4.2 就是按照这个规则生成：

表 4.2　舍入到最近进位真值表

			v	in	Inc（L 是否向前进位）

					0
					0
					0
					0
					0
					0
					0
					0
					0
					0
					0
					1
					0
					1
					1
					1
					0
					0
					0
					1
					0
					1
					1
					1
					1
					1
					1
					1

					1
					1
					1

舍入到无穷的结果最低位和 L，Rv,Cin 有关，三者都是同权重的，如表 4.3 所示。

表 4.3 舍入到无穷进位真值表

	v	in	Inc（L 是否向前进位）
			0
			0
			0
			1
			0
			1
			1
			1

最终的舍入结果表示为如表 4.4 所示。

表 4.4 最终舍入结果选择

nc	v	结果
		{A,dp_lsb}
		{1'b1,A}
		{B,dp_lsb}
		{1'b1,B}

6. 算法整体流程

整个算法流程如图 4.17 所示。

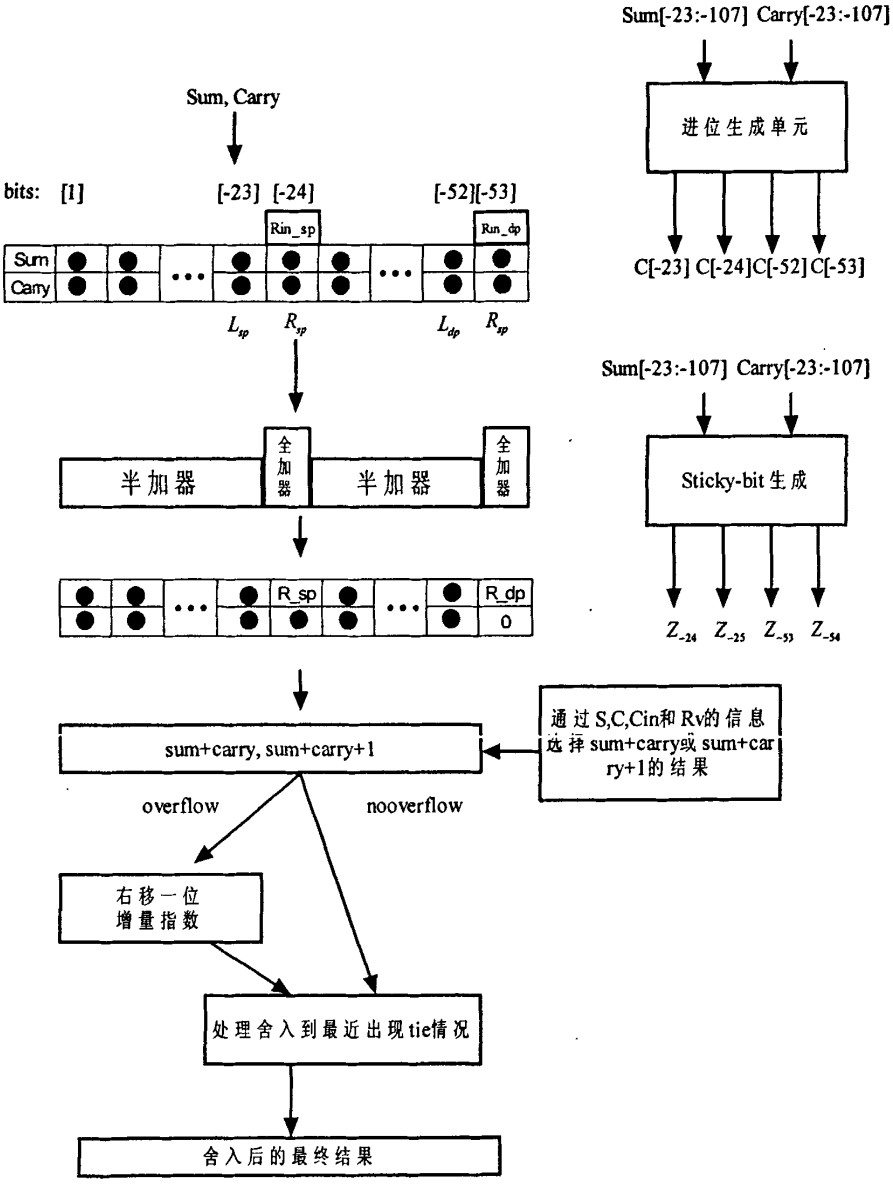

图 4.17 算法流程图

其中：

dp 代表 double precision

sp 代表 single precision

4.2.7 对阶移位器

结构是右移桶形移位器，可采用图 4.18 所示结构设计，选择 D-WIDTH = 108 即可。优点，结构简单，占用面积小。缺点：移位距离越大经过路径越长，达到 6 级传输门延时。延时较大。但在本设计中可以满足要求。

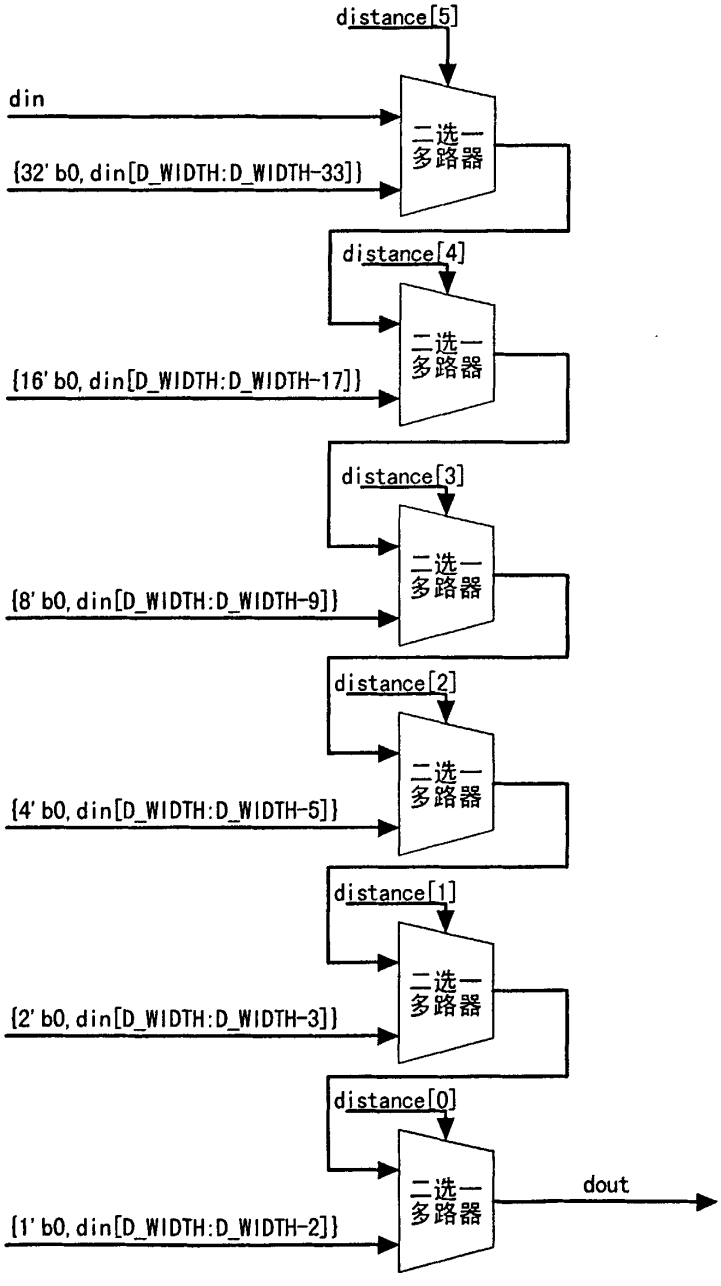

图 4.18 右移桶形移位器结构

4.2.8　前导零预测单元

围绕前导零预测电路已经有了很多的研究并得到广泛的应用[20][21]。以双精度浮点数为例，最高位为符号位，11 位指数位和 52 位尾数位，由于尾数的首位“1”被省略，所以在运算时要先在 52 位尾数前加一位“1”，这 53 位尾数加上后面用于保证精度的扩展位后 A 和 B 都为 m 位。在 A-B 的运算中对 A>B 和 A≤B 两种情况分别进行前导零预测。当 A>B 时,$A-B=A+\overline{B}+1>0$，$A+\overline{B}+1$ 必定会向第 m 位进位，所以此时前导零预测可由式②求出。当 A≤B 时,$A-B=A+\overline{B}+1\leqslant 0$，$A+\overline{B}+1$ 必定不会向第 54 位进位，并且所得到的尾数结果是以绝对值形式存储，所以此时前导零预测可由式③求出。

$$Q_i=\begin{cases}(A_i\odot\overline{B_i})\cdot(A_{i-1}+\overline{B_{i-1}}) & i=1,2,...,m-1\\ (A_i\odot\overline{B_i}) & i=0\end{cases} \qquad ②$$

$$R_i=\begin{cases}(A_i\odot\overline{B_i})\cdot\overline{(A_{i-1}\cdot\overline{B_{i-1}})} & i=1,2,...,m-1\\ (A_i\odot\overline{B_i}) & i=0\end{cases} \qquad ③$$

其中 m 为尾数前补“1”后的位数。公式②和③所预测的前导零位数与实际的前导零位数相同或少一位，在少一位的情况将在最后结果进行归一化移位时进行调整。下图中例举了尾数为 10 位的 A-B 的 4 种情况，Q 和 R 为前导零预测的结果，C 为实际运算的结果。

```
A:  +1010111001          A:  +1010111001
B:  - 0011101011         B:  - 1001101010
        ⇩                        ⇩
A:  0 1010111001         A:  0 1010111001
B': 1 1100010100 +1      B': 1 0110010101 +1
Q:    1001010010         Q:    0001010011
C:  0 0111001110         C:  0 0001001111
```

```
A:  +0010111001          A:  +1010111001
B:  - 1111101011         B:  - 1010111110
        ⇩                        ⇩
A:  0 0010111001         A:  0 1010111001
B': 1 0000010100 +1      B': 1 0101000001 +1
R:    1101010010         R:    0000000101
C:  1 1100110010         C:  1 0000000101
```

前导零预测举例：

其电路结构如图 4.19 所示，表示的是长度为 m 位的前导零预测。

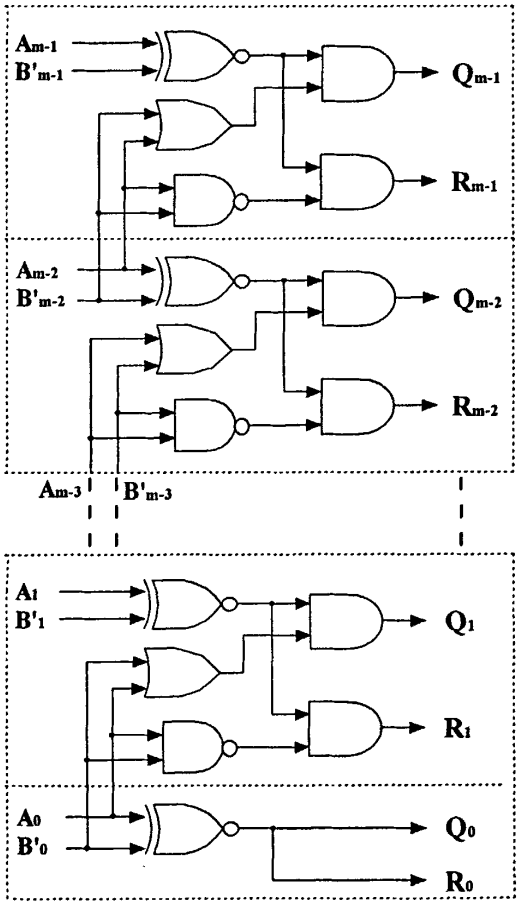

图 4.19 前导零预测电路具体结构

通过此预测电路就无需提前判断尾数 A 和 B 的大小，减法运算的指数和尾数部分结构如图 4.20 所示。

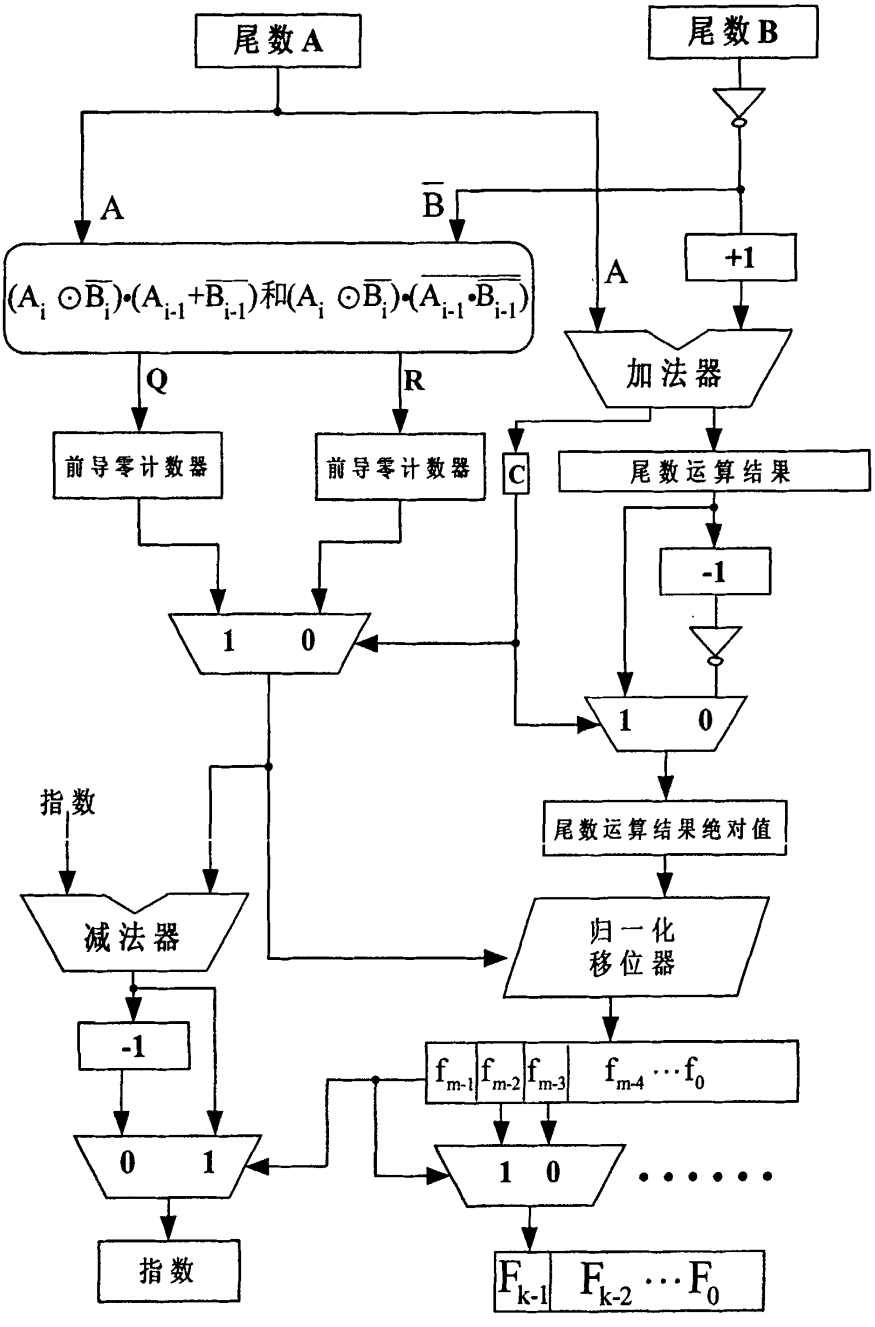

图 4.20 采用前导零预测电路的浮点减法电路结构

图 4.20 中的尾数 A 和尾数 B 是经过首位补“1” 、尾数末位扩展、指数比较并移位后的尾数。A 和 B 运算后产生进位位 C，通过 C 对 Q 和 R 的前导零计数结果进行选择，并控制是否需要将结果转化为绝对值形式。另外进位位 C 和 +/- 控制信号以及原操作数的符号共同决定最终结果的符号位。计算结果由桶形

移位器进行归一化移位，如果前导零预测完全正确则将首个“1”移到最高位，按照 IEEE 754 标准舍去首位的“1”，从第 m-2 位开始取相应的位作为最终结果的尾数，同时将原操作数较大的指数减去前导零的个数作为最终结果的指数。当前导零预测结果比实际少 1 位时，桶形移位器会将首个“1” 移到第 m-2 位，此时从第 m-3 位开始取相应的位作为最终结果的尾数，同时将原操作数较大的指数减去前导零的个数再减 1 后作为最终结果的指数。这两处选择操作由第 fm-1 位进行控制。

E 的非零最高位是不精确的估计。因此首零计数后要根据加法器所得结果最高位情况对首领计数结果进行调整。方法是在 E6 级比较估计值 C 和加法器所得结果最高位，如果相等，则直接利用首零计数的值对结果规格化。不相等，则将尾数规格化结果左移一位修正，指数减一。

4.2.9 前导零计数器

前导零计数器采用树形结构构成。

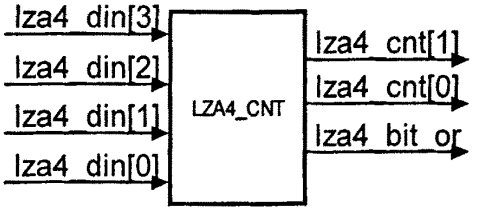

图 4.21 4bit 前导零计数器框图

逻辑关系为：

lza4_cnt[1] = ~(lza4_din[3] | lza4_din[2]);

lza4_cnt[0] = ~(lza4_cnt[1] ? lza4_din[1] : lza4_din[3]);

lza4_bit_or = (~lza4_cnt[1]) | (lza4_din[1] | lza4_din[0]);

8 位计数器结构如图 4.22 所示。

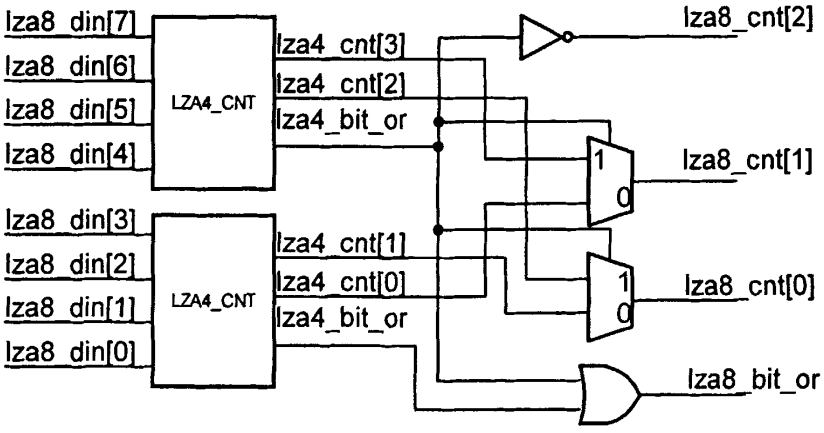

图 4.22 8bit 前导零计数器

逻辑关系为：

lza8_cnt[2] = ~lza4_bit_or[1];

lza8_cnt[1] = (~lza4_bit_or[1]) ? lza4_cnt[1] : lza4_cnt[3];

lza8_cnt[0] = (~lza4_bit_or[1]) ? lza4_cnt[0] : lza4_cnt[2];

lza8_bit_or = lza4_bit_or[1] | lza4_bit_or[0];

同理，位长为 2^n 的 16 位，32 位也是用此方法构成。输出 cnt 的高位为高位子模块 bit_or 位的取反，其余各位是输出 cnt 高位控制的多路器输出，最后输出 bit_or 位是高位子模块和低位子模块 bit_or 位的或。

16 位计数器与 8 位子模块逻辑关系为：

lza16_cnt[3] = ~lza8_bit_or[1];

lza16_cnt[2] = (~lza8_bit_or[1]) ? lza8_cnt[2] : lza8_cnt[5];

lza16_cnt[1] = (~lza8_bit_or[1]) ? lza8_cnt[1] : lza8_cnt[4];

lza16_cnt[0] = (~lza8_bit_or[1]) ? lza8_cnt[0] : lza8_cnt[3];

lza16_bit_or = lza8_bit_or[1] | lza8_bit_or[0];

而用于单精度的 24 位的计数器采用 16 位和 8 位的计数器构成。用于双精度的 53 位计数器采用 48 位和 5 位计数器构成。而 48 位计数器采用 32 位和 16 位计数器构成。对于 48 位计数器这种由位长不同的子模块构成的结构，其构成方法和 16 位、32 位计数器的构成方法还是类似。即将次高位输出的多路器一个选

择支置为 0 即可。进一步，可以将这个多路器用一个与门代替。54 位计数器也是这样构成的。

48 位计数器的结构如图 4.22 所示。

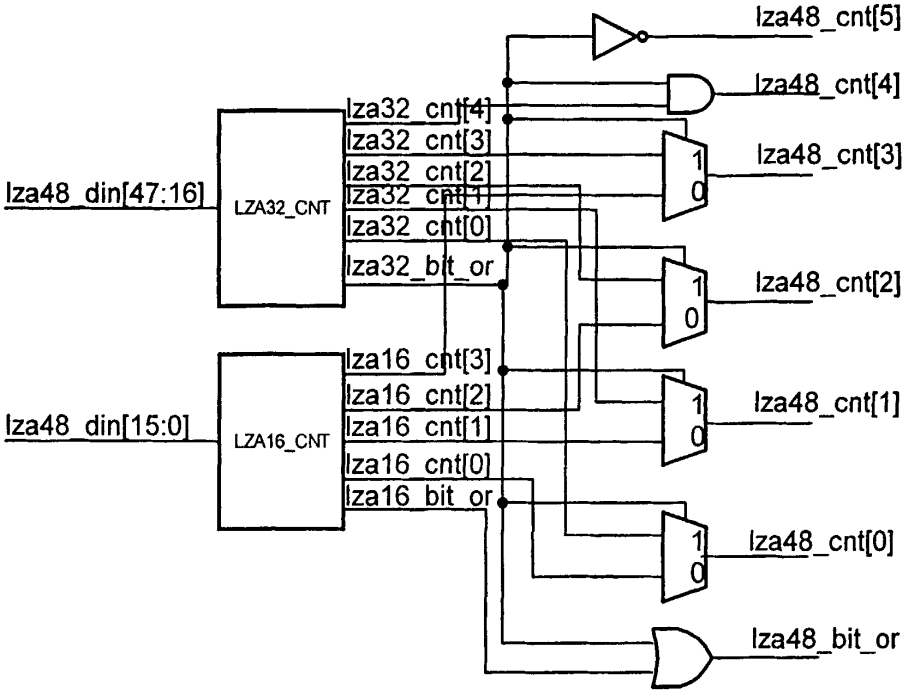

图 4.23 48 位前导零计数器的结构

```
lza48_cnt[5] = ~lza32_bit_or;
lza48_cnt[4] = lza32_bit_or  &  lza32_cnt[4];
lza48_cnt[3] = (~lza32_bit_or)  ?  lza16_cnt[3]  :  lza32_cnt[3];
lza48_cnt[2] = (~lza32_bit_or)  ?  lza16_cnt[2]  :  lza32_cnt[2];
lza48_cnt[1] = (~lza32_bit_or)  ?  lza16_cnt[1]  :  lza32_cnt[1];
lza48_cnt[0] = (~lza32_bit_or)  ?  lza16_cnt[0]  :  lza32_cnt[0];
lza48_bit_or = lza32_bit_or  |  lza16_bit_or;
```

4.2.10 浮点转换的算法

浮点之间的转换主要是舍入和补齐操作。浮点和整数之间的转换主要是知识运算和尾数移位操作。整数和浮点转换可以同算术运算逻辑进行资源共享。现列举算法流程如下。

(1)双精度浮点数转单精度浮点数，如图 4.24 所示。

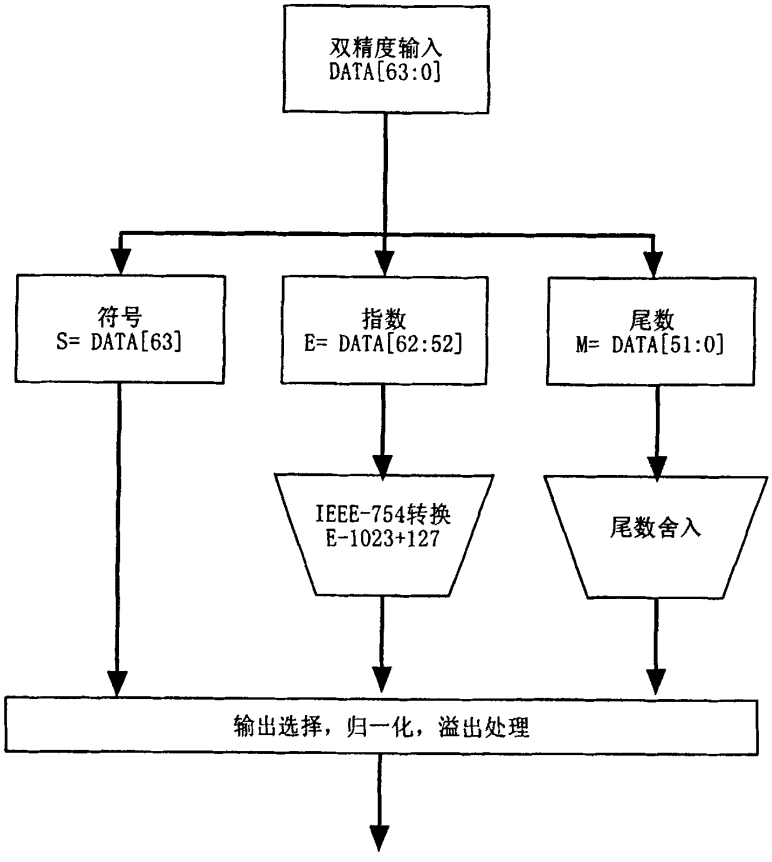

图 4.24 双精度浮点数转单精度浮点数

(2)单精度浮点数转双精度浮点数，如图 4.25 所示。

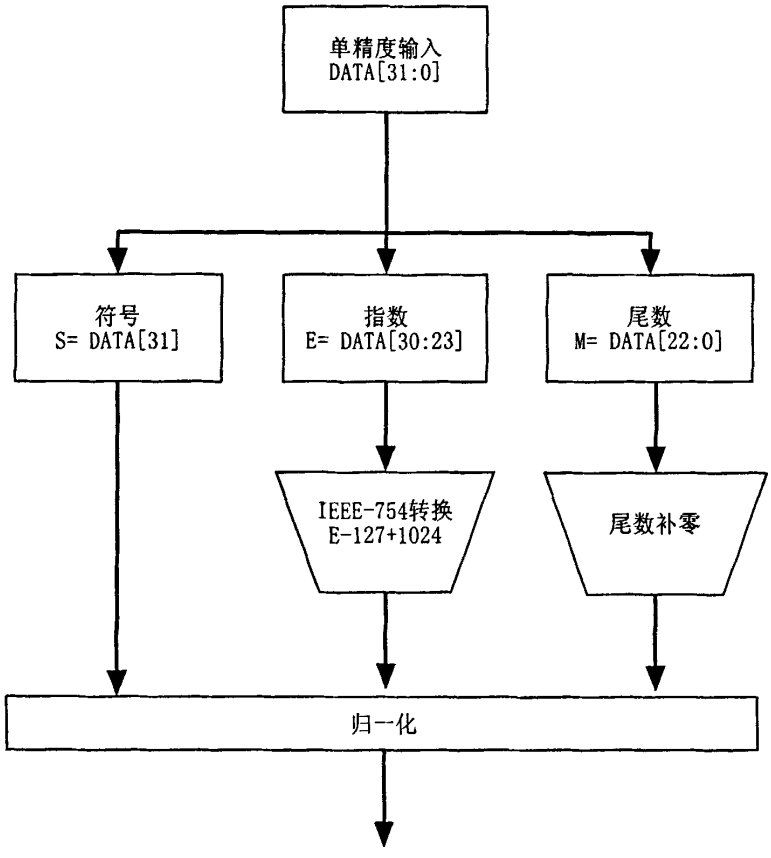

图 4.25 单精度浮点数转双精度浮点数

(3)双精度浮点数转有符号整数。如图 4.26 所示。

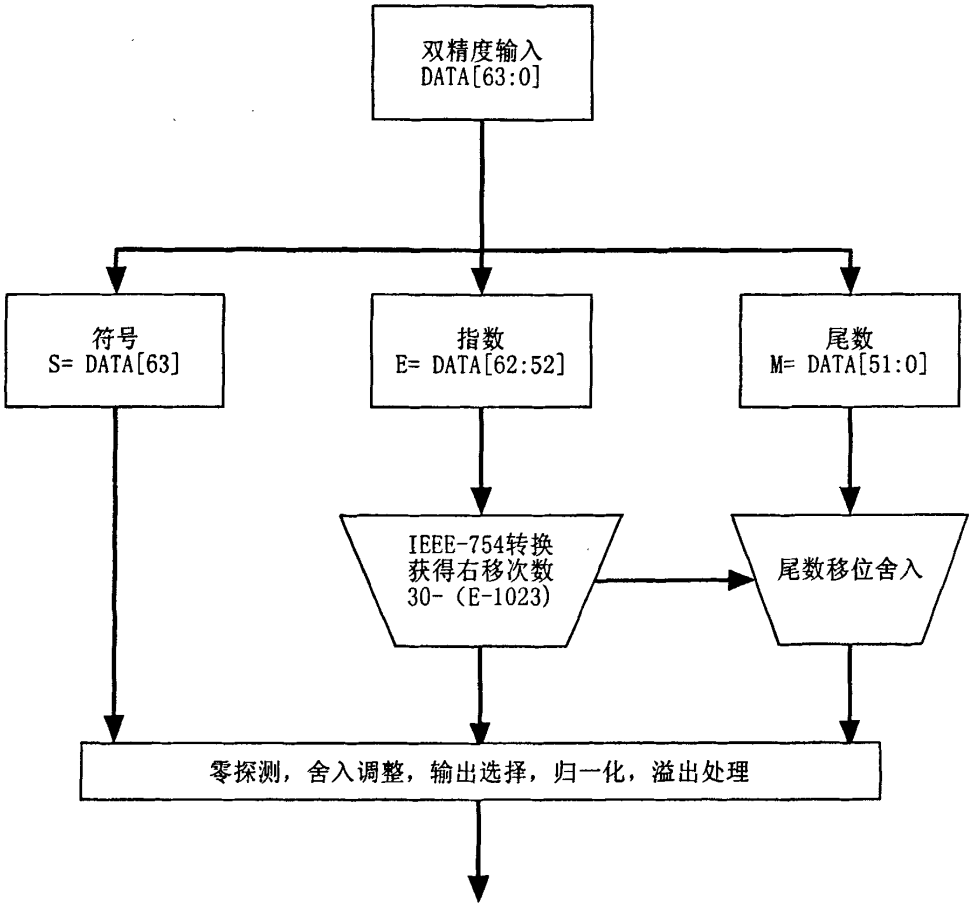

图 4.26 双精度浮点数转有符号整数

(4)双精度浮点数转无符号整数。 如图 4.27 所示。

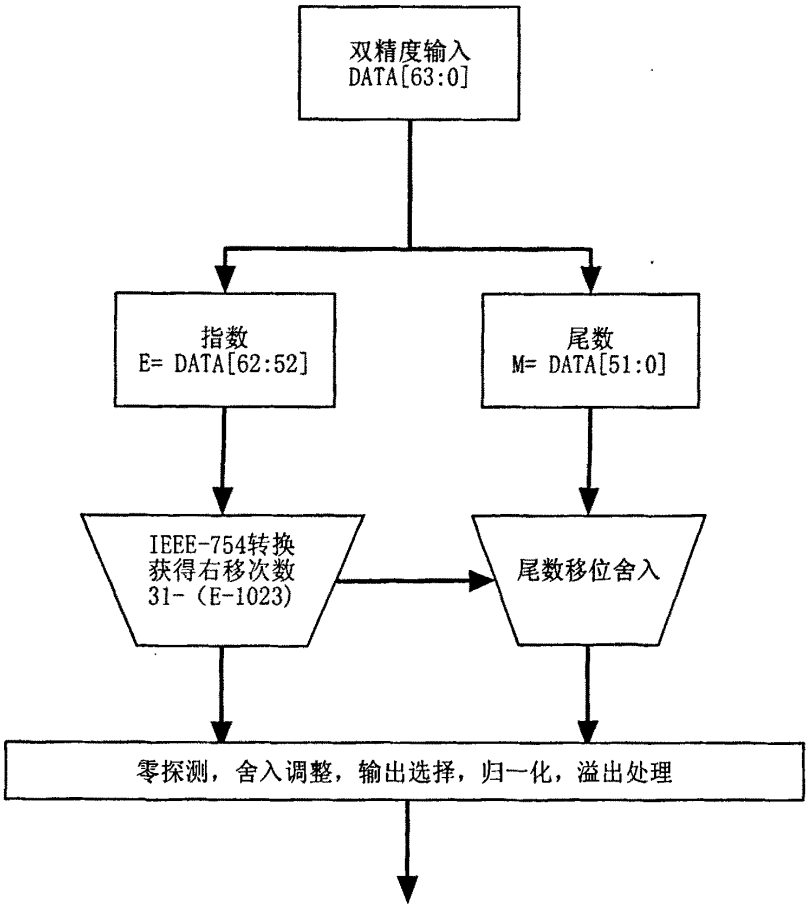

图 4.27 双精度浮点数转无符号整数

(5)有符号整数转双精度浮点数。 如图 4.28 所示。

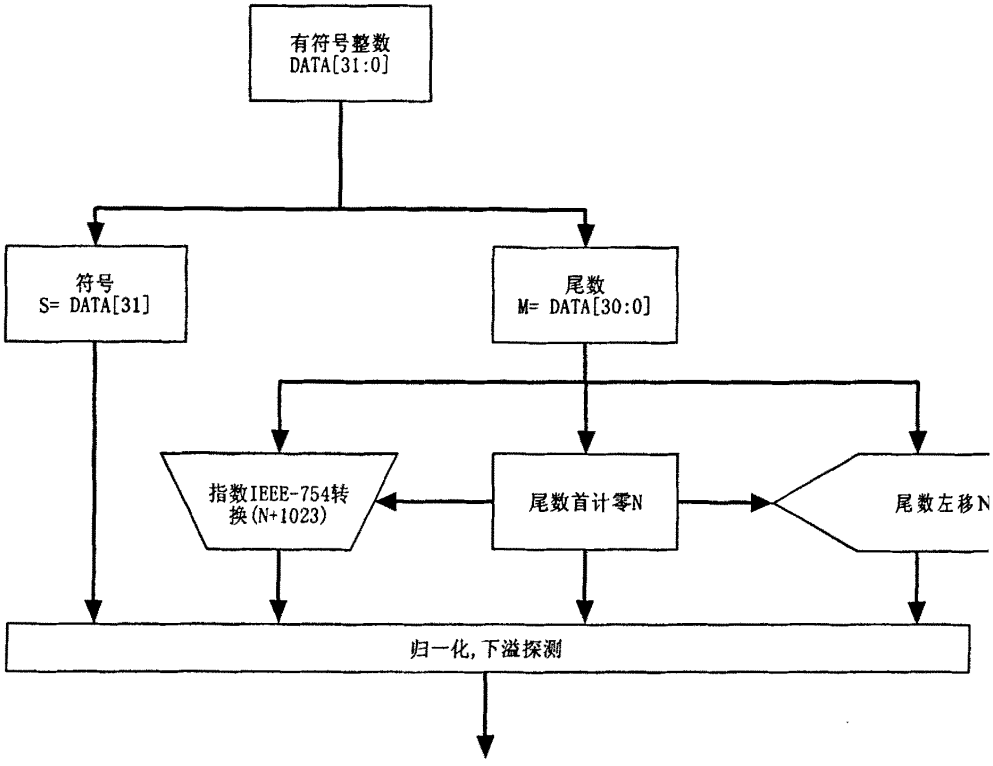

图 4.28 有符号整数转双精度浮点数

(6)无符号整数转双精度浮点数。 如图 4.29 所示。

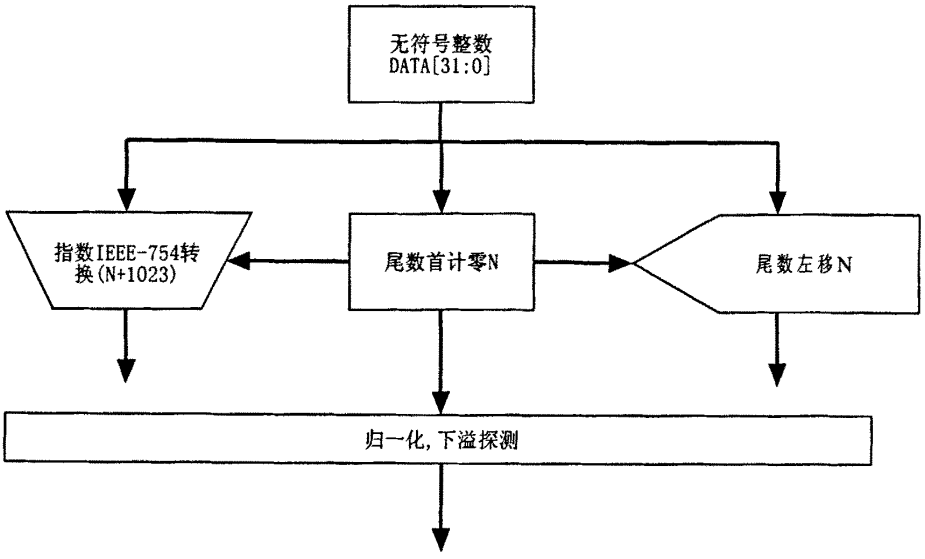

图 4.29 无符号整数转双精度浮点数

4.2.11 指数比较

指数比较加法器实际完成的任务是减法，因此采用了 end_around 的思想，即减法时，负数采用反码表示而不是补码。

2'complement 二进制补码运算负数结果转化为正数结果需要复杂步骤。1'complement 补码形式（即反码表示）更适合浮点加减法。1'complement 补码中正数格式不变，负数 x 的表示方法是 $\bar{x}$（直接取反）。使用这种方法节省了输入和输出负数和正数转化的步骤。简单的 1'complement 补码形式减法举例如下：

计算： 01010100 - 01000011

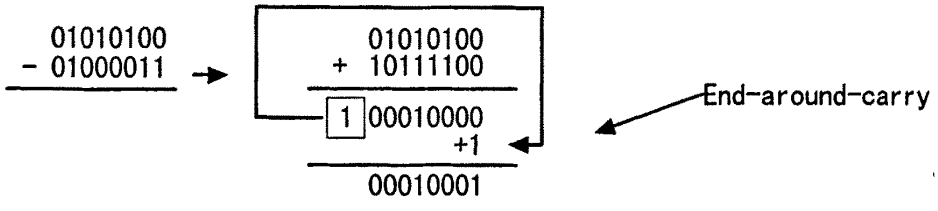

图 4.30 a 1'complement 补码形式减法

结果为：+(00010001)

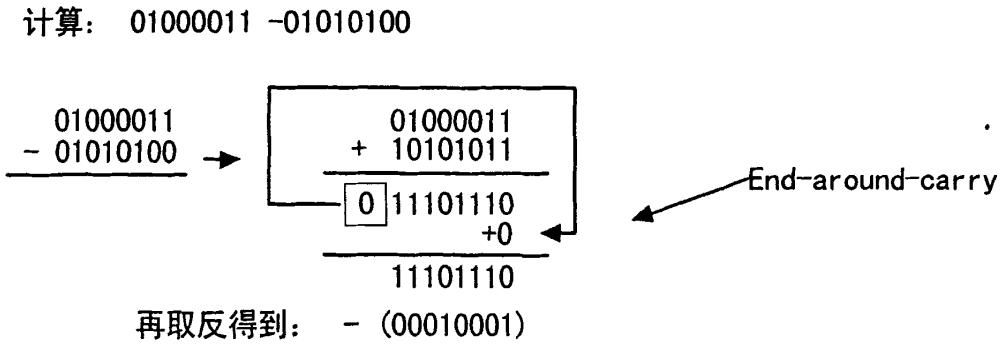

图 4.31　b 1'complement 补码形式减法

结果为：-(00010001)

实现这种运算的加法器结构称为 End-around-adder。即在正常加法完成后将进位为再次加到结果最低位上。

它的实现方法可以通过下图 4.32 来表示。

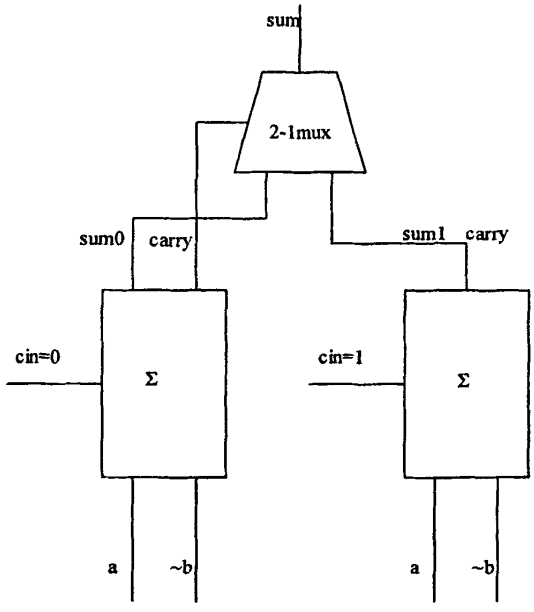

图 4.32　End-around-adder 示意图

如图 4.32 所示，采用两个加法器，一个的进位为 0，一个进位为 1，通过进位为 0 的加法器的进位来选择究竟选取是进位为 0 所算出的结果，还是进位为 1 所算出的结果。而进位为 0 的加法器跟进位为 1 的加法器是同时进行的。它们采用 12 位的进位级联加法器（RCA）。(因为位数不多，采用简单的级连加法器即

可)

因为在上面所描述的 End around adder 中，我们主要描述的减法所转换成的运算，在浮点数运算中，我们只对尾数进行运算，而尾数全是正数表示，因此最前一位经符号扩展是都是 0，当相减时，其中减数则被取反，因此最高位变成了 1，当被减数与减数取反后的值相加时，如果进位为 0，则说明他们相减后变成了负数，那么我们还是取本身算出的结果，不过最后理解时，我们知道这是一个负数，它的绝对值只要将 SUM 取反即可。

而当进位为 1 时，则说明它们相减后还是正数，如果要得到真正的结果，只要在 SUM 的最低位上加上 1。

这样的话，我们则利用了一个进位选择加法器的原理，因为进位为 0 跟 1 的结果可以并行算出，从而大大提高了速度。

4.2.12 符号预测

乘累加运算中指令（我们这里只列出了单精度的指令，双精度的跟单精度一样编码。有：

MACS Fd := Fd + (Fn*Fm),

NMSCS Fd := - Fd -(Fn*Fm),

MSCS Fd := -Fd +(Fn*Fm),

NMACS Fd := Fd - (Fn*Fm),

四个不同的乘加组合，但实际上由于三个操作数的符号可正可负，而尾数只能是正数，四个指令会显得比较复杂。这里我们只是将其作个预测，分成同号运算，同号即代表相加，同号运算的情况下可确定符号，这里分别用 00，01 来代表同号运算，同正还有同负。另外为异号运算，即相减。这里分别用 10，11 来表示是 Fm*Fn-Fd,跟 Fd-Fm*Fn。

表 4.5 符号预测结果含义

sign_det	含义
00	同号操作，结果符号为+
01	同号操作，结果符号为-
10	异号操作，结果为 fm*fn-fd
11	异号操作，结果为 fd -fm*fn

每个指令因为三个数的符号不同，可能出现 8 种情况，而 4 条指令则共有 32 种情况。表 4.6 逐一列出。

表 4.6 符号预测情况列表

指令	操作数 fd	操作数 fm	操作数 fn	操作类型	结果信息	sign_det
MAC	+	+	+	同号	正数	00
MAC	+	+	-	异号	fm*fn-fd	10
MAC	+	-	+	异号	fd-fm*fn	11
MAC	+	-	-	同号	正数	00
MAC	-	+	+	异号	fm*fn-fd	10
MAC	-	+	-	同号	负数	01
MAC	-	-	+	同号	负数	01
MAC	-	-	-	异号	fm*fn-fd	10
MSC	+	+	+	异号	fm*fn-fd	10
MSC	+	+	-	同号	正数	00
MSC	+	-	+	同号	正数	00
MSC	+	-	-	异号	fm*fn-fd	10
MSC	-	+	+	同号	负数	01
MSC	-	+	-	异号	fd-fm*fn	11
MSC	-	-	+	异号	fd-fm*fn	11
MSC	-	-	-	同号	负数	01
NMSC	+	+	+	同号	负数	01
NMSC	+	+	-	异号	fm*fn-fd	10
NMSC	+	-	+	异号	fm*fn-fd	10
NMSC	+	-	-	同号	负数	01
NMSC	-	+	+	异号	fd-fm*fn	11
NMSC	-	+	-	同号	正数	00
NMSC	-	-	+	同号	正数	00
NMSC	-	-	-	异号	fd-fm*fn	11
NMAC	+	+	+	异号	fd-fm*fn	11

NMAC	+	+	-	同号	正数	00
NMAC	+	-	+	同号	正数	00
NMAC	+	-	-	异号	fd-fm*fn	11
NMAC	-	+	+	同号	负数	11
NMAC	-	+	-	异号	fm*fn-fd	10
NMAC	-	-	+	异号	fm*fn-fd	10
NMAC	-	-	-	同号	负数	01

符号预测的结果将被送进 E3 级的多路器来控制选择操作数。

4.2.13 默认结果选择

该单元在 E7 级结果选择模块。根据异常类型选择标准的 IEEE754 输出结果。

表 4.7 IEEE754 定义默认结果

异常类型	默认正结果	默认负结果
无效操作	QNaN	QNaN
除零	正无穷	负无穷
上溢	舍入到最近，舍入到正无穷： 正无穷 舍入到零，舍入到负无穷： 正的最大值	舍入到最近，舍入到负无穷： 负无穷 舍入到零，舍入到正无穷： 负的最大值
下溢	正常舍入结果	正常舍入结果
不精确	正常舍入结果	正常舍入结果

所有浮点转整数指令的异常当作无效操作类型处理，整数转浮点数不会产生异常，首先利用浮点异常探测单元对浮点数转整数的异常探测确定异常情况，然后按照表 4.8 将默认结果写回。

表 4.8 转换类指令默认结果

	FTOUIS/FTOUID		FTOSIS/FTOSID	
舍入后的输入	写回结果	IOC 置位	写回结果	IOC 置位
$x>2^{32}$ （上溢）	0xFFFFFFFF	是	0x7FFFFFFF	是
$2^{31} \leqslant x < 2^{32}$	正常整数	否	0x7FFFFFFF （上溢）	是

$0 \leqslant x < 2^{31}$	正常整数	否	正常整数	否
$0 \geqslant x \geqslant -2^{31}$	0x00000000 （下溢）	是	正常整数	否
$x < -2^{31}$ （下溢）	0x00000000	是	0x80000000	是
NaN	0x00000000	是	0x00000000	是
$+\infty$	0xFFFFFFFF	是	0x7FFFFFFF	是
$-\infty$	0x00000000	是	0x80000000	是

第五章 除法开方流水线设计

本章介绍了除法开方流水线的设计。首先介绍合并的除法开方迭代算法。进一步给出除法开方迭代单元的实现方法。包括试根查找表的实现，迭代函数的实现，以及迭代控制器的设计。

5.1 概述

除法和开方流水线通过迭代完成单精度/双精度除法和开方操作。向量运算的最终结果等同于顺序完成的一系列等价操作，其流水线示意图如图 5.1 所示。

除法开方指令发射后，根据操作数单精度设定控制单元的迭代次数计数器的结束值。在执行一级，操作数送往乘加流水线进行异常预判断。迭代过程中，迭代次数计数器每次加 1，迭代过程不需要外界参与。迭代结束后，整理单元根据舍入模式对结果舍入。若发生异常未使能的异常时，结果选择单元根据当前所处模式选择系统默认值。

除法运算的过程是在发射级，操作数 Fm、Fn 作为被除数和除数进入流水线。在 E1 级通过多路器进行迭代选择。在 E2 级计算除法的部分项，并通过特定算法的选择器进行下一迭代的生成。在 E3 级对累加结果按照特定的模式和目标精度进行舍入，并检查异常、完成规格化。在 E4 级进行最后结果的选择。在写回级，最终结果写入 Fd 寄存器。

开方运算的过程是在发射级，操作数 Fm 作为被开方数进入流水线。在 E1 级通过多路器进行迭代选择。在 E2 级计算开方的部分项，并通过特定算法的选择器进行下一迭代的生成。在 E3 级对累加结果按照特定的模式和目标精度进行舍入，并检查异常、完成规格化。在 E4 级进行最后结果的选择。在写回级，最终结果写入 Fd 寄存器。

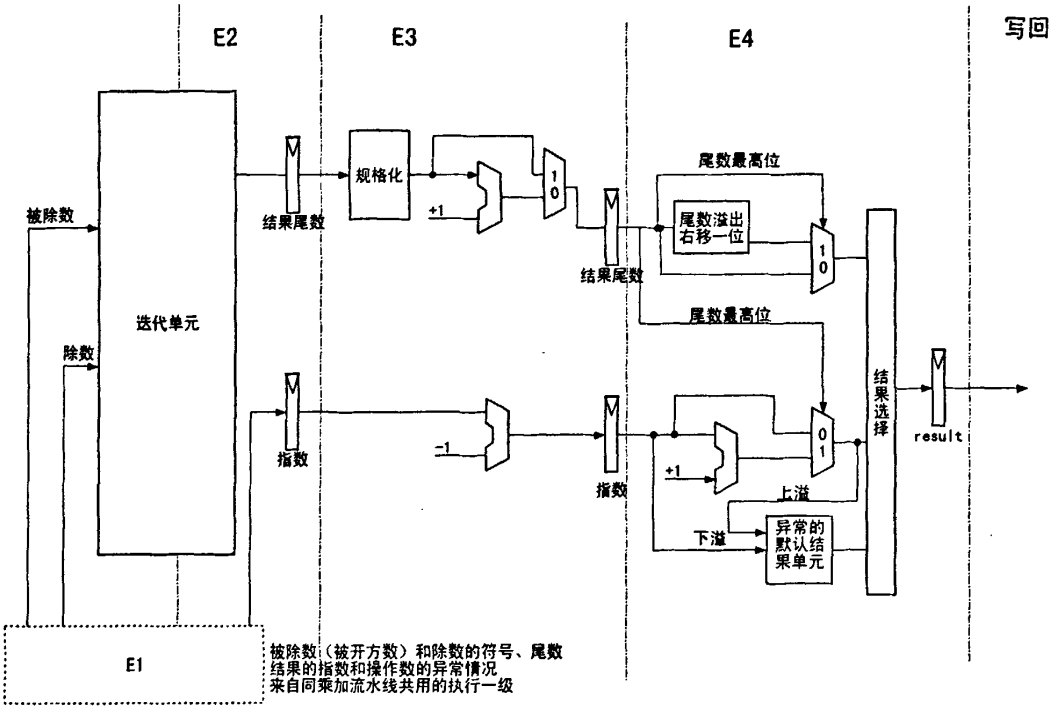

图 5.1 除法开方流水线结构图

5.2 SRT 算法介绍

此模块采用改进的 SRT 除法开方运算算法，SRT 算法[38][39]能够很好地把除法和开方运算很好地综合起来，大幅度地减少芯片的面积， SRT 算法的具体的迭代公式：

$$R_{i+1} = r * R_i - D * q_{i+1} \tag{1}$$

$$R_{i+1} = r * R_i - Q_i q_{i+1} - \frac{1}{2} \frac{q_{i+1}^2}{r^{i+1}} \tag{2}$$

其中（1）式用于除法（2）式用于开方。R_i 代表第 i 次迭代后的余数，r 代表 SRT 算法的基，D 代表除数， Q_i 代表 i 次迭代后的商或根。初始条件 Q_0=1.0 。

$$Q_i = \sum_{j=0}^{i} q_j r^{-j} \tag{3}$$

q_i 代表商或根 Q_i 的第 i 位。我们令上面公式（1）（2）中的 r 等于 4 形成基

为 4 的 SRT 算法，好处是迭代次数变为基为 2 的迭代算法的 1/2。例如 54 位数字相除只需迭代 28 次。基为 4 的 SRT 算法商的每一位将满足 $q_i \in \{-2,-1,0,1,2\}$，以冗余的形式表示，一次迭代产生一位 qi。第 i 次迭代得到的部分商满足 $Q_i = \sum_{j=0}^{i} q_j 4^{-j}$。将冗余形式表示的 Q_i 转化为二进制表示的就是计算结果。将除法和开方运算集合起来可以形成如下公式：

$$R_{i+1} = 4R_i + F_i \qquad i=0, 2, 4, 6 \ldots$$

其中

$$F_i = \begin{cases} -q_{i+1}D & \text{（除法）} \\ -(Q_i q_{i+1} + \frac{1}{2}4^{-(i+1)} q_{i+1}^2) & \text{（开方）} \end{cases}$$

$$R_0 = \begin{cases} X - D & \text{（除法）} \\ \frac{1}{2}(X-1) & \text{（开方）} \end{cases}$$

其中，X 是被除数/被开方数

$$q_{i+1} = \begin{cases} Selcet(\hat{y}, \hat{D}) & \text{（除法）} \\ Select(\hat{y}, \hat{Q_i}) & \text{（开方）} \end{cases}$$

对于除法，q_i 是第 i 次迭代的商 Q_i 的第 i 位。q_{i+1} 由选择函数 $q_{i+1}=select(\hat{y},\hat{D})$ 获得。其中 $\hat{y}$ $\hat{D}$ 是通过截取 4 倍的部分余数 $4R_i$ 和除数 D 若干高位数字得到的。

对于开方，q_i 是第 i 次迭代的平方根结果 Q_i 的第 i 位。q_{i+1} 由选择函数 $q_{i+1}=select(\hat{y},\hat{Q_i})$ 获得。其中 $\hat{y}$ $\hat{Q_i}$ 是通过截取 4 倍的部分余数 $4R_i$ 和部分根 Q_i 若干高位数字得到的。

在 SRT 算法里除数设定范围为 $0.5 \le D < 1$，根设定满足 $0.5 \le Q_i < 1$，从上面公式可得出余数满足 $0.25 \le R_i < 1$。这样我们通过对已经规格化的满足 IEEE754 的操作数的尾数稍加变换就能使之参与运算。

同时余数满足 $0.25 \le R_i < 1$ 可以保证被开方数的指数总是偶数。因为按照 IEEE754 规格化的开方操作数 $1.S \times 2^e$ 其尾数总是满足 $1 \le 1.S < 2$，通过，得到(0.25，1)范围内的被开方数。表 5.1 为预变换说明列表。

表 5.1　开方运算操作数的预变换

指数奇偶性	相应变换操作	变换后指数奇偶性
e 为奇数	尾数右移 1 位指数加 1 变换	偶数
e 为偶数	尾数右移 2 位指数加 2 变换	奇数

除法开方运算每一次迭代涉及运算：

(1)把 R_i 左移两位，产生 $4R_i$

(2)使用 $q_{i+1} = select(\hat{y}, \hat{D})$ 确定 q_{i+1}

(3)计算 F_i

(4)把 F_i 和 $4R_i$ 相加产生 R_{i+1}

(5)把 q_{i+1} 拼接在 Q_i 后面形成 Q_{i+1}

迭代单元结构示意图如图 5.2 所示。

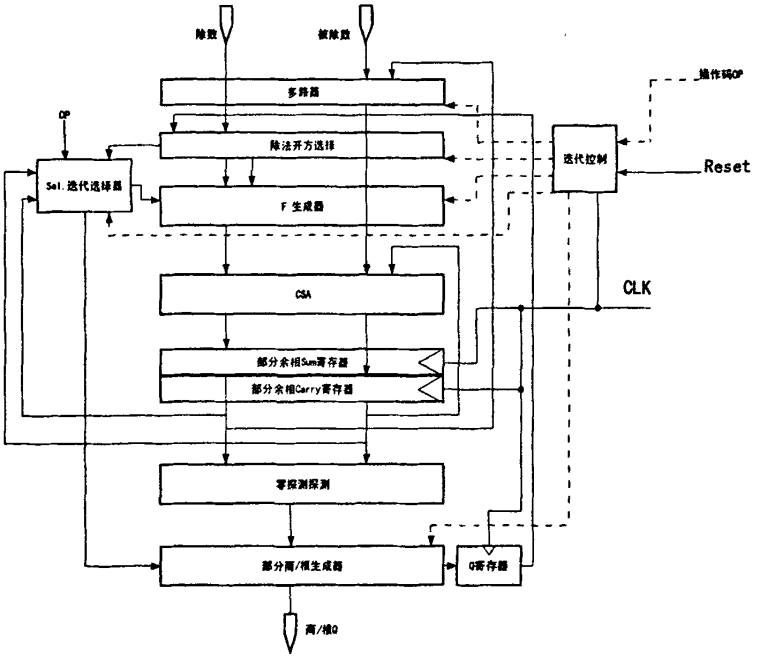

图 5.2　迭代单元结构示意图

5.3 查找表 Select 设计

SRT 算法采用查找表的方法得到每次部分商/平方根的最高位。Select()单元主要包括两部分：加法器和选择逻辑。

加法器是一个 8 位 CPA 加法器，它将每次迭代形成的用进位保留表示的 $4R_i$ 的和 sum 和进位 carry 高 8 位相加，提供选择逻辑使用。表示为 $\hat{y}$。

选择逻辑为查找表形式，有两个输入项

$$q_{i+1} = \begin{cases} Selcet(\hat{y},\hat{D}) & \text{（除法）} \\ Select(\hat{y},\hat{Q}_i) & \text{（开方）} \end{cases}$$

对于除法，其中 $\hat{D}$ 是除数 D 的权重为 2^{-2}，2^{-3}，2^{-4} 位

对于开方，其中 $\hat{Q}_i$ 选择如下：

表 5.2 $\hat{Q}_i$ 选择表

迭代	$\hat{Q}_i$ [2]	$\hat{Q}_i$ [1]	$\hat{Q}_i$ [0]
第一次迭代（i=0）	1	0	1
若 $\hat{Q}_i$ 权重为 2^{-0} 位等于 1 并且 i>0	1	1	1
若 $\hat{Q}_i$ 权重为 2^{-0} 位等于 0 并且 i>0	权重为 2^{-2} 位	权重为 2^{-3} 位	权重为 2^{-4} 位

选择结果 $q_i \in \{-2,-1,0,1,2\}$，使用 4 位编码形式简化 F_i 生成器逻辑门设计。

表 5.3 q_i 编码表

2				
1				

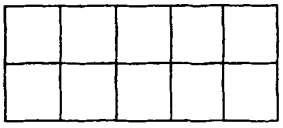

表 5.4 Select 查找表

$\widehat{D}$或$\widehat{Q_i}$ / q_{i+1}	0.1000	0.1001	0.1010	0.1011	0.1100	0.1101	0.1110	0.1111
-2	<1110.0110	<1110.0100	<1110.0000	<1101.1110	<1101.1100	<1101.1000	<1101.0100	<1101.0010
-1	1110.0110~ 1111.1000	1110.0100~ 1111.0110	1110.0000~ 1111.0100	1101.1110~ 1111.0100	1101.1100~ 1111.0100	1101.1000~ 1111.0000	1101.0100~ 1111.0000	1101.0010~ 1111.0000
0	1111.1000~ 0000.1000	1111.0110~ 0000.1000	1111.0100~ 0000.1000	1111.0100~ 0000.1000	1111.0100~ 0000.1100	1111.0000~ 0000.1100	1111.0000~ 0001.0000	1111.0000~ 0001.0000
1	0000.1000~ 0001.1000	0000.1000~ 0001.1100	0000.1000~ 0010.0000	0000.1000~ 0010.0000	0000.1100~ 0010.0100	0000.1100~ 0010.1000	0001.0000~ 0010.1000	0001.0000~ 0010.1100
2	>0001.1000	>0001.1100	>0010.0000	>0010.0000	>0010.0100	>0010.1000	>0010.1000	>0010.1100

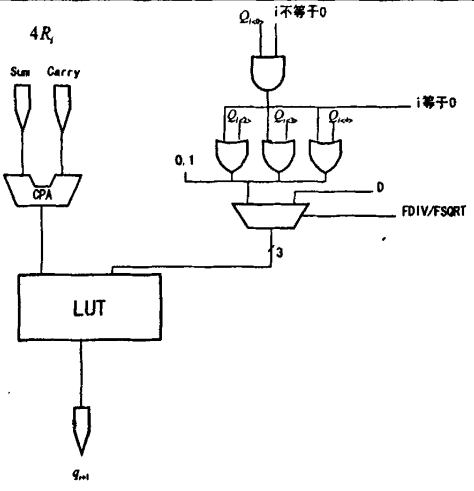

图 5.3 Select 单元结构示意图

查找表对应的数值是确定的，行数据位数宽度为 3，从 000－111；列数据位数宽度为 8，从 00000000－11111111。把整个查找表对应为真值表，把输入的每一位定义成为 PLA 的输入，divquot_in[2:0]定义为 A,B,C；partial_in[7:0]定义为 P7-P0；sel_out[3:0]定义为 S3,S2,S1,S0；用“与”逻辑和“或”逻辑写出输入和输出之间的对应关系。

这样用硬连线就能借助组合逻辑实现查找表的功能，因为组合逻辑电路可以为流水线节省时间，但是这样将会产生 2048 个乘积项，对于很多变量的逻辑表

达式我们用 Quine-McCluskey 的化简方法来对逻辑表达式化简。其基本的原理表达式为：

$$\bar{A}\bar{B}CD\bar{E}+\bar{A}\bar{B}CD\bar{E}=\bar{A}\bar{B}C\bar{E}(D+D)=\bar{A}\bar{B}C\bar{E}$$

采用手工运算得到最后的化简结果为：

$$S1=\bar{P}10\bar{P}7\bar{P}6\bar{P}5\bar{P}4P3+\bar{P}10\bar{P}9\bar{P}7\bar{P}6\bar{P}5P4\bar{P}3+\bar{P}10\bar{P}9P8\bar{P}7\bar{P}6\bar{P}5P4P3\bar{P}2+$$
$$P10\bar{P}9\bar{P}7\bar{P}6\bar{P}5\bar{P}4P3P2+P10\bar{P}9P8\bar{P}7\bar{P}6P5\bar{P}4\bar{P}3\bar{P}2+P10P9\bar{P}7\bar{P}6P5\bar{P}4\bar{P}3+$$
$$P10P9P8\bar{P}7\bar{P}6\bar{P}5P4\bar{P}3P2+P10\bar{P}7\bar{P}6\bar{P}5P4+P10\bar{P}9\bar{P}7\bar{P}6\bar{P}5P4$$

$$S2=\bar{P}10\bar{P}9\bar{P}8\bar{P}7\bar{P}6\bar{P}5P4P3+P10\bar{P}9\bar{P}8\bar{P}7\bar{P}6P5\bar{P}4\bar{P}3P2+$$
$$P10\bar{P}9P8\bar{P}7\bar{P}6P5P4P3P2+P10\bar{P}9P8\bar{P}7\bar{P}6P5\bar{P}4P3+$$
$$P10\bar{P}9P8P7\bar{P}6P5P4P3+P10P9P8P7\bar{P}6P5P4P3+$$
$$P10\bar{P}7\bar{P}6P5+P10P9P8\bar{P}7\bar{P}6P5P4P3P2+$$
$$\bar{P}7\bar{P}6P5+P10\bar{P}7\bar{P}6P5P4+$$
$$P10P9P8P7\bar{P}6P5P4+P10P9P7\bar{P}6P5P4+$$
$$\bar{P}7P6P5P4+P7P6P5P4P3$$
$$P7P6P5P4P3\bar{P}2+P7P6P5P4P3P2P1$$
$$+P10\bar{P}9\bar{P}8\bar{P}7\bar{P}6\bar{P}5\bar{P}4$$

$$S-2=P7\bar{P}6\bar{P}5\bar{P}4\bar{P}3\bar{P}2\bar{P}1P0+P7\bar{P}6\bar{P}5\bar{P}4\bar{P}3\bar{P}2P1+P7\bar{P}6\bar{P}5\bar{P}4P3\bar{P}2+$$
$$P7\bar{P}6\bar{P}5\bar{P}4P3+P7\bar{P}6\bar{P}5+P10\bar{P}9\bar{P}7\bar{P}6\bar{P}5+P10\bar{P}9\bar{P}8\bar{P}7\bar{P}6\bar{P}5+$$
$$P10\bar{P}7\bar{P}6\bar{P}5P4+P10P9P8P7\bar{P}6P5P4+P10P9P7\bar{P}6P5P4P3P2+$$
$$P10P9P8P7\bar{P}6P5P4P3P2P1+P10P9P8P7\bar{P}6P5P4P3P2P1+$$
$$P10P9P8P7\bar{P}6P5P4P3P2+P10P9P7\bar{P}6P5P4P3+$$
$$P10P9P8P7\bar{P}6P5P4P3+P10P8P7\bar{P}6P5P4P3P2+P10P9P8P7\bar{P}6P5P4P3P2P1$$

$$S-1=P7\bar{P}6P5\bar{P}4+\bar{P}10\bar{P}9P7\bar{P}6P5\bar{P}4P3+\bar{P}10\bar{P}9\bar{P}8P7\bar{P}6P5\bar{P}4P3P2P1+$$
$$\bar{P}10\bar{P}9\bar{P}8P7\bar{P}6P5P4\bar{P}3+\bar{P}10\bar{P}9P8P7\bar{P}6P5\bar{P}4\bar{P}3P2+\bar{P}10\bar{P}9P8P7\bar{P}6P5P4\bar{P}3\bar{P}2+$$
$$\bar{P}10P9P8P7\bar{P}6P5\bar{P}4P3P2P1+P10\bar{P}9\bar{P}8P7\bar{P}6P5P4P3P2+P10\bar{P}9\bar{P}8P7\bar{P}6P5P4\bar{P}3\bar{P}2+$$
$$P10P9P7\bar{P}6P5\bar{P}4\bar{P}3P2+P10P9P7\bar{P}6P5\bar{P}4P3+P10P9P8P7\bar{P}6P5\bar{P}4\bar{P}3\bar{P}2P1$$

上述四种情况对应的取值都为 0 时，也就是 S=0 的逻辑。所以通过上述简化则使组合逻辑实现简单得多，每一个表达式需要的主项不大于 20 个，所以整个的列选择数即主项的总数不大于 80 个，而不进行化简的项数则为 2048 个，采用 Quine-McCluskey 法化简逻辑表达式能够用很少的逻辑门电路来实现查找表单元。

5.4 F 生成器

此模块生成 SRT 算法的 F 值，把除法和开方运算合并起来。

$$F_i = \begin{cases} -q_{i+1}D & \text{（除法）} \\ -(Q_i q_{i+1} + \frac{1}{2}4^{-(i+1)} q_{i+1}^2) & \text{（开方）} \end{cases}$$

由于 $q_{i+1} \in \{-2,-1,0,1,2\}$，该单元有输入项：$q_{i+1}$[3:0], D[54:0], Q_i[54:0]和当前迭代次数 i，输出项：F_i[54:0]该单元主要由移位器、反相器和加法器组成：

表 5.5 q_{i+1} 对应的 F 操作

q_{i+1}	F_i（除法）	F_i（开方）
-2	D 取反加 1，左移 1 位	Q_i 取反加 1，左移 1 位；加上 1 右移（2i+1）位
-1	D 取反加 1	Q_i 取反加 1；加上 1 右移（2i+3）位
0	0	0
1	D	Q_i 加上 1 右移（2i+3）位
2	D 左移 1 位	Q_i 左移 1 位；加上 1 右移（2i+1）位

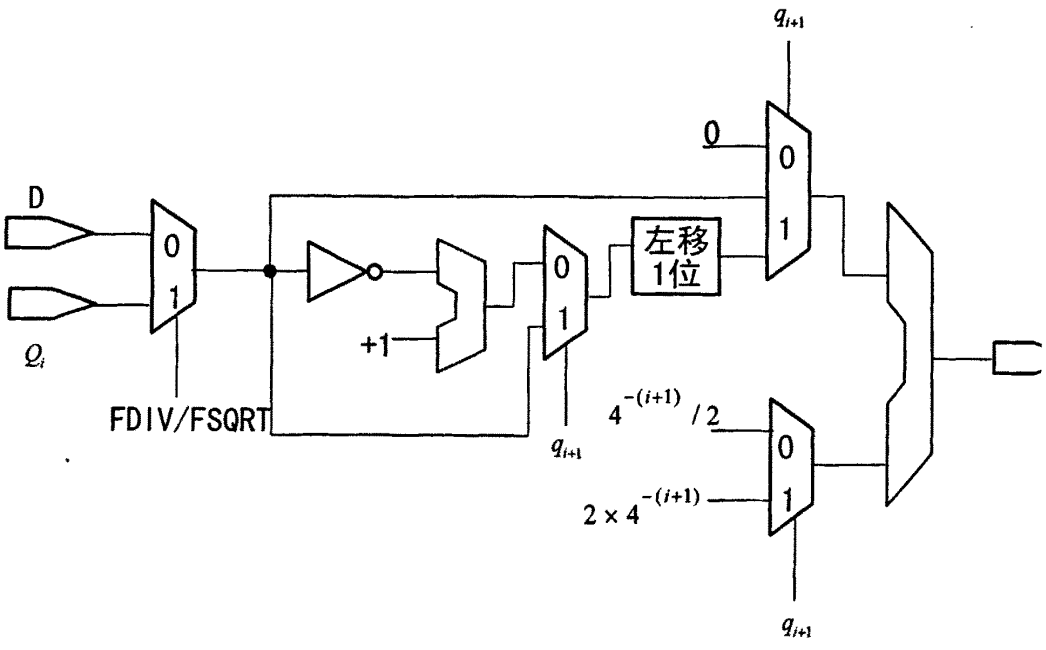

图 5.4 F 生成器结构示意图

伪代码描述为：

```
if(ds==1'b1)                    //div
  case(s)
  4'b0000: f=62'b0;             // 0
  4'b0001: f=(~d+1'b1);     // f=-d
  4'b0010: f=((~d)+1'b1)<<1;   // f=-2d
  4'b0100: f=d<<1;      // f=2d
  4'b1000: f=d;         // f=d
  default: f=62'b0;     //default
  endcase
else
  case(s)                       //sqrt
  4'b0000: f=62'b0;     // 0
  4'b0001: f=(~q+1'b1)+~((3'b001,59'b0)>>(2*cnt+3))+1'b1;
                                        // f=-q-1/2*4exp[-(i+1)]
  4'b0010: f=((~q+1'b1)<<1)+~((3'b001,59'b0)>>(2*cnt+1))+1'b1;
                                        // f=-4q-1/2*4exp[-i]
  4'b0100: f=(q<<1)+~((3'b001,59'b0)>>(2*cnt+1))+1'b1;
                        // f=4q-1/2*4exp[-i]
  4'b1000: f=q+~((3'b001,59'b0)>>(2*cnt+3))+1'b1;
                                        // f=q-1/2*4exp[-(i+1)]
  default: f=62'b0;       //default
```

部分商生成模块

本模块完成此次迭代运算的部分商/平方根的输出，返回给迭代回路。该单元迭代的部分商的生成公式为：

$$Q_{i+1} = Q_i + q_{i+1}4^{-(i+1)}$$

该单元有输入项：q_{i+1}[3:0]，Q_i[54:0]，输出项Q_{i+1}[54:0]。该单元主要由移位器、反向器和加法器组成：

表 5.6 q_i对应商的操作

	Q_{i+1}
2	Q_i加上-1 右移(2i+1)位
1	Q_i加上-1 右移(2i+2)位
	0
	Q_i加上 1 右移(2i+2)位
	Q_i加上 1 右移(2i+1)位

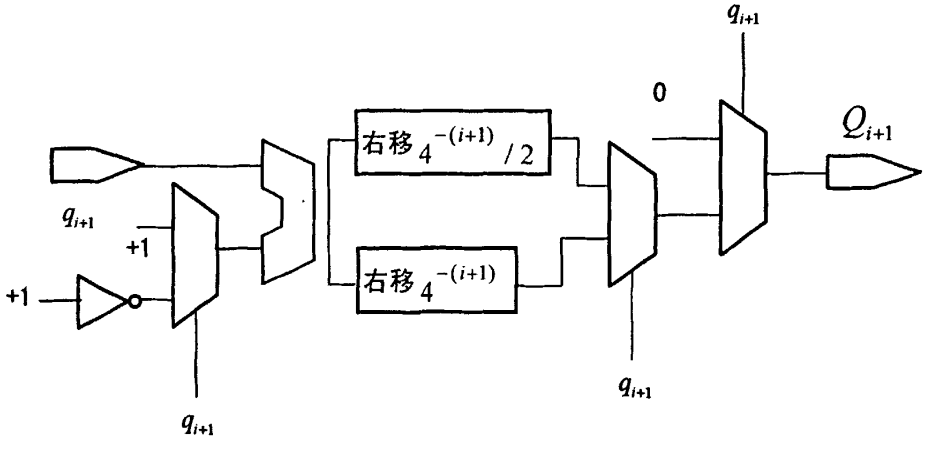

图 5.5 Q_{i+1}生成器结构示意图

伪代码描述为：

```
case(s)
4'b0000:    qout = qin;
4'b0001:    qout = qin+({3'b0,2'b01,57'b0} >>(2*cnt));          // q[i+1]=+1
4'b0010:    qout = qin+({3'b0,2'b10,57'b0} >>(2*cnt));          // q[i+1]=+2
4'b0100:    qout = qin+~({3'b0,2'b10,57'b0} >>(2*cnt))+1'b1;    // q[j+1]=-2
4'b1000:    qout = qin+~({3'b0,2'b01,57'b0} >>(2'cnt))+1'b1;    // q[j+1]=-1
default:    qout=qin;
//default
endcase
//quotient generation
```

5.5 迭代控制模块

本模块产生载入信号，完成信号和调整信号，并记录此次迭代运算的次数 count。图 5.6 为控制流程图。

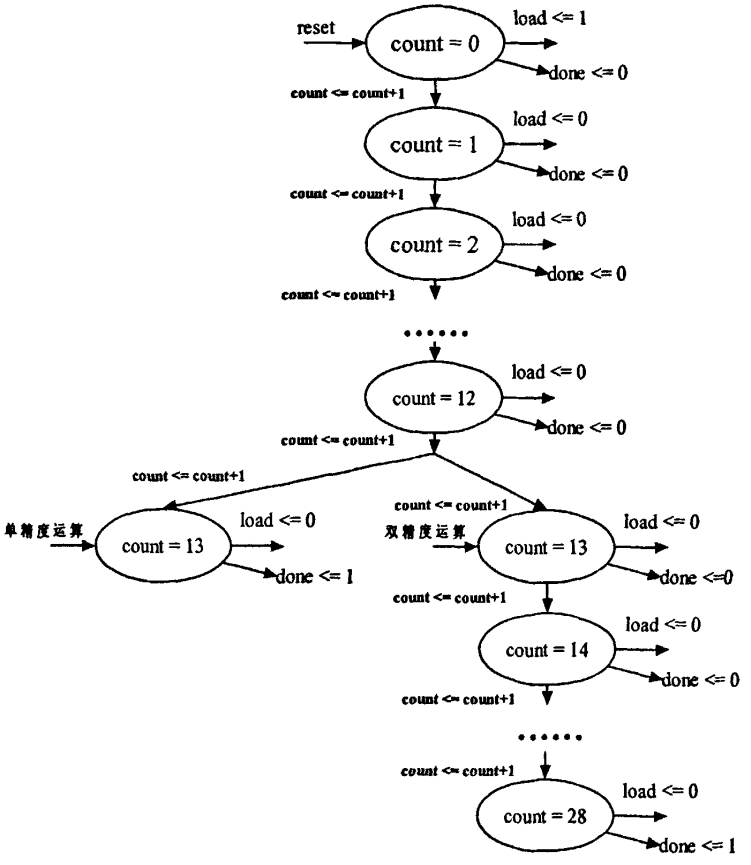

图 5.6 迭代单元控制示意图

迭代控制包含迭代次数计数器：count。如图 5.8 所示，开始迭代时，计数器复位，count = 0，也即迭代公式中的 i = 0，置 load 为 1,控制迭代初值载入。done 置零，表明迭代未结束。

进入下一次计数，load 置低电平，done 保持。

count<=count+1 表示将计数器值增 1。

当操作是单精度操作时，计数器计数到 13 就停止。done 置高电平。迭代结束。

当操作是双精度操作时，计数器继续计数到 28 停止。done 置高电平。迭代结束。

伪代码如下：

```
if(!clear)
        begin
        load<=1'b1;                    //给出迭代赋初值信号
```

```
        count<=0;
        done<=1'b0;
        adj<=1'b0;                          //余数为负的结果调整，
                                            //在有保护位的情况下可以不实现。
        end
    else if(count<(bit_to_calc-1'b1))       // bit_to_calc 是要迭代的次数
                                            //双精度 bit_to_calc=28
                                            //单精度 bit_to_calc=13；
        begin
        load<=1'b0;
        count<=count<+1'b1;                 //count 增 1
        done<=1'b0;
        adj<=1'b0;
        end
    else if(count<==(bit_to_calc-1'b1))
        begin
        load<=1'b0;
        done<=1'b1;                         //给出迭代结束信号
        count<=count<+1'b1;
        if(w[61]==1'b1)                     //给出调整信号
            adj<=1'b1;
        else
            adj<=1'b0;
        end
    else if(count<==bit_to_calc)
        begin
        load<=1'b0;
        done<=1'b1;
        count<=count<+1'b1;
        adj<=adj;
        end
    else
        begin
        load<=1'b0;
        done<=1'b1;
        count<=count<;
        adj<=adj;
        end
```

第六章 异常处理机制

本章介绍了协处理器兼容 IEEE 754 标准的的浮点运算异常处理机制。

6.1 概述

浮点算术运算在运行过程中可能会产生浮点异常，IEEE754 对这些异常进行了分类，规定了异常的处理方法通过以下两种方式进行。

1. 不调用浮点异常处理程序处理

未定义指令异常发生后，由硬件将入 IEEE754 标准定义的默认结果写回。置系统寄存器相应的标志位为 1，该异常指令的所有涉及到的结果寄存器写入 IEEE754 定义的默认结果。程序继续执行。

表 6.1 IEEE754 定义默认结果

异常类型	默认正结果	默认负结果
无效操作	QNaN	QNaN
除零	正无穷	负无穷
上溢	舍入到最近，舍入到正无穷：正无穷 舍入到零，舍入到负无穷：正的最大值	舍入到最近，舍入到负无穷：负无穷 舍入到零，舍入到正无穷：负的最大值
下溢	正常舍入结果	正常舍入结果
不精确	正常舍入结果	正常舍入结果

所有浮点转整数指令的异常当作无效操作类型处理，整数转浮点数不会产生异常，首先利用浮点异常探测单元对浮点数转整数的异常探测确定异常情况，然后按照下表将默认结果写回。

表 6.2 浮点转整数异常默认结果列表

	FTOUIS/FTOUID		FTOSIS/FTOSID	
舍入后的输入	写回结果	IOC 置位	写回结果	IOC 置位
$x>2^{32}$	0xFFFFFFFF	是	0x7FFFFFFF	是
$2^{31} \leqslant x < 2^{32}$	正常整数	否	0x7FFFFFFF	是

$0 \leqslant x < 2^{31}$	正常整数	否	正常整数	否
$0 \geqslant x \geqslant -2^{31}$	0x00000000	是	正常整数	否
$x < -2^{31}$	0x00000000	是	0x80000000	是
NaN	0x00000000	是	0x00000000	是
$+\infty$	0xFFFFFFFF	是	0x7FFFFFFF	是
$-\infty$	0x00000000	是	0x80000000	是

2. 中断调用浮点异常处理程序处理

通过对系统寄存器特定位的置位来激活模式。异常发生后，通过中断反馈给主处理器，主处理器调用浮点异常处理程序处理异常，结束后返回原程序继续执行后续指令。

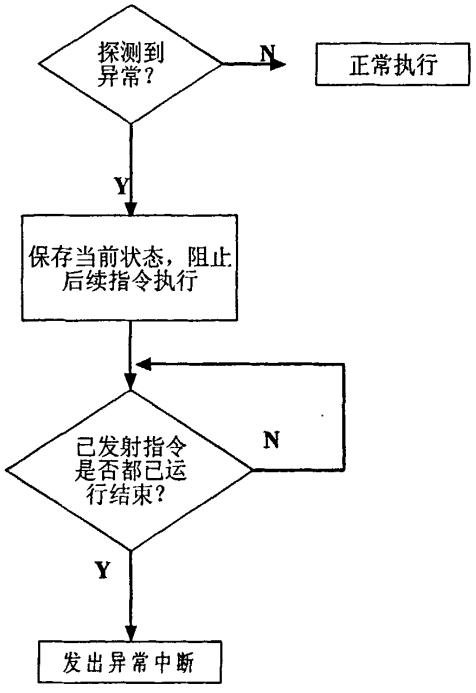

图 6.1 调用浮点异常处理程序的异常处理流程图

6.2 异常探测单元

我们在设计中使用异常探测单元支持第二种异常处理方式，即第三章 3.4 节给指的非精确异常处理方法。异常探测单元由乘累加流水线和除法开方流水线共用，位于乘累加流水线执行级第 1 级。它根据操作码和操作数探测无效操作和除

零操作，下上溢和下溢探测。异常探测逻辑基于查找表结构。接受输入操作数的指数、符号和尾数，以及当前指令。由这些信息查找异常是否产生，以及异常产生的类型。该查找表主要完成以下判断。

1. 无效操作和除零操作的判断

无效操作：结果不能被表示，或结果未定义。它包含：

除 FCPY，FNEG，FABS 之外的 CDP 指令在操作数含 SNAN 时就产生此异常

FADD: $(+\infty)+(-\infty)$ or $(-\infty)+(+\infty)$

FSUB: $(+\infty)-(+\infty)$ or $(-\infty)-(-\infty)$

FCMPE/FCMPEZ: 含有 NaN operand

FMUL/FNMUL: $0\times \pm\infty$ or $\pm\infty\times 0$

FDIV: 0/0or ∞/∞.

FMAC/FNMAC: 同 FADD、FMUL

FMSC/FNMSC：同 FSUB、FMUL

FSQRT：操作数为负数

FTOUI：结果超出（0，232）

FTOSI：结果超出（-231，231）

FTOUI 操作数为负数

含有值为∞的 A 操作数和潜在的结果溢出的 FMAC 类

2. 除零操作。

包括正常数除以 0 和过小数除以 0。在 round to zero 模式探测除零异常时过小数看作+0，结果根据无效操作是否使能判断。

3. 上溢出和下溢出

上溢出和下溢出异常是根据操作数的指数推测出来的。它探测的是潜在可能的异常。最终是否发生异常需要由中断处理程序判断。

6.3 部分异常探测查找表内容

6.3.1 浮点加法和减法指令溢出探测查找表

在本协处理器中减法转化为加法运算。异常和最初结果的指数即操作数最大的指数有关。首先需要根据指令和操作数符号判断两操作数绝对值参与的真实运算类型。操作类型如下表 6.3 所示。发生异常中断的指数阈值如表 6.4 所示。

表 6.3 ADD 和 SUB 操作类型列表

指令	操作数 fd	操作数 fm	操作类型
FADD	+	+	同号
FADD	+	-	异号
FADD	-	+	异号
FADD	-	-	同号
FSUB	+	+	异号
FSUB	+	-	同号
FSUB	-	+	同号
FSUB	-	-	异号

表 6.4 加法类异常中断阈值

最初结果的指数			状态	
双精度（DP）	单精度（SP）	浮点值	单精度（SP）	双精度（DP）
>0x7FF	-	DP overflow	-	中断
0x7FF	-	DP NaN or 无穷	-	中断
0x7FE	-	DP overflow	-	中断
0x7FD	-	DP overflow	-	中断
0x7FC	-	DP 正常	-	正常
>0x47F	>0xFF	SP overflow	中断	正常
0x47F	0xFF	SP NaN or 无穷	中断	正常
0x47E	0xFE	SP overflow	中断	正常
0x47D	0xFD	SP overflow	中断	正常
0x47C	0xFC	SP 正常	正常	正常
0x3FF	0x7F	e = 0 偏移值	正常	正常
0x3A0	0x20	SP 正常	最小值(异号)	正常
0x39F	0x1F	SP underflow	中断（异号）or 正常（同号）	正常
0x381	0x01	SP 正常	最小值(同号)	正常

0x380	0x00	SP sub 正常	中断	正常
<0x380	<0x00	SP underflow	中断	正常
0x040	-	DP 正常	-	正常（同号） or 最小值（异号）
0x03F	-	DP underflow（异号）	-	正常 （同号）or 中断（异号）
0x001	-	DP 正常（同号）	-	最小值（同号）or 中断（异号）
0x000	-	DP 低于正常	-	中断
<0x000	-	DP underflow	-	中断

6.3.2 浮点乘法类指令溢出探测查找表

浮点乘法溢出异常判断基于被乘数和乘数指数相加得到的乘积项指数。尾数溢出增大该指数将产生溢出。

表 6.5 乘法类异常中断阈值

最初结果的指数			状态（在全模式时）	
双精度（DP）	单精度（SP）	浮点值	单精度（SP）	双精度（DP）
>0x7FF	-	DP overflow	-	中断
0x7FF	-	DP NaN or 无穷	-	中断
0x7FE	-	DP 最大值 正常	-	中断
0x7FD	-	DP 正常	-	中断
0x7FC	-	DP 正常	-	正常
>0x47F	>0xFF	SP overflow	中断	正常
0x47F	0xFF	SP NaN or 无穷	中断	正常
0x47E	0xFE	SP 最大值 正常	中断	正常
0x47D	0xFD	SP 正常	中断	正常
0x47C	0xFC	SP 正常	正常	正常
0x3FF	0x7F	e = 0 偏移量	正常	正常
0x381	0x01	SP 正常	正常	正常
0x380	0x00	SP 低于正常	中断	正常
<0x380	<0x00	SP underflow	中断	正常
0x001	-	DP 正常	-	正常
0x000	-	DP 低于正常	-	中断
<0x000	-	DP underflow	-	中断

6.3.3 浮点乘累加类指令溢出探测查找表

乘累加类操作通过尾数溢出到指数产生异常，为规格化尾数，最终结果指数最大增量为 2。因此乘累加类指令要在乘法、加法的异常中断阈值基础上考虑最终结果增 2 的因素。

表 6.6 乘累加类操作类型列表

指令	操作数 fd	操作数 fm	操作数 fn	操作类型
FMAC	+	+	+	同号
FMAC	+	+	−	异号
FMAC	+	−	+	异号
FMAC	+	−	−	同号
FMAC	−	+	+	异号
FMAC	−	+	−	同号
FMAC	−	−	+	同号
FMAC	−	−	−	异号
FMSC	+	+	+	异号
FMSC	+	+	−	同号
FMSC	+	−	+	同号
FMSC	+	−	−	异号
FMSC	−	+	+	同号
FMSC	−	+	−	异号
FMSC	−	−	+	异号
FMSC	−	−	−	同号
FNMSC	+	+	+	同号
FNMSC	+	+	−	异号
FNMSC	+	−	+	异号
FNMSC	+	−	−	同号
FNMSC	−	+	+	异号
FNMSC	−	+	−	同号
FNMSC	−	−	+	同号
FNMSC	−	−	−	异号
FNMAC	+	+	+	异号
FNMAC	+	+	−	同号
FNMAC	+	−	+	同号
FNMAC	+	−	−	异号
FNMAC	−	+	+	同号
FNMAC	−	+	−	异号
FNMAC	−	−	+	异号
FNMAC	−	−	−	同号

表 6.7 乘累加类异常中断阈值

最初结果的指数	状态

双精度（DP）	单精度（SP）	浮点值	单精度（SP）	双精度（DP）
>0x7FF	-	DP overflow	-	中断
0x7FF	-	DP NaN or 无穷	-	中断
0x7FE	-	DP overflow	-	中断
0x7FD	-	DP overflow	-	中断
0x7FC	-	DP overflow	-	中断
>0x47F	>0xFF	SP overflow	中断	正常
0x47F	0xFF	SP NaN or 无穷	中断	正常
0x47E	0xFE	SP overflow	中断	正常
0x47D	0xFD	SP overflow	中断	正常
0x47C	0xFC	SP 正常	正常	正常
0x3FF	0x7F	e = 0 偏移值	正常	正常
0x3A0	0x20	SP 正常	最小值(异号)	正常
0x39F	0x1F	SP underflow	中断（异号）or 正常（同号）	正常
0x381	0x01	SP 正常	最小值(同号)	正常
0x380	0x00	SP sub 正常	中断	正常
<0x380	<0x00	SP underflow	中断	正常
0x040	-	DP 正常	-	正常（同号） or 最小值（异号）
0x03F	-	DP underflow（异号）	-	正常 （同号）or 中断（异号）
0x001	-	DP 正常（同号）	-	最小值(同号)or 中断（异号）
0x000	-	DP 低于正常	-	中断
<0x000	-	DP underflow	-	中断

第七章 系统测试及综合

本章介绍浮点协处理器的测试及综合。包括仿真测试方案的制定，测试平台的搭建，以及浮点运算误差的分析并使用综合工具给出 RTL 设计的综合结果。

7.1 系统仿真与测试

7.1.1 方案简介

测试分两个阶段：

第一阶段针对各个功能模块分别编写相应的测试激励，验证环境采用简单的 DUT（Device Under Test）和 TestBench 的形式进行验证，如图 7.1 所示。具体说就是根据 feature list 编写测试用例集。仿真过程中 TestBench 负责通过 DUT 的接口信号向测试模块发控制信号并记录其计算结果，并将其与测试 case 中的参考结果进行自动比较，将比较结果保存在 report 文件中。由于是分模块测试,为使此阶段可以找出尽量多的 bug，我们采用在测试文件中加强制赋值语句,来影响 VFP 内部记分牌和状态寄存器的状态,从而人为设置资源冲突,以此来提高测试可控性和效率.此阶段可以保证覆盖各子模块的绝大部分 case，并找出大部分的 bug。此阶段所测模块与 VFP 设计文档所分模块相匹配。

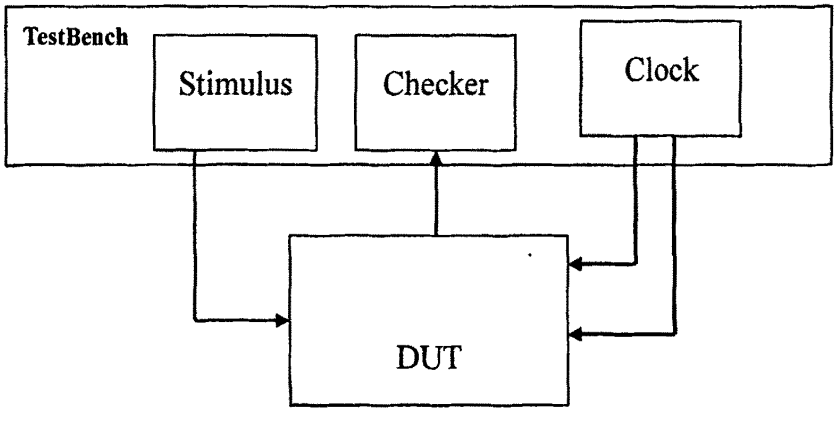

图 7.1 测试环境示意图

第二阶段测试在第一步测试基础上进行，测试环境移植到包含主处理器总线模型 Core BFM（Bus Functions model 简称 BFM）和浮点协处理器参考模型 VFP RM（Reference model，简称 RM）的平台测试环境，如图 7.2 所示。此阶段的激励是汇编格式的指令，随后由验证人员利用软件将指令翻译成二进制指令格式，初始化到 Memory BFM 中，通过 BFM，对 VFP （DUT）进行测试，并由 VFP RM 得出参考测试结果。此类激励主要测试 VFP 的模块整合后的协调性和健壮性，针对人工较难考虑周全的难点，受约束的随机验证成为必不可少的验证环节。同时这也是静态/动态配置大量组合的需求，因为人工不可能去遍历组合，只能选取我们认为重要的，但这样往往容易忽略某些组合。测试结果的比较由 Checker 模块自动完成，并生成详尽的测试结果报告。

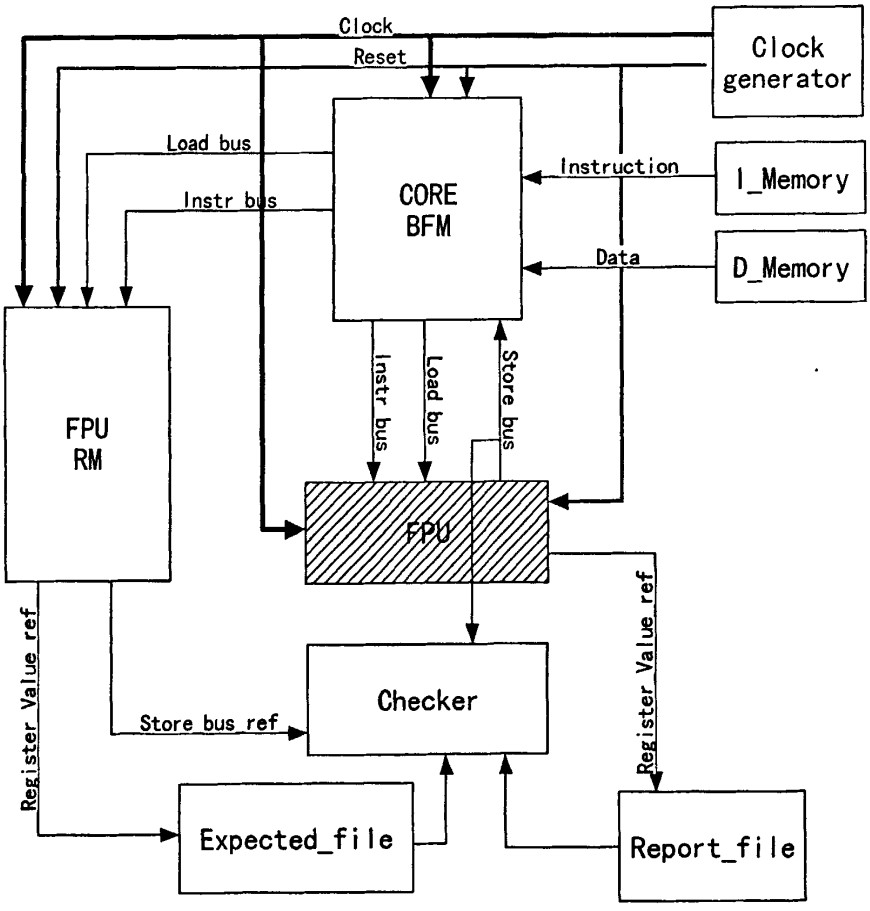

图 7.2 整体验证环境

利用随机测试方法对 VFP 的指令集进行随机组合测试，可以相应提高测试

覆盖率，特别是在测试后期，随机测试可以测试到较多的 corner case 和 boundary case。验证人员在对测试覆盖率统计后，对未覆盖代码行或状态反馈给设计人员，协商解决以保证覆盖率达到设计要求。

我们采用从简单的单一的测试环境到复杂的整合的测试环境过渡，采用从主要模块入手，将影响功能与性能的模块进行优先测试，而把次要的模块放到后面进行测试，并实行先直接 test case（简称 TC），后随机 test case 的策略。

下面主要描述第二阶段。第二阶段直接测试激励基于第一阶段的测试激励，将各模块的测试方案综合以验证模块整合后的协调性，主要是测试单条流水线从取指到写回结束各流水级的协调以及级与级之间 stall 的处理。由于各并行流水线在发射级后分开，且除去除法开方流水线在异常检测时需利用乘加流水线模块外，无相互干扰，因此直接 TC 可以按照三条流水线分为三类，每一类对应一条流水线，所用测试激励综合前期简单测试环境所用激励，并根据覆盖率统计来完备此激励集，最后用一类测试激励专门测试三条流水线指令混合序列.此阶段的测试激励采用汇编语言编写指令，并由工具生成二进制代码，做为测试指令在指令 Memory 中初始化。

测试第二阶段是异常检测，对于不同的异常情况，我们需要不同的关心重点。一些数据异常(主要指 IEEE754 所列情况)，我们需根据 IEEE754 标准进行严格的测试，而另外一些控制异常，我们则需要关心其处理机制是否合理。

随机测试阶段，随机 TC 采用 VFP 指令集随机组合方式，做为测试激励.在随机测试中，利用 Synopsys 的自动化测试生成工具 vera 内嵌的随机生成函数做为测试激励的生成源，此类随机产生函数具有可重复性和可配置性的优点，可以在测试期间将指令和操作数的产生随机化，并比对 RM 与 DUT 的最终结果。

虽然上述测试可以大幅提高测试覆盖率，但为了保证覆盖率达到要求，仍需要用一些实际应用程序代码来测试 VFP(DUT)的功能. 测试程序的选取，我们依据以下原则:有成熟算法，算法规模适中，测试指令丰富。另外我们尽量选取针对 VFP 有相关测试数据的算法：如 3D 图像处理中的矩阵与向量相乘以及矩阵相乘算法，无线通信中的 Viterbi 算法、数字滤波器（如 chebyshev Filter），多媒体领域的 DCT 变换、FFT，数学算法中常用的泰勒级数展开求正弦函数值等。

测试通过标准：第一阶段所用参考结果均为手工得出，因此通过标准为计算

结果与参考结果完全一致。第二阶段为计算结果与 VFP RM 计算参考结果相比较，由于 SystemC 提供了 sc_fixed, sc_ufixed, sc_fix, sc_ufix 数据类型来对硬件的定点数据操作进行位(bit)精确的建模,并且 Floating-point Operator 的输入操作数可以是单精度、双精度、定制三种类型；因此我们建立的 RM 参考模型可以提供与 VFP RTL 相同精度的参考结果。RM 内部数据用 sc_fix 来声明，位宽可以随时调整,因此测试通过标准为两者计算结果一致。

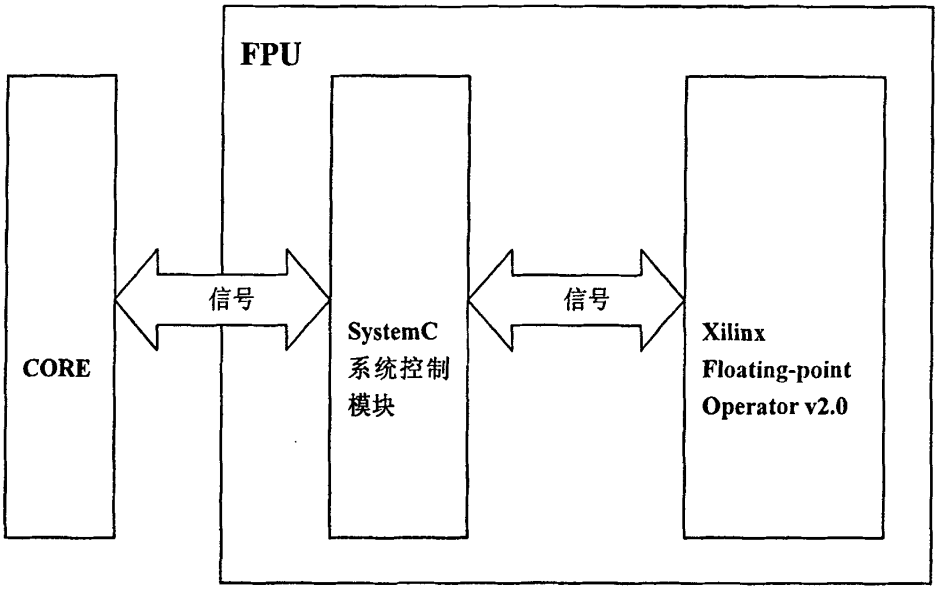

图 7.3 VFP RM 结构图

7.1.2 参考模型设计

VFP 参考模型采用 Xilinx 的 Floating-point Operator 来实现浮点数的加减乘除等运算。SystemC 系统控制模块实现与主处理器的交互、内部流水线的控制以及对 Xilinx 的 Floating-point Operator 的调用，如图 7.3 所示。与主处理器 BFM 的交互通过周期精确的五条控制队列和两条数据总线实现，参考模型采用五级流水线实现。其流水级结构如图 7.4 所示。

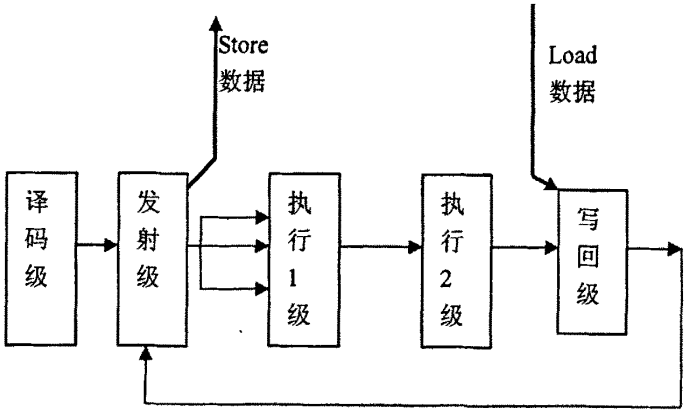

图 7.4 VFP RM 流水线示意图

参考模型采用五级流水，顺序发射，顺序提交。

译码级：实现与主处理器的指令队列的交互

发射级：实现寄存器的读写操作，并对存在 RAW 冲突的指令暂停发射（不存在 WAW 和 WAR 冲突）。并通过 store 数据队列与 ARM11 交互

执行 1 级：计算结果，并根据控制信号决定当前指令是否向下流动

执行 2 级：计算需 latency 为 2 的指令的最终结果

写回级：进行写回控制，通过 Load 数据队列与主处理器交互。

SystemC 系统控制模块与 Xilinx 的 Floating-point Operator 的交互

Floating-point Operator 是由 Xilinx 公司提供的一个 IP 核，可以实现浮点数的加、减、乘、除、开方、比较以及定点与浮点数的转换等运算，操作数可以为单精度、双精度和定制三种类型；影响状态位的操作会有相应的状态输出端口，增加了执行的透明性；执行 latency 可支持 0 到 Maximum(与运算类型相关)，从而为 SystemC 的控制提供了极大的便利。其与 SystemC 系统控制模块的接口信号如图 7.5 所示。

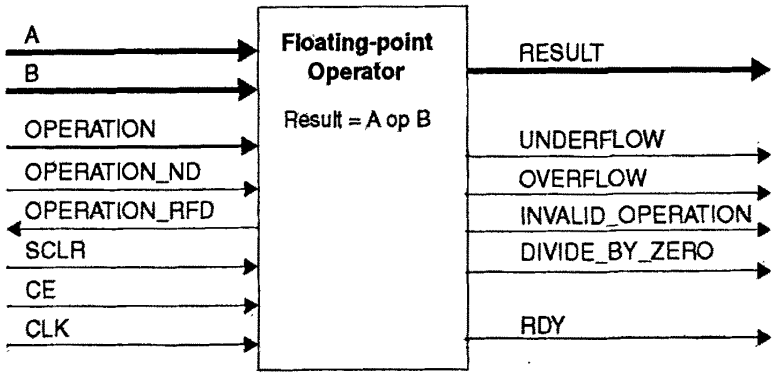

图 7.5 Floating-point Operator 接口信号

SystemC 控制逻辑将寄存器中所读到的操作数或立即数在执行级赋值给 A 和 B 操作数，通过 OPERTION 定义当前操作的类型（乘、加、除法等）；如果是乘加或乘减操作，则需要级联两个 IP 核，乘操作的计算结果与第三个操作数组成下一级运算单元（加或减）的输入操作数，最终得出运算结果，如图 7.6 所示。

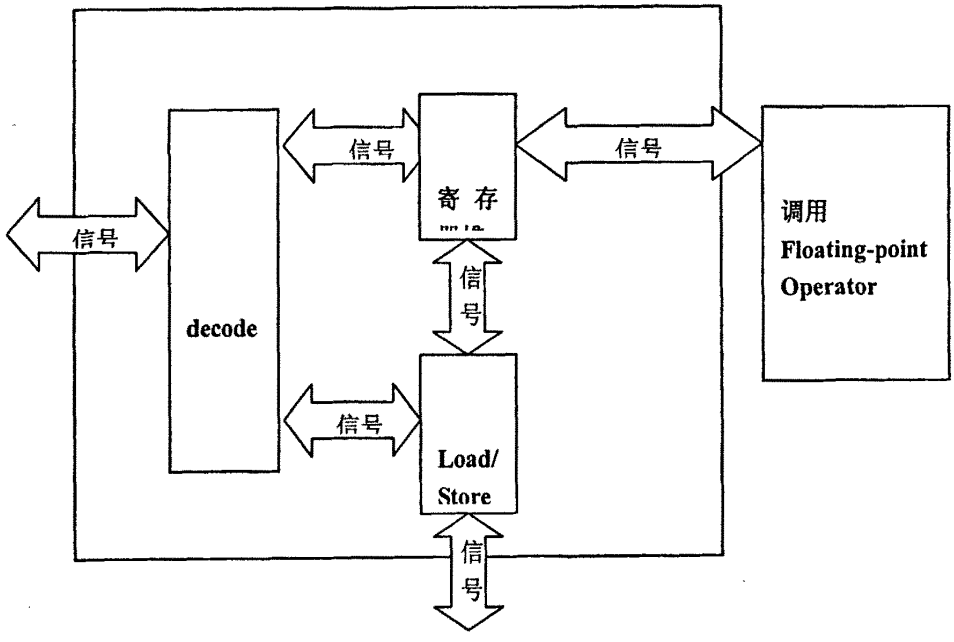

图 7.6 SystemC 控制结构

SystemC 控制部分将操作数和控制指令输出给 Floating-Point Operator 单元，并在结果有效时，将结果写入相应的目的寄存器，SystemC 在写入操作数前可以

提前进行异常探测（如操作数为 NaN，被零除等异常），并在结果有效时对结果的异常位进行判断，一旦发生异常，则调用异常处理函数。

7.1.3 仿真结果

仿真工具使用 Synopsys 的 VCS。仿真波形文件通过 PLI 转化为 Novas 公司的调试软件 Debussy 可以识别的 fsdb 文件。并在 Debussy 中打开。现摘录部分波形如下图 7.7、7.8 所示。

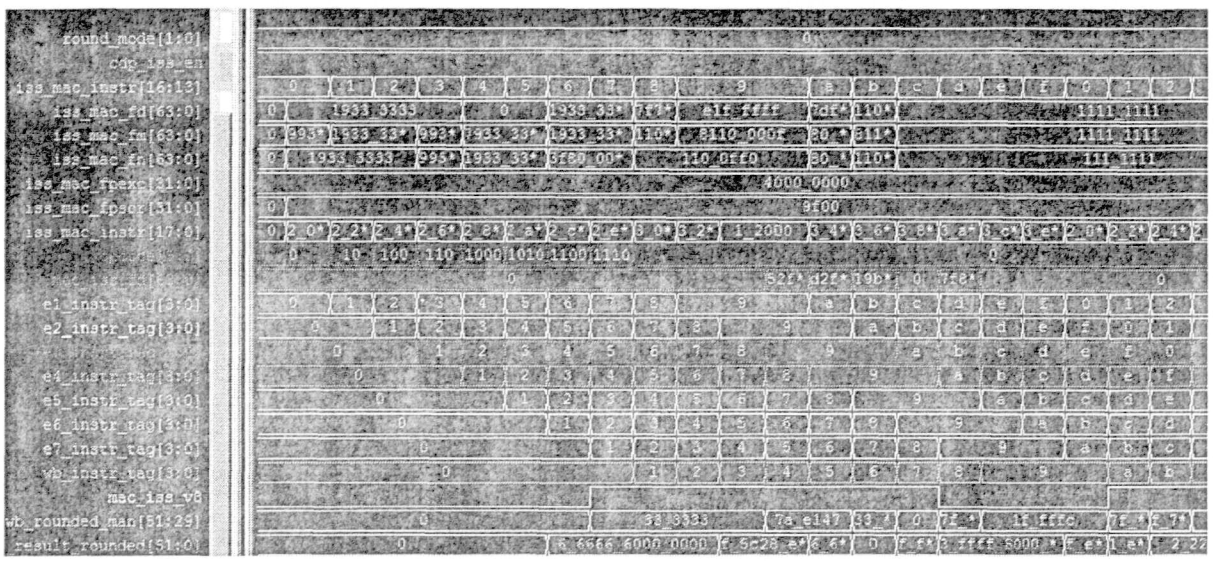

图 7.7 乘累加流水线指令运行波形图

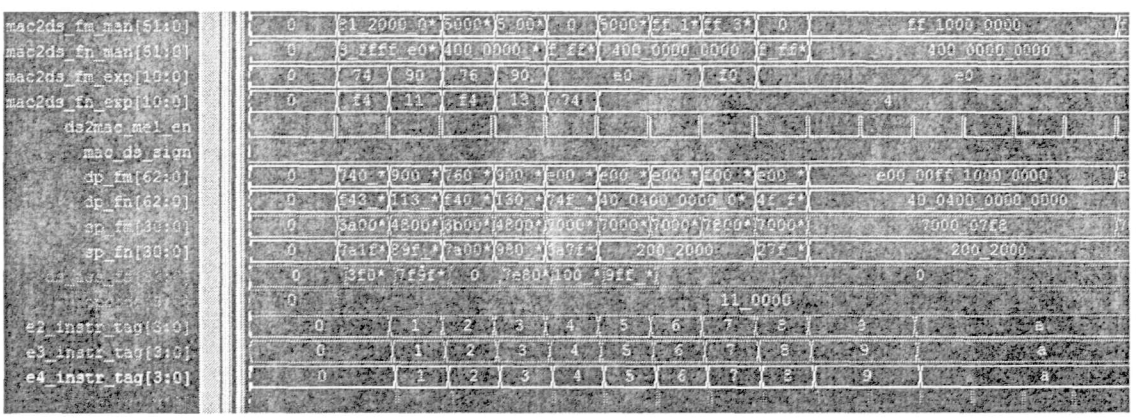

图 7.7 除法开方流水线指令运行波形图

经比对程序（checker）自动化对比生成的结果，得到本浮点协处理器的处理结果同 Xilinx 的浮点程序库运行情况完全吻合。部分 Xilinx 不支持的浮点异常行为也同预先通过软件模拟的参考文件完全一致。这说明整个系统在功能设计上是比较正确的。

7.2 系统舍入误差分析

由第 2 章三个定理就能大致分析清楚在 VFP 中的舍入误差，再把这三个公式列举如下：

$$fl(x)=x(1+\delta),|\delta|\le u$$

$$u=\begin{cases}\frac{1}{2}\beta^{1-t},\text{用舍入法}\\ \beta^{1-t},\text{用截法}\end{cases}$$

$$fl(a\circ b)=(a\circ b)(1+\delta),|\delta|\le u$$

$$1-nu\le\prod_{i=1}^{n}(1+\delta_i)\le 1+1.01nu$$

对于实数转换成浮点数，当在第 n 位进行舍入运算时有：

$$fl(x)=x(1+2^{-n}) \tag{12}$$

特殊情况，当表示为 32 为单精度浮点数时：

$$fl(x)=x(1+2^{-32})$$

当表示为 64 位双精度浮点数时：

$$fl(x)=x(1+2^{-64})$$

对应的相对误差分别是：2^{-n}，2^{-32}，2^{-64}。

对于浮点数算术运算时的误差分析，假如在第 m 位进行舍入运算，则：

$$1-n2^{-m}\le\frac{fl(a\circ b)}{(a\circ b)}\le 1+1.01n2^{-m} \tag{13}$$

当表示为 32 位单精度浮点数时：

$$1-n2^{-32}\le\frac{fl(a\circ b)}{(a\circ b)}\le 1+1.01n2^{-32}$$

当表示为 64 位双精度浮点数时：

$$1-n2^{-64}\le\frac{fl(a\circ b)}{(a\circ b)}\le 1+1.01n2^{-64}$$

其中：$nu\le 0.01$

在乘法过程中是符合 IEEE 标准的规定的双精度的范围内的，整个数据的移

位及运算过程中没有对大于 54 位数据进行舍入，不会丢掉可能对结果有影响的数据，这样就可以判断出乘法器的运算是精确运算，而假如在得出结果时会进行一次舍入，也是在机器精度之内的，完全符合 IEEE 标准。

另外最终结果从 108 位舍入到 54 位的舍入误差为与从无穷远舍入到 54 位的舍入误差之间相差就是从无穷远舍入到 108 位时的舍入误差值，即 2^{-108}。

加法/减法运算采用保护位来保证精确误差。一般来说对于加法/减法的运算在十进制的浮点数中只需要 1 位数就可以了，所以在二进制的浮点数中转换过来则需要 4 位保护位来进行判断，即在 54 位的加法器的后面多加 4 位成为 58 位。在添加保护位的基础上，加法/减法运算是精确运算。

最后进行的舍入是从 57 位舍入到 54 位，相对误差：2^{-54}。相对误差的计算公式为：

$$Relative\_error = \frac{x - fl(x)}{x}$$

除法开方单元整个迭代单元就是产生商/平方根的过程，迭代的次数恰好是二进制浮点数尾数的位数，IEEE 标准下只要满足在双精度或者大于 54 位的精度的操作的运算都是精确的运算，所以整个除法运算也是精确的机器精度内的算术运算。

对于除法开方结果的舍入，舍入单元即要使规格化产生 56 位结果舍成 54 位（双精度）或 25 位（单精度）的尾数，而且是标准的尾数表现形式，即前面两位分别是符号位 0 和默认的 1。得到除法开方运算中的舍入引入的相对误差为：单精度 2^{-25}，双精度 2^{-54}。

7.3 设计综合

综合的过程是借助综合工具将行为描述和 RTL 级别的的电路转换到基于工艺库的门级网表的过程。综合工具根据一个系统逻辑功能与性能的约束，从单元库中的不同的结构、功能、性能逻辑元件中，寻找出一个逻辑网络结构的最佳实现方案。综合主要包括三个阶段：转换(translation)、优化(optimization)与映射(mapping)。转换阶段综合工具将高层语言描述的电路用门级的逻辑单元来实现，优化是综合工具对已有的初始电路进行逻辑、时序、面积分析，化简掉电路中的冗余单元，并对不满足限制条件的路径进行优化，映射是将优化之后的电路映射到由制造商提供的工艺库上。

图 7.8 是业界常用的 Synopsys 公司的综合工具 Design Compiler(简称 DC)综合的的基本流程图。为了实现自动化综合，可以编写脚本由 DC 按照脚本自动完成综合。综合脚本一般包括综合库的指定，指定 DC 自动读取 RTL 设计文件的路径和方法，综合时序面积等约束，DC 的优化方向和努力程度，最后给出综合结果报告。

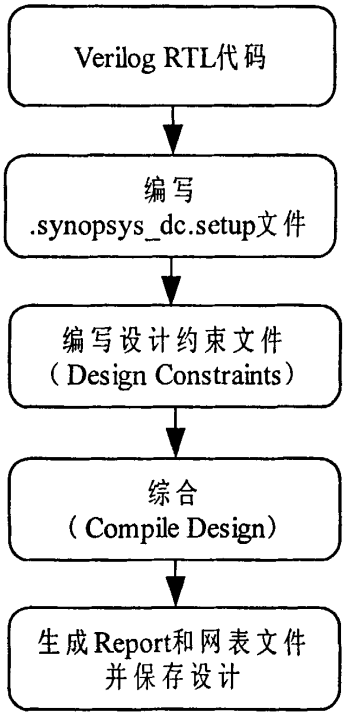

图 7.8 DC 综合流程图

7.3.1 综合脚本

综合的过程使用 TCL 脚本自动化进行。下面是综合使用的脚本的解释。

1. 基本参数设置

```
remove_design -all
##-------------------------------------------##
## SET MODULE NAME                           ##
##-------------------------------------------##
set TOP_MODULE_NAME     [ list "VFP_TOP" ]

##-------------------------------------------##
## SET LIBRARY VARIABLE                      ##
##-------------------------------------------##
set TECH_LIB        [ list "scmetro_tsmc_cl013g_rvt_ss_1p08v_125c" ]

set DRIVE_CELL_BUF1     [ list "BUFX2M" ]
set DRIVE_CELL_BUF2     [ list "BUFX2M" ]
set DRIVE_CELL_BUF4     [ list "BUFX4M" ]
set LOAD_CELL_INV1      [ list "INVX1M" ]
set LOAD_CELL_INV2      [ list "INVX2M" ]
set LOAD_CELL_INV4      [ list "INVX4M" ]

set DRIVER_OUT_PIN      [ list "Y" ]
set DRIVER_IN_PIN       [ list "A" ]
set LOAD_CELL_PIN       [ list "A" ]

set WL_LIBRARY      [ list "scmetro_tsmc_cl013g_rvt_ss_1p08v_125c" ]
set WL_MODEL_MAX        [ list "tsmc13_wl50" ]
set WL_MODEL_MIN        [ list "tsmc13_wl10" ]

set OP_CONDITION_MAX    [ list "scmetro_tsmc_cl013g_rvt_ss_1p08v_125c" ]
set OP_CONDITION_MIN    [ list "scmetro_tsmc_cl013g_rvt_ff_1p32v_0c" ]
set OP_CONDITION_TYPE   [ list "scmetro_tsmc_cl013g_rvt_tt_1p2v_25c"]
```

```
set FALSE               [ list "false" ]
set TRUE                [ list "true" ]
```

2. 读入设计文件

```
#################################################################
### READ DESIGN OBJECTS                          ############
#################################################################
acs_read_hdl  $TOP_MODULE_NAME  -format  verilog  -hdl_source  $RTL_PATH  >
$LOG_PATH/elab_all.log

current_design $TOP_MODULE_NAME
uniquify -force
link

check_design > chk_design.rpt
```

3. 设置工艺要求

```
#################################################################
### SPECIFY TECHNOLOGY REQUIRMENT                  ############
#################################################################
set_max_transition 0.5 [get_ports "*"]
set_max_transition 0.5 [get_designs * -hier]
set_scan_style multiplexed_flip_flop
set_max_fanout 16.0 [get_designs * -hier]
```

4. 设置设计环境

```
#################################################################
### DEFINE DESIGN ENVIRONMENT                      ############
#################################################################

set_operating_conditions $OP_CONDITION_MAX -lib $TECH_LIB
set current_design $TOP_MODULE_NAME
set_wire_load_model -name "tsmc13_wl30" -lib $WL_LIBRARY
set_wire_load_mode enclosed
```

5. 添加约束

```
#####################################################################
### SET DESIGN CONSTRAINTS                                ###########
#####################################################################
##-------------------------------------------##
## CREATE CLOCKs                             ##
##-------------------------------------------##
set CLK_HOLD_UNCERTAINTY        0.25
set CLK_SETUP_UNCERTAINTY       0.25
set CLK_MAX_TRANSITION          0.40
set CLK_MIN_TRANSITION          0.40
set CLK_INFINITE_DRIVE          0
set SYS_CLK                     clk
set SYS_CLK_PERD                2.50

create_clock -period $SYS_CLK_PERD -name $SYS_CLK [ get_port " clk " ]
set_clock_uncertainty -setup  $CLK_SETUP_UNCERTAINTY \
                              [ get_clocks " clk " ]


set_clock_transition -max [ list $CLK_MAX_TRANSITION ] [ get_clocks "*" ]

set_clock_transition -min [ list $CLK_MIN_TRANSITION ] [ get_clocks "*" ]

set_dont_touch_network [ get_clocks "*" ]

set_drive [ list $CLK_INFINITE_DRIVE ] [ list $SYS_CLK ]

##-------------------------------------------##
## CREATE RESET                              ##
##-------------------------------------------##
set RST_SOURCE                  rst_n
set RST_LATENCY_SCALE           0.5
set RST_LATENCY_VALUE           [expr [get_attribute [ get_clocks " clk " ]
\period ] * [ list $RST_LATENCY_SCALE ]]
```

```
    set_input_delay  [ list $RST_LATENCY_VALUE ] \
                     -clock [ list $SYS_CLK ] [ list $RST_SOURCE ]
    set_dont_touch_network [ list $RST_SOURCE ]

    set_drive [ list $CLK_INFINITE_DRIVE ] [ list $RST_SOURCE ]

    ##-------------------------------------------##
    ## CREATE INPUT PORT CONDITION                ##
    ##-------------------------------------------##
    set ALL_INPUTS          [ remove_from_collection [all_inputs]   \
                                  [ list $SYS_CLK $RST_SOURCE ]  ]

    set INPUT_LOAD_NUMBER          6
    set INPUT_FANOUT_LOAD          6

    set_driving_cell -lib_cell [ list $DRIVE_CELL_BUF4 ] \
                     -library  [ list $TECH_LIB ] -pin [ list $DRIVER_OUT_PIN ]
$ALL_INPUTS

    set_load [expr[ load_of [ format %s%s%s%s%s $TECH_LIB "/"
    $LOAD_CELL_INV1 "/" $LOAD_CELL_PIN ]] \
    * $INPUT_LOAD_NUMBER ] $ALL_INPUTS

    set_port_fanout_number [ list $INPUT_FANOUT_LOAD ] $ALL_INPUTS

    set_input_delay [ expr $SYS_CLK_PERD * 0.4 ] -clock $SYS_CLK [ list $ALL_INPUTS ]
    ##-------------------------------------------##
    ## CREAT OUTPUT PORT CONDITION                ##
    ##-------------------------------------------##

    set ALL_OUTPUTS                [ all_outputs ]
    set OUTPUT_LOAD_NUMBER         8
    set OUTPUT_FANOUT_LOAD         8
```

```
set_load [ expr [ load_of [ format %s%s%s%s%s $TECH_LIB "/"
$LOAD_CELL_INV1 "/" $LOAD_CELL_PIN ]] \
* $OUTPUT_LOAD_NUMBER ] $ALL_OUTPUTS
set_port_fanout_number [ list $OUTPUT_FANOUT_LOAD ] $ALL_OUTPUTS
set_false_path -from [ list $RST_SOURCE ]
set_output_delay [ expr $SYS_CLK_PERD * 0.4 ] -clock $SYS_CLK [ list
$ALL_OUTPUTS ]
```

6. 综合方式和力度

```
#########################################################################
### SET COMPILE OPTINIONS                                  ############
#########################################################################

set_structure -design [ list $TOP_MODULE_NAME ] -boolean $FALSE -timing $TRUE
set_flatten -design [ list $TOP_MODULE_NAME ] $TRUE
foreach_in_collection design [ get_designs "*" ] {
   current_design $design
   set_fix_multiple_port_nets -all -feedthroughs -outputs -buffer_constants


current_design $TOP_MODULE_NAME

#########################################################################
###COMPILE                                                 ############
#########################################################################
set_cost_priority { max_delay max_fanout max_capacitance max_transition }
compile -area_effort low

compile -incremental_mapping -map_effort high  -area_effort low
```

7. 生成报告

```
#########################################################################
###     REPORT                                             ############
#########################################################################
set RPT_PATH /mnt/hgfs/rpt
```

```
    ###############
    # Report compile
    ###############
    report_compile_options > $RPT_PATH/rpt_compile.rpt
    write_sdc $RPT_PATH/rpt_sdc.rpt

    ###############
    # Report case
    ###############
    report_case_analysis > $RPT_PATH/rpt_case_analy.rpt

    ###############
    # Check design
    ###############
    check_design > $RPT_PATH/chk_design.rpt

    ###############
    # Check Timing
    ###############
    check_timing > $RPT_PATH/check_timing.rpt

    ###############
    # Report Timing
    ###############
    report_timing -delay max -input_pins -path full_clock -capacitance -net
-max_paths 50 -transition_time -sort_by slack > $RPT_PATH/rpt_timing_setup.rpt
    report_timing -delay min -input_pins -path full_clock -capacitance -net
-max_paths 50 -transition_time -sort_by slack > $RPT_PATH/rpt_timing_hold.rpt

    ###############
    # Report Design
    ###############
    report_design > $RPT_PATH/rpt_design.rpt

    ###############
```

```
    # Report resource
    ###############
    report_resources -hierarchy > $RPT_PATH/rpt_resource.rpt

    ###############
    # Report Area
    ###############
    report_area > $RPT_PATH/rpt_area.rpt

    ###############
    # Report Clock
    ###############
    report_clock -attributes -skew > $RPT_PATH/rpt_clock.rpt
    report_transitive_fanout -clock_tree > $RPT_PATH/rpt_clock_tree.rpt

    ###############
    # Report test
    ###############
    #check_test > $RPT_PATH/check_test.rpt
    #preview_scan > $RPT_PATH/preview_scan.rpt

    ###############
    # Report power
    ###############
    report_power > $RPT_PATH/rpt_power.rpt
    report_clock_gating      -hier   -verbose   -gated   -gating_elements   >
$RPT_PATH/clock_gating.rpt
    report_clock_gating_check > $RPT_PATH/clock_gating_check.rpt

    ###############
    # Report violators
    ###############
    report_constraint -all_violators -verbose > $RPT_PATH/rpt_violate.rpt
    report_constraint -all_violators > $RPT_PATH/rpt_violate_sum.rpt
```

```
###############
# Report high fanout net
###############
report_net_fanout -threshold 10 > $RPT_PATH/rpt_hifanout.rpt

###############
# Write netlist out
###############

#write -format verilog -hierarchy -output $NET_PATH/$TOP_MODULE_NAME.v
write $TOP_MODULE_NAME -format db -hierarchy -output
$DB_PATH/$TOP_MODULE_NAME.db
#write_sdf $NET_PATH/$TOP_MODULE_NAME.sdf

###############
# Quit
###############
quit
```

7.3.2 综合结果

时序在125度、1.08v电压的最坏条件下，得到综合后系统时钟频率301MHz。面积关键路径位于乘累加流水线乘法器所在的执行二级。面积大约10万门左右。由于在深亚微米工艺层次，布局布线前综合得到的面积跟实际面积会有较大误差，因此面积结果在此仅作参考。综上所述，最终的综合结果达到我们预定的设计标准。

结束语

本文介绍了浮点协处理器的 VLSI 设计的方法和完整流程。它采用自顶向下的设计方法，从设计目标的制定、处理器架构的设计到主要运算部件设计、流水线设计完成了整个系统的构建，完成了浮点协处理器的前端设计。利用 Synopsys 公司的 Design Compiler 工具，采用 TSMC 的 0.13μm 的工艺库，对 RTL 设计进行综合，在最差综合状况下得到 300MHz 的系统主频，达到了预定的设计目标，具有十分广阔的应用前景。

学术界和工业界对浮点处理器这种重要的处理部件的研究一直在持续进行中，浮点协处理器的结构和性能一直在不断得到改进。本文虽然吸收了先进的设计思想，提出了一些针对自身特点进行的改进，但是还存在很多待改进的地方。设计中遇到的一些问题还需要进一步的研究。列举其中一些关键问题如下。

作为一款 VLSI 电路设计，协处理器设计的后续工作还有静态时序分析、后仿真、布局布线、DRC(设计规则检查)等等。这将留待进一步工作中完成。

乘法器部分积压缩是系统的关键路径，这部分的逻辑连线复杂，在后期布线占用版图面积较大，有针对布局布线时再次优化压缩器阵列的排列和布局，以便进一步提高速度，减少面积和功耗。

发射级完成冲突检测、向量指令迭代和多流水线指令数据的分发等任务，是影响整个系统的性能瓶颈。后续工作中有待进一步发掘流水线的并行机制。可以尝试应用乱序多发射和寄存器重命名等技术手段对架构加以改进。

本协处理器中向量指令是通过迭代实现的。进一步的性能优化可以考虑使用并行的处理单元实现向量指令个元素之间真正的并行执行。同时这要求有更加复杂的冲突处理机制来保证指令的正确执行。

可以研究多协处理器并行执行的可能性，提出多协处理器并行执行的实现技术。

总之，浮点协处理器的设计仍需持久深入的研究，这里不再赘述。

参考文献

[1] 王金城, 邓博仁, 金西. 一种基于系统级算法的芯片快速成型设计流程. 微型机与应用 [J] 2005 Vol.24 No.3。

[2] 栾玉霞，李存志，32 ×32 乘法器的一种设计，西安电子科技大学学报(自然科学版)，2004年2月第31卷

[3] 何伟，设计高性能浮点加法器，优秀硕士毕业论文，合肥工业大学，2004.3

[4] 刘哲，64位高性能浮点运算单元的设计与验证，上海交通大学硕士论文，2005.3

[5] 刘若珩，双精度浮点运算单元的设计，中国科学院数学与系统科学研究所硕士论文，2001.5

[6] 赵倩，RISC芯片中浮点运算电路的设计，哈尔滨工业大学工学硕士学位论文，2004.3

[7] 李笑盈，浮点加法运算器的电路设计及流水线性能分析，北京科技大学硕士论文，2002.1

[8] ANSI/IEEE Std 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic".

[9] John L. Hennessy, David A. Patterson, "Computer Architecture : A Quantitative Approach", Morgan Kaufmann Publishers,Inc.

[10]John L. Hennessy, David A. Patterson, "Computer Organization & Design The Hardware/Software Interface", Morgan Kaufmann Publishers,Inc.

[11]Sun Microsystems, Inc. "Numerical Computation Guide", Sun Microsystems, Inc.

[12]Jan M. Rabaey "Digital Integrated Circuits- A Design Perspective" Prentice Hall,1998,12

[13]Donald Alpert , Dror Avnon, Architecture of the Pentium Microprocessor, IEEE Micro, v.13 n.3, p.11-21, May 1993

[14]S.F.Anderson, J .G .Earle, R.E.Goldschmidt, D.M.Powers. The IBM System/360 Model 91: Floating-Point Execution Unit. IBM Journal of Research And Development. 1967(11):3453

[15] A.Beaumont-Smith, N.Burgess, S. Lefrere and C.C. Lim, Reduced Latency IEEE Floating-Point Standard Adder Architectures, Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on 14-16 April 1999 Page(s):35 - 42

[16]M.P .Farmwald. On the Design of High Performance Digital Arithmetic Units. PhD Thesis of Stanford University, 1981

[17]H.P.Sit, M.R.Nofai, S.Kim. An 80MFLOPS Floating-Point Engine in the i860 Processor. International Conference on Computer Design.Cambridge, 1989:374-379

[18]R.V.K. Pillai, S.Y.A.Shah, A.J. Al-Khalili, D. Al-Khalili, "Low Power Floating Point MAFs- A Comparative Study" In proceedings of the 6th ISSPA, Kuala Lumpur, Aug. 2001

[19]Peter-Michael Seidel and Guy Even Delay-Optimized Implementation of IEEE Floating-Point Addition 。IEEE TRANSACTIONS ON COMPUTERS, VOL. 53, NO. 2, FEBRUARY 2004

[20]Hiroaki Suzuki, Hiroyuki Morinaka Leading-Zero Anticipatory Logic for High-speed Floating Point Addition。IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 31, NO. 8, AUGUST 1996

[21]Haiping Sun and Minglun Gao Unified Bit Pattern for Leading-Zero Anticipatory Logic for High-speed Floating-point Addition Signal Processing and Information Technology, 2003. Proceedings of the 3rd IEEE International Symposium on 14-17 Dec. 2003 Page(s):786 - 789

[22]David Dahan, 17x17 bit, High-Performance, Fully Synthesizable Multiplier. IFIP International Workshop on Based Synthesis Design – Grenoble, France Dec. 1999

[23]AD Booth. A Signed Binary, Multiplication Technique.Quarterly Journal of Mechanics and Applied Mathematics, 1951, 236～240.

[24]C.S. Wallace, "A suggestion for a fast multiplier", IEEE Trans. Electron. Comp., vol. EC-13, pp. 14-17, Feb, 1964.

[25]L Dadda. Some Schemes for Parallel Multipliers. Alta Frequenza, 1965, 34: 349～356.

[26]Goto, G. Sato, T. Nakajima, M. Sukemura, T, "A 54×54-b regularly structured tree multiplier",Solid-State Circuits, IEEE Journal, Sep 1992, Volume: 27, : 1229-1236

[27]Bedrij O. Carry Select Adder[J] . Electronic Computers , 1962 , 11(2) : 3402346.

[28]BS Cherkauer and EG Friedman, "A hybrid radix-4/radix- 8 low power signed multiplier architecture," Circuits and Systems II: Analog and Digital Signal Processing,

IEEE Trans, Vol. 44 , pp.656-659

[29]Townsend, W.; Swartzlander, E.; Abraham , J. (2003-08-06). "A Comparison of Dadda and Wallace Multiplier Delays". SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII.

[30] G. Even and P.M. Seidel, "A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication," in Proceedings of the 13th Symposium on Computer Arithmetic, pp. 225--232, 1997.

[31]M. Santoro, G. Bewick, and M. Horowitz. Rounding algorithms for IEEE multipliers. In Proceedings 9th Symposium on Computer Arithmetic, pages 176–183, 1989.

[32]N. Quach, N. Takagi, and M. Flynn. On fast IEEE rounding. Technical Report CSL-TR-91-459, Stanford, Jan. 1991.

[33]P.M. Kogge and H.S. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations", IEEE Trans. Computers, Vol. C-22, No. 8, 1973, pp.786-793.

[34]R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", IEEE Transaction on Computers, Vol. C-31, No. 3, p. 260-264, March, 1982.

[35]Beaumont-Smith, A.; Lim, C.C, "Parallel prefix adder design", Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on Volume , Issue , 2001 Page(s):218 – 225

[36]J.D.Bruguera, T.Lang, "Rounding in floating-point addition using a compound adder", Internal Report, Dept. of Electronic and Computer Engineering, University of Santiago de Compostela, Spain, July 2003

[37]Nielsen, A.M. Matula, D.W. Pipelined packet-forwarding floating point. II. An adder Computer Arithmetic, 1997. Proceedings., 13th IEEE Symposium on,6-9 Jul 1997

[38]Ercegovac MD, Tomas Lang, "Radix-4 square root without initial PLA", IEEE Trans on Computers, 1990, 39(8):1016～1024

[39]Alberto Nannarelli, Tomas Lang, "Low-Power Radix-4 Combined Division and Square Root," iccd, p. 236, 1999 IEEE International Conference on Computer Design (ICCD'99), 1999

[40]Gerwig, G., H. Wetter, E. M. Schwarz, J. Haess,C. A. Krygowski, B. M. Fleischer, and

M. Kroener, "The IBM eServer z990 floating-point unit"

[41] Burgess, N.Hinds, C.Cardiff Sch. , Design issues in radix-4 SRT square root & divide unit, Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on 2001, vol.2 : 1646-1650

[42] Alan Edelman, The Mathematics of the Pentium Division Bug, SIAM Review, Vol. 39, No. 1 (Mar., 1997), pp. 54-67

[43] Obermann, S.F.Flynn, M.J.Comput, "Division algorithms and implementations", Computers, IEEE Transactions on Aug 1997, Volume: 46, Issue: 8: 833-854

[44]Ercegovac MD, Tomas Lang. On-the-fly conversion of redundant into conventional representations. IEEE Trans on Computers, 1987, c-36(7):895～897

[45]Tomás Lang , Elisardo Antelo, Radix-4 Reciprocal Square-Root and Its Combination with Division and Square Root, IEEE Transactions on Computers, v.52 n.9, p.1100-1114, September 2003

[46]Tomás Lang , Paolo Montuschi, Very High Radix Square Root with Prescaling and Rounding and a Combined Division/Square Root Unit, IEEE Transactions on Computers, v.48 n.8, p.827-841, August 1999.

[47]Montuschi, P. Ciminiera, L, "Reducing iteration time when result digit is zero for radix 2 SRTdivision and square root with redundant remainders", Computers, IEEE Transactions on Feb 1993, Volume: 42: 239-246

[48]M. Daumas and D.W. Matula, "Recoders for partial compression and rounding", Technical Report RR97-01,Ecole Normale Superieure de Lyon, LIP, 1997.

[49]P.-M. Seidel and G. Even. On the design of fast IEEE floating-point adders. In proceedings of the 15th International Symposium on Computer Arithmetic.2001.US patent pending

[50]D.Goldberg, "Computer arithmetic", Appendix A of J.L.Hennessy and D.A.Patterson, Computer architecture: a quantitative approach, Morgan Kaufmann Publishers Inc, 1990

[51] Indradeep Ghosh, Krishna Sekar, Vamsi Boppana, Design for verification at the register transfer level, Proceddings of the 15th International Conference on VLSI Design, 2002 IEEE

[52]Rawat, K.; Darwish, T.; Bayoumi, M.; "A low power and reduced area carry select adder", Circuits and Systems, Page(s): I -467-70 vol.1, 4-7 Aug. 2002

[53]Dimitrakopoulos, G.; Vergos, H.T, "A family of parallel-prefix modulo 2 adders", Proceedings. IEEE International Conference on Application-Specific Systems, Architectures, and Processors, Page(s): 315 -325, 24-26 June 2003

致 谢

论文是我三年来主要工作的缩影，然而三年的硕士生涯中我收获却远不止如此。这要感谢我的老师和同学一直以来对我的帮助和关怀。

首先，我要衷心感谢我的导师——金西副教授。从踏入实验室的那一刻起，您对我的人生就产生了不可磨灭的影响。您开阔的视野、敏锐的思维启迪我找到前进的方向。您平易近人、一切以学生为本的作风给我们营造了良好的学习和发展的空间。您智慧的谈吐、果敢的行动一直是我学习的榜样。您不但是我学习上的导师，而且将永远是我生活中的良师益友。

感谢多年来同学们和实验室各位兄弟姐妹对我的帮助和支持，让我在这个大家庭中快乐的工作和学习。共同分享成功的喜悦，共同渡过困难的磨练。

感谢王金城和杜学亮师兄对我的关心和帮助，从你们身上我学到了如何做人，如何做事的方方面面。

感谢我的同门孙岩并肩战斗，克服了科研中遇到的种种难题，在学习生活中给我很多帮助。

感谢我的女友曹建兰对我工作和学习的支持，对我生活无微不至的照顾。

最后要感谢我的父母，你们对我的期望一直是我前进的动力，使我终有回报你们的机会。

张鑫

中国科学技术大学

微电子实验室

二零零八年五月

硕士期间发表论文与获奖情况

发表论文

[1] 张鑫，王金城，孙岩，金西. 基于 Radix-8 Booth 译码 Montgomery 模乘的 RSA 算法的设计和硬件实现，小型微型计算机系统，2008 年第 5 卷

[2] 孙岩，张鑫，金西. 基于并行预测的前导零预测电路设计，电子测量技术，2008 年第 31 卷第 1 期

[3] Sun Yan, **Zhang Xin**, Jin Xi. High-Performance Low-power Carry Select Adder using fast all-one finding logic, Second Asia International Conference on Modelling & Simulation Kuala Lumpur, Malaysia 13 – 15 May 2008

获奖情况

[1] 2007 年度，研究生“光华”奖学金

[2] 2008 年度，省优秀毕业生