

# 原创性声明

**郑重声明：**此篇题为《基于 FPGA 的单精度浮点数乘法器设计》的论文，是作者在导师的指导下，于武汉大学攻读硕士学位期间，进行研究工作所取得的成果。根据作者所知，论文中除了参考文献列举的地方外，不包含其他人已经发表或撰写过的研究成果。本声明的一切法律结果由本文作者承担。

**作者签名：**旷捷 毛雪莹 彭俊淇

**导师签名：**黄启俊 常胜

**撰写日期：**二零一零年三月十八日

## 基于 FPGA 的单精度浮点数乘法器设计

作者：旷捷 毛雪莹 彭俊淇

导师：黄启俊 常胜

(武汉大学物理科学与技术学院，武汉，430072)

**摘要：**本文设计了一个基于 FPGA 的单精度浮点数乘法器。乘法器为五级流水线结构。设计中采用了改进的带偏移量的冗余 Booth3 算法和跳跃式 Wallace 树型结构，减少了部分积的数目，缩短了部分积累加的耗时；提出了对尾数定点乘法运算中 Wallace 树产生的 2 个伪和采用部分相加的处理方式，有效地提高了的运算速度；并且加入了对特殊值的处理模块，完善了乘法器的功能。单精度浮点数乘法器在 Altera DE2 开发板上进行了验证，其在 Cyclone II EP2C35F672C6 器件上的最高工作频率达到 212.13 MHz。

**关键词：**改进的带偏移量的冗余 Booth3 算法；跳跃式 Wallace 树；单精度浮点数乘法器；FPGA

### An FPGA Implementation of Single Precision Floating-point Multiplier

Author: KUANG Jie, MAO Xueying, PENG Junqi

Tutor: HUANG Qijun, CHANG Sheng

(Department of Physics Science and Technology, Wuhan University, Wuhan, 430072)

**Abstract:** An FPGA implementation of single precision floating-point multiplier is introduced in this thesis. With the usage of modified redundant Booth3 with bias and leapfrog Wallace tree, and the application of partial addition in fixed-point multiplication, the efficiency of the 5-stage multiplier is promoted. Moreover, a module dealing with special values is introduced to perfect the function of the multiplier. The verification of the multiplier is accomplished on Altera DE2, and the Fmax on Cyclone II EP2C35F672C6 reaches 212.13 MHz.

**Key words:** modified redundant Booth3 with bias; leapfrog Wallace tree; single precision floating-point multiplier; FPGA

随着数字信号处理技术的不断发展，人们对数据的精确性和处理的实时性的要求日益提高，浮点数逐渐取代定点数，成为应用最广的数据格式。如何实现高速的浮点数算法已成为了人们关心的热点问题。传统的利用 DSP 实现浮点算法的方案由于 DSP 串行执行指令的工作方式，在速度上已逐渐难以满足应用的要求。FPGA 作为具有并行处理结构的器件，具有

大量逻辑单元、嵌入式存储资源以及 DSP 处理单元，具备了作为系统核心的条件。因此，利用 FPGA 实现浮点数算法成为了数字信号处理技术发展的一个新的趋势。

作为最常用的基本运算之一，浮点数乘法引起了人们的较多关注。乘法算法从基本迭代算法和并行相加算法发展到了 Booth 算法与 Wallace 树型结构的结合，关键在于部分积数目的减少和部分积相加效率的提高。但是目前应用频率最高的 Booth2 算法（基 4-Booth 算法）和传统 Wallace 树型结构<sup>[1][2]</sup>存在如下缺点：Booth2 算法仅能将 N 位定点乘法的部分积数目从 N 减少至约 N/2；传统的 Wallace 树型结构会由于各条路径输入信号到达时间不一致而引起不必要的延时。采用 Booth2 算法和传统 Wallace 树型结构的文献[1]在 Cyclone II EP2C35F672C8 器件上完成自定义 26 位浮点数乘法运算需要 67.579ns；文献[2]的设计在 5 个时钟周期内完成单精度浮点数乘法运算，但是采用 Cyclone EP1C6Q240C8 器件仅能在 80 MHz 时钟下稳定运行，难以满足数字信号处理技术在运算速度上日益提高的需求。文献[3]-[5]在 Booth2 算法和传统 Wallace 树型结构上作出改进：文献[3]提出带偏移量的冗余 Booth3 算法，能将 N 位定点乘法的部分积数目减至约 N/3，取得比 Booth2 算法更高的运算效率；文献[4]和[5]将跳跃式 Wallace 树型结构引入乘法器电路，以解决传统 Wallace 树型结构各条路径输入信号到达时间不一致的问题。但文献[3]-[5]的方案仅在 ASIC 上进行了实现，且仅应用于定点乘法器，未应用于浮点乘法器。此外，上述浮点数乘法器的文献（[1]和[2]）并未提到对特殊值（如 NaN、零值、无穷大等）的处理。

由于 FPGA 正以其设计的灵活性在数字信号处理领域得到越来越广泛的应用，因此将高效率的 ASIC 算法应用于 FPGA 浮点运算正成为人们研究的重点。本文在带偏移量的冗余 Booth3 算法的基础上作出改进，并引入跳跃式 Wallace 树型结构，在尾数定点乘法运算中对 Wallace 树产生的 2 个伪和采用部分相加的方式以平衡各级流水线的耗时，有效地提高了乘法器的运算速度；同时加入了对特殊值的处理，完善了乘法器的功能。本文的设计在 Altera DE2 开发板上进行了验证。乘法器在 5 个时钟周期内完成单精度浮点数乘法运算，在 Cyclone II EP2C35F672C6 器件上的最高工作频率达到 212.13 MHz。在与文献[1]和[2]选用相同器件进行的时序分析和对比中，本文的设计在运算速度上提高了约 50%。

## 一. 单精度浮点数及运算规则

IEEE 754 标准所定义的单精度浮点数<sup>[6]</sup>由 32 位二进制数表示，其格式如图 1.1。



图 1.1 IEEE754 标准单精度浮点数格式

按区域可划分为符号位 S (sign, 1 位)、指数位 E (exponent, 8 位) 和尾数位 M (mantissa, 23 位)。其中：

- (1) 符号位 S 表示浮点数是正数或负数。S=0 时为正数，S=1 时为负数。
- (2) 指数位 E 表示浮点数带偏移量 127 的指数，实际指数为  $e=E-127$ 。
- (3) 尾数位 M 表示规格化浮点数尾数的小数部分。规格化浮点数的尾数取值在 1 与 2 之间，而 M 省略了该尾数整数部分的 1。

综上所述，图 1.1 表示的单精度浮点数的值为

$$F = (-1)^S \times 1.M \times 2^{E-127} \quad (1.1)$$

相应地，两个单精度浮点数 A 和 B 的乘法运算分为以下四个部分：

- (1) 符号位运算：对被乘数的符号位 signa 和乘数的符号位 signb 做异或运算，得到乘积的符号位 signr。

(2) 指数位运算：由被乘数的指数位  $expa$  和乘数的指数位  $expb$  通过式 (1.2) 得到乘积的指数位  $expr$ 。

$$expr = (expa - 127) + (expb - 127) + 127 = expa + expb - 127 \quad (1.2)$$

(3) 尾数位运算：由被乘数的尾数位  $manta$  和乘数的尾数位  $mantb$  通过式 (1.3) 得到乘积的尾数位  $mantr$ ，利用 24 位定点乘法器实现。

$$1.mantr = 1.manta \times 1.mantb \quad (1.3)$$

(4) 规格化处理：使浮点数乘法运算的结果以符合 IEEE 754 标准的格式表示。首先，判断 (3) 中 24 位定点乘法的 48 位结果的最高位是否为 0。如果不为 0，说明尾数相乘结果的取值不在 1 与 2 之间，需要将尾数右移 1 位，并对  $expr$  作加 1 的修正处理。然后，根据最终的指数位  $expr$  判断运算结果是否上溢出 ( $expr > 254$ ) 或下溢出 ( $expr < 1$ )。

## 二. 单精度浮点乘法器设计思路

单精度浮点数乘法器的设计思路如下：

(1) 在尾数定点乘法运算中采用改进的带偏移量的冗余 Booth3 算法，减少部分积的数目。

(2) 在尾数定点乘法运算中引入跳跃式 Wallace 树型结构，以解决传统 Wallace 树型结构各条路径的输入信号到达时间不同而引起的不必要的延时。

(3) 在尾数定点乘法运算中对 Wallace 树产生的 2 个伪和采用部分相加的方式，平衡各级流水线的延时。

(4) 加入特殊值运算模块，对 NaN、零值、无穷大等特殊情况进行处理。

### 2.1 改进的带偏移量的冗余 Booth3 算法

Booth 算法<sup>[3][7]</sup>通过特殊的编码方式，对乘数进行编码，再根据编码的结果产生部分积。

文献中常用的 Booth2 算法先在乘数的最低位之后补一个 0，然后从低位至高位对乘数的每相邻 3 位数据（有 1 位重叠）进行如表 2.1 的编码（其中  $M$  为被乘数）。对于  $N$  位乘法运算，Booth2 编码能将部分积的个数减少至约  $N/2$ ，其部分积的集合为  $\{\pm 0, \pm M, \pm 2M\}$ 。

表 2.1 Booth2 编码

相邻 3 位	编码	相邻 3 位	编码
000	+0	100	-2M
001	+M	101	-M
010	+M	110	-M
011	+2M	111	-0

Booth3 算法对每相邻 4 位数据进行如表 2.2 的编码。对于  $N$  位乘法运算，Booth3 编码能将部分积的个数减少至约  $N/3$ ，其部分积的集合为  $\{\pm 0, \pm M, \pm 2M, \pm 3M, \pm 4M\}$ 。

表 2.2 Booth3 编码

相邻 4 位	编码	相邻 4 位	编码	相邻 4 位	编码	相邻 4 位	编码
0000	+0	0100	+2M	1000	-4M	1100	-2M
0001	+M	0101	+3M	1001	-3M	1101	-M
0010	+M	0110	+3M	1010	-3M	1110	-M
0011	+2M	0111	+4M	1011	-2M	1111	-0

与 Booth2 编码相比,虽然 Booth3 编码进一步减少了部分积的个数,但是部分积的形式更加复杂。如部分积  $3M$ ,不能由被乘数  $M$  通过简单的移位产生,而需要先将  $M$  左移一位得到  $2M$  再与  $M$  本身相加得到,增加了部分积产生所需的时间。针对这个问题,文献[3]提出了带偏移量的冗余 Booth3 算法,对部分积  $+3M$  的产生做出了改进,如图 2.1。

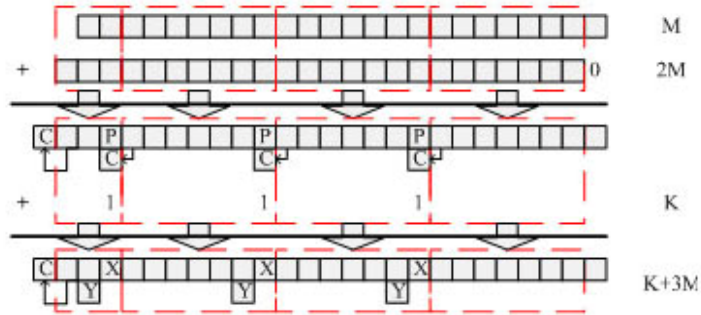


图 2.1 部分积  $+3M$  的产生

该算法将加法运算  $2M+M$  中的  $N+1$  位加法器由若干个位宽较短的加法器代替。低位加法器的 Cout (图中各 C 处)并未连接到高位加法器的 Cin 端口,而是结合偏移量  $K$  和高位加法器结果的最低位  $P$ ,得到  $X$  和  $Y$  值。其中,

$$\begin{cases} X = P \oplus C \\ Y = P \cdot C \end{cases} \quad (2.1)$$

各个短位宽加法器的运算并行进行,高位加法器的运算结果不依赖于低位加法器的运算结果,有效地缩短了 Booth3 算法中部分积产生的时间。

将带偏移量  $K$  的部分积  $+3M$  逐位取反后加 1 (符号位  $S$ ) 得到  $Z$ ,如图 2.2,  $K+3M$  与  $Z$  的和为  $2K$ ,即  $Z$  为带偏移量  $K$  的部分积  $-3M$ 。除  $-3M$  之外,其余各负部分积  $-M$ 、 $-2M$  和  $-4M$  也能根据相同的方法分别由相应的正部分积  $+M$ 、 $+2M$  和  $+4M$  得到。偏移量  $K$  的作用在于,带偏移量的负部分积可以由带偏移量的正部分积通过逐位取反后加 1 的方法得到。在之后对部分积相加的过程中,只需要加上一个补偿常数  $-9K$  ( $9$  为 Booth3 编码产生的部分积个数),就可以抵消各部分积中偏移量的效果,得到正确的结果。

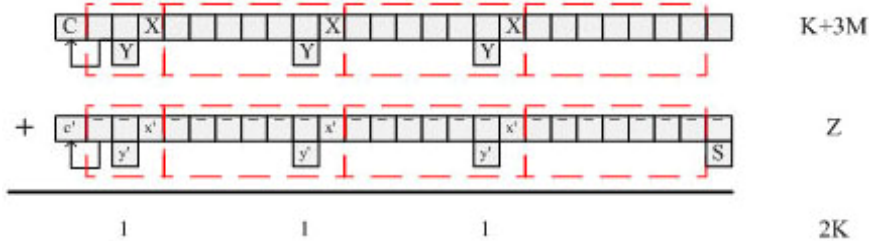


图 2.2 部分积  $-3M$  的表示

对于部分积中  $Y$  (或  $y'$ ) 位和符号位  $S$  (在本文中统称为冗余位) 的处理,文献[3]提出将  $-RK$  单独作为一个部分积 compensation,并将各部分积的冗余位拼接成另一个部分积 extra,如图 2.3。这样在之后的 Wallace 树中共有 11 个部分积参与相加过程: 9 个 Booth3 编码的结果 partial\_product0~partial\_product8、补偿常数 compensation 和冗余位 extra。本文在此方案上作出改进。由于部分积 partial\_product8 的低 24 位均为 0、补偿常数 compensation=48'hF6C900093700 也有较多位数为 0,所以可以将各部分积的冗余位分别填入 partial\_product8 的低位和补偿常数 compensation 中,如图 2.4,绿色的冗余位填入 partial\_product8 的低位,红色的冗余位填入补偿常数 compensation。经此改进后,参与 Wallace 树相加过程的部分积个数减少至 10 个,进一步减少部分积累加过程的运算量。

## 2.2 跳跃式 Wallace 树型结构

Wallace 树型结构<sup>[8]</sup>由进位存储加法器（CSA）构成的 3-2 压缩器、4-2 压缩器<sup>[9][10]</sup>取

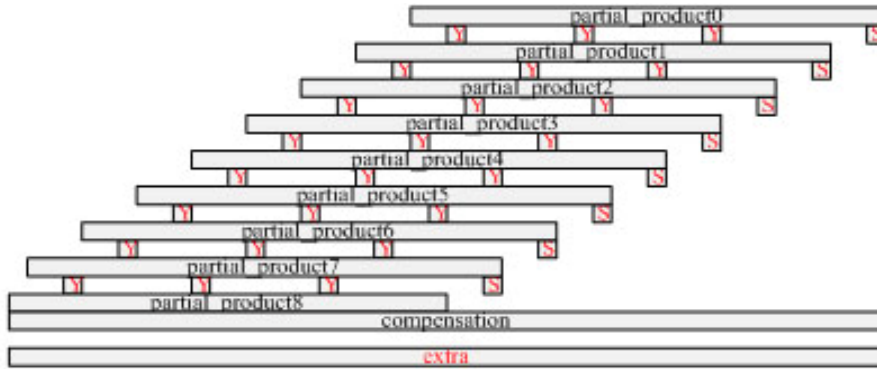


图 2.3 文献[3]提出的方案

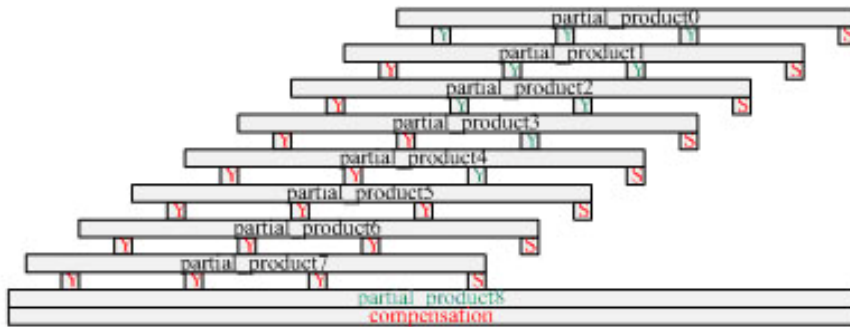


图 2.4 本文改进后的方案

代传统树型结构中的加法器，对各部分积进行累加运算。3-2 压缩器由  $N$  个 1 位全加器组成，将 3 个加数  $dataa[N-1:0]$ 、 $datab[N-1:0]$  和  $datac[N-1:0]$  的各位分别接至各全加器的 3 个输入端口，并将各全加器的输出  $S[N-1:0]$  和  $C[N:0]$  作为 3-2 压缩器的压缩结果，如图 2.5。传统的 4-2 压缩器由两级全加器构成，如图 2.6，相当于连续做两次 3-2 压缩。

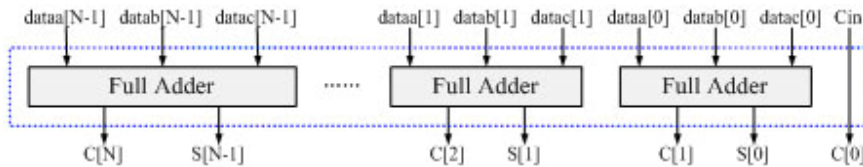


图 2.5 3-2 压缩器

与加法器相比，3-2 压缩器和 4-2 压缩器的优势在于其延时较小。3-2 压缩器的延时为一个全加器的延时，4-2 压缩器的延时为两个全加器的延时，且基本上不受位数  $N$  的影响。文献[10]提出了利用 2 选 1 多路选择器代替全加器中异或门的改进方案，如图 2.7，缩短了 4-2 压缩器关键路径的延时，并使 4 个输入信号同时参与运算，从而优化了 4-2 压缩器的性能。

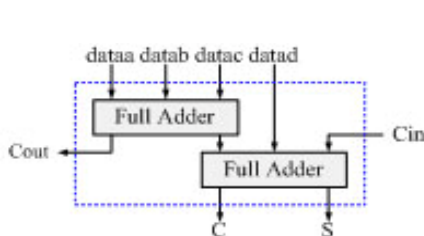


图 2.6 传统的 4-2 压缩器

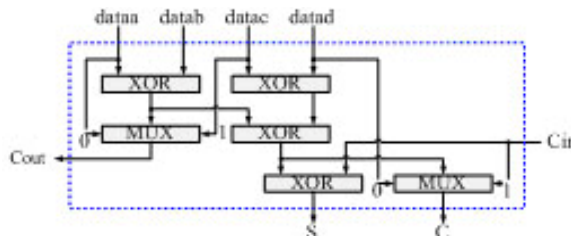


图 2.7 改进的 4-2 压缩器

再从 Wallace 树的结构上考虑。由于 3-2 压缩器和 4-2 压缩器中不同的门电路存在不同的延时，因此如何使 Wallace 树各条路径的输入信号是否同时到达成为了影响累加运算耗时的重要因素。文献[11]提出设 1 级门的延时为  $1td$ ，则与、或、与非、或非逻辑门的延迟为 1

td, 而异或门的延迟为 2 td。根据此假设, 3-2 压缩器的 C 输出延时为 2td, S 输出延时为 4td; 4-2 压缩器的 C 输出延时为 4td, S 输出延时为 6td<sup>[5][11]</sup>。为了使 Wallace 树各条路径的输入信号尽量同时到达, 本文提出了如图 2.8 的跳跃式 Wallace 树型结构对由改进的带偏移量的冗余 Booth3 算法产生的 10 个部分积进行累加。将第一级的三个 3-2 压缩器的 S 和 C 输出信号分开, 三个 S (4td) 送入第二级的 3-2 压缩器, 三个 C (2td) 和另一个部分积送入第二级的 4-2 压缩器, 这样可以使得第二级 3-2 压缩器和 4-2 压缩器的 S 输出信号同时到达第三级的 4-2 压缩器 (8td)。运用跳跃式 Wallace 树型结构, 从 10 个部分积输入到累加至 2 个伪和输出共延时 14td。与图 2.9 所示的运用传统 Wallace 树型结构对由 Booth2 算法产生的 13 个部分积的累加过程 (共延时 18td) 相比, 本文的设计有效地缩短了累加过程所需的时间。

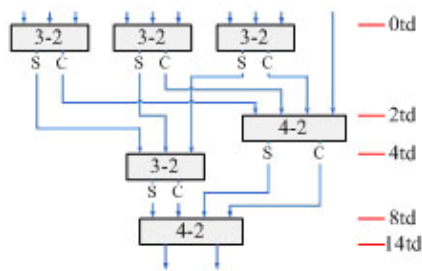


图 2.8 跳跃式 Wallace 树型结构

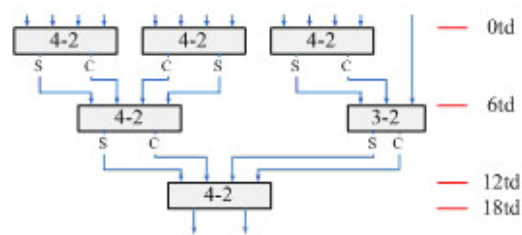


图 2.9 传统 Wallace 树型结构

### 2.3 伪和的部分相加

对于 Wallace 树型结构输出的 2 个 48 位伪和, 各文献中传统的处理方式是运用加法器直接相加得到定点乘法的结果。由于 48 位加法运算耗时较长, 使得加法器所在的一级流水线运算较慢, 因此影响了整个流水线乘法器的速度。实际上, 单精度浮点数乘法的尾数运算仅需要定点乘法结果的高 26 位, 所以本文提出对 2 个伪和采用部分相加的方式, 用 1 个 26 位加法器和 1 个 22 位加法器代替 48 位加法器, 如图 2.10。低 22 位加法器的进位输出 mult\_result\_cin 并未作为高 26 位加法器的进位输入信号, 而是直接输出, 用于后续尾数规格化处理。相应地, 需要对尾数规格化处理的舍入逻辑作出如表 2.3 的修正。

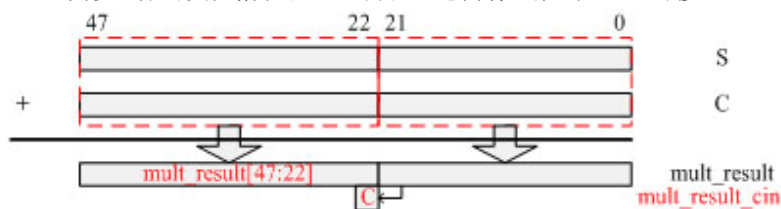


图 2.10 伪和的部分相加

表 2.3 尾数规格化舍入逻辑的修正

mult_result[47]	原舍入逻辑		修正的舍入逻辑	
1'b0	mult_result[22]==1'b0	舍	mult_result[22]==1'b0 && mult_result_cin==1'b0	舍
	mult_result[22]==1'b1	入	mult_result[22]==1'b1    mult_result_cin==1'b1	入
1'b1	mult_result[23]==1'b0	舍	mult_result[23]==1'b0 && {mult_result[22],mult_result_cin}!=2'b11	舍
	mult_result[23]==1'b1	入	mult_result[23]==1'b1    {mult_result[22],mult_result_cin}==2'b11	入

采用部分相加的方式, 伪和的高 26 位加法和低 22 位加法可以并行执行, 缩短了定点乘法运算所需的时间。如此改进的依据是, 尾数规格化处理部分的最高工作频率 (274.20 MHz)

远高于定点乘法运算的最高工作频率（185.08 MHz），而流水线结构的整体速度由速度最慢的流水线级决定，所以流水线浮点数乘法器的最高工作频率（191.13 MHz）受到了定点乘法运算所在流水线级的制约。采用部分相加的方式平衡各级流水线的耗时，虽然使尾数规格化处理略变复杂，但是有效地提高了定点乘法运算的速度，从而提高了整个浮点数乘法器的最高工作频率，达到 212.13 MHz。

## 2.4 特殊值的处理

单精度浮点数可以表示如 NaN（not a number）、无穷大和零值等特殊值，其中：

- (1) NaN:  $E=255, M \neq 0$ 。
- (2) 无穷大:  $E=255, M=0$ 。根据 S 分为正无穷大和负无穷大。
- (3) 零值:  $E=0$ 。

特殊值的运算规则如表 2.4。

表 2.4 特殊值运算规则

被乘数	乘数	积	被乘数	乘数	积
NaN	NaN	NaN	零值	NaN	NaN
	无穷大	NaN		无穷大	NaN
	零值	NaN		零值	零值
	有效值	NaN		有效值	零值
无穷大	NaN	NaN	有效值	NaN	NaN
	无穷大	无穷大		无穷大	无穷大
	零值	NaN		零值	零值
	有效值	无穷大		有效值	有效值

本文在单精度浮点数乘法器的设计中加入特殊值运算模块，根据被乘数和乘数判断运算结果是否为特殊值。如果运算结果是特殊值，则由相应的端口输出特殊值标识信号，实现与 Altera 浮点数乘法 IP 核 ALTFP\_MULT 相同的扩展功能。

## 三. 单精度浮点乘法器模块设计

单精度浮点数乘法器的结构如图 3.1，由 sign、mult\_24bit、nan\_zero\_inf、exponent 和 mantissa 五个模块构成。

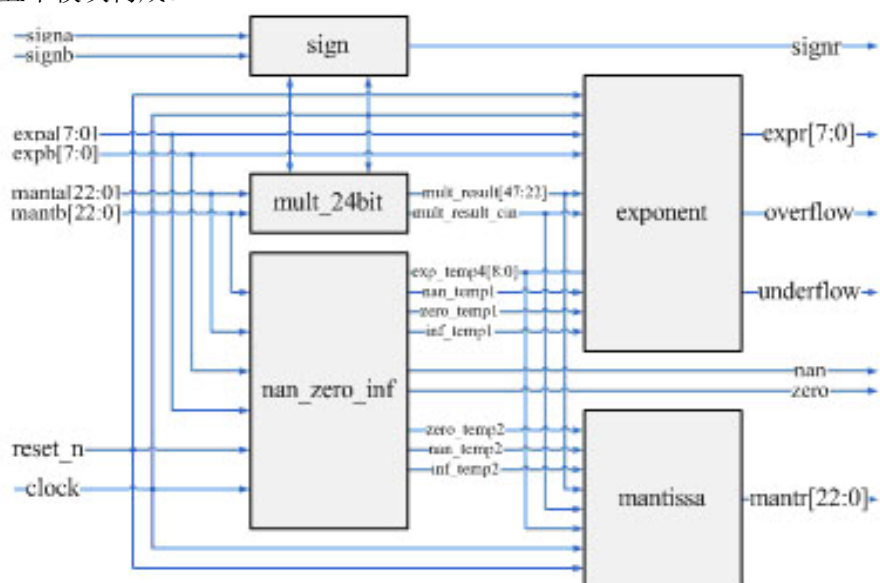


图 3.1 单精度浮点数乘法器结构图

### 3.1 sign 模块

sign 模块对被乘数的符号位 signa 和乘数的符号位 signb 作异或运算，得到乘积的符号位 signr。

### 3.2 mult\_24bit 模块

mult\_24bit 模块为 24 位定点乘法器，用于对被乘数的尾数位 manta 和乘数的尾数位 mantb 进行乘法运算，其结构如图 3.2。

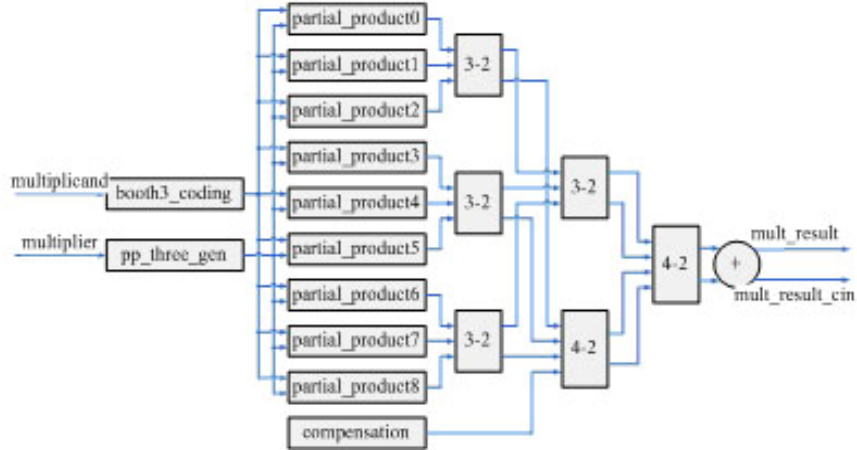


图 3.2 mult\_24bit 模块结构图

模块分为三级流水线：

#### (1) Booth3 编码和部分积的产生

流水线的第一级由 booth3\_coding、pp\_three\_gen、compensation 和 9 个 partial\_product 子模块构成。booth3\_coding 对乘数进行 Booth3 编码，编码结果送至 9 个 partial\_product。各 partial\_product 根据相应的编码结果分别利用多路选择器生成相应的部分积，其中 +2M 和 +4M 由被乘数移位产生、+3M 由 pp\_three\_gen 采用带偏移量的冗余 Booth3 算法产生，并根据编码结果判断是否需要为正部分积进行取反加 1 操作。各部分积的冗余位按照图 2.4 分别填入 compensation 子模块和 partial\_product8 子模块中。

#### (2) Wallace 树型结构

流水线的第二级采用图 2.8 所示的跳跃式 Wallace 树型结构，利用 3-2 压缩器和 4-2 压缩器，将部分积从 10 个累加至 2 个。

#### (3) 伪和的部分相加

流水线的第三级将 Wallace 树输出的 2 个伪和通过如图 2.10 的方式部分相加，得到高 26 位的加法结果 mult\_result[47:22] 和低 22 位的加法进位输出 mult\_result\_cin，用于后续尾数规格化处理。

### 3.3 nan\_zero\_inf 模块

nan\_zero\_inf 模块根据表 2.4 的运算规则，通过被乘数和乘数判断积是否为 NaN、无穷大或零等特殊值。若为 NaN 和零值，则直接由 nan 和 zero 端口输出标识信号；若为无穷大，则发送信号至 exponent 模块，通过 exponent 模块输出 overflow 信号。

### 3.4 exponent 模块

exponent 模块用于计算浮点数乘法的指数位，并判断运算结果是否溢出。

首先由 nan\_zero\_inf 模块的输出信号判断运算结果是否为特殊值，若为特殊值则按特殊值的规则输出相应的指数位。若运算结果不为特殊值，则由被乘数和乘数的指数位 expa 和 expb 计算积的粗略指数位，并根据 mult\_24bit 模块的运算结果判断是否需要指数位作加 1

的修正处理。最后，根据修正后的指数位判断运算结果是否上溢出或下溢出。其中，根据 mult\_24bit 模块运算结果的加 1 修正和溢出判断共需要 2 级流水线完成。

### 3.5 mantissa 模块

mantissa 模块用于对乘积的尾数部分进行规格化处理。

首先由 nan\_zero\_inf 模块的输出信号判断运算结果是否为特殊值，若为特殊值则按特殊值的规则输出相应的尾数位。若运算结果不是特殊值，则由 mult\_24bit 模块运算结果的最高位判断是否需要将尾数位右移 1 位。最后，对尾数位进行舍入（保留 24 位），并省略整数部分的 1 后输出。其中，根据 mult\_24bit 模块运算结果的右移和舍入共需要 2 级流水线完成。

## 四. 综合与验证

本文的设计在 Quartus II 9.0 中进行了综合和仿真，并在 Altera DE2 开发板上进行了硬件验证。

在 Quartus II 选择 Altera DE2 开发板所用的 Cyclone II EP2C35F672C6 器件对单精度浮点数乘法器进行综合，综合结果如表 4.1，并与 Altera 单精度浮点数 IP 核作了对比。

表 4.1 单精度浮点数乘法器综合结果

	Logic elements	Registers	Embedded Multiplier 9-bit elements	Latency	Fmax (MHz)
本文设计	1895	506	0	5	212.13
Altera IP (未调用 DSP 资源)	932	534	0	5	131.77
Altera IP (调用 DSP 资源)	269	219	7	5	164.47

本文的设计完全由逻辑单元实现，没有调用片内 DSP 资源。由于采用了改进的算法和结构，本文的设计虽然占用了较多的 Logic elements，但是在速度上有明显的优势，Fmax 达到 212.13 MHz。在流水线级数相同的情况下，与同样未调用片内 DSP 资源的 Altera IP 核相比，Fmax 提高了 61%；与调用片内 DSP 资源（7 个 9-bit 乘法单元）的 IP 核相比，Fmax 提高了 29%。

表 4.2 将本文的设计与文献[1]和[2]的时序分析结果进行对比，反映了本文算法和结构在速度上相对于 Booth2 算法和传统 Wallace 树型结构的优势。为了排除器件差异对 Fmax 的影响，表 4.2 中分别采用与文献[1]和[2]相同的器件进行时序分析。

表 4.2 时序分析结果对比

	Device	Latency	Fmax (MHz)	Time (ns)	
本文	Cyclone II	5	152.98	32.68	
文献[1]	EP2C35F672C8	-	-	67.579	自定义 26 位浮点数
本文	Cyclone	5	127.55	39.20	
文献[2]	EP1C6Q240C8	5	80	62.50	

与文献[1]相比，本文的设计只用到 48%的运算时间完成了在更高精度的浮点运算；与文献[2]相比，在流水线级数相同的情况下，本文设计的 Fmax 提高了 59%。对比结果充分说明了本文的算法和结构在运算速度上的优势。

本文的设计在 Quartus II 中的仿真结果如图 4.1。与 Altera IP 核的运算结果作对比，本文的设计能正确地计算出浮点乘法的结果，并能判断运算结果是否特殊值或溢出。

在 Altera DE2 开发板上对本文的设计进行硬件验证，验证方案如图 4.2。

首先将被乘数和乘数的数据文件通过 DE2\_control\_panel 写入 DE2 开发板的 FLASH 中。然后将包含单精度浮点数乘法器的测试模块下载到 Cyclone II FPGA。测试模块中还包含

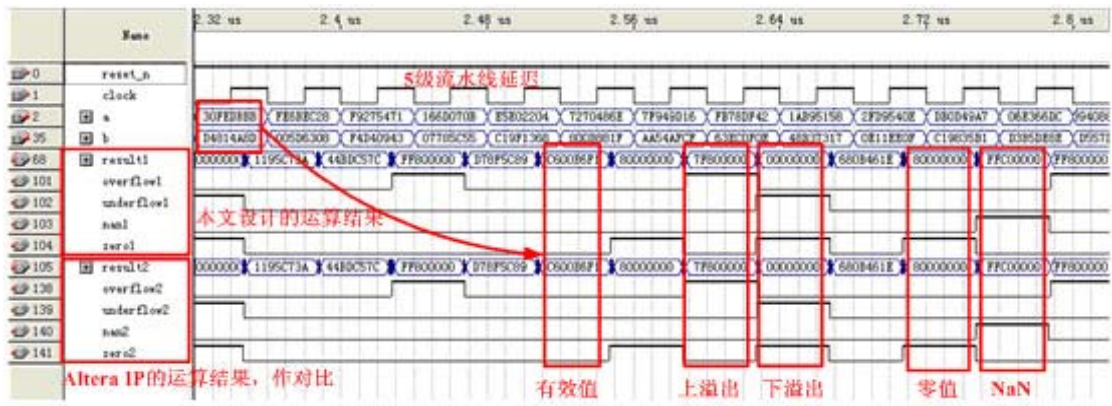


图 4.1 单精度浮点数乘法器仿真结果

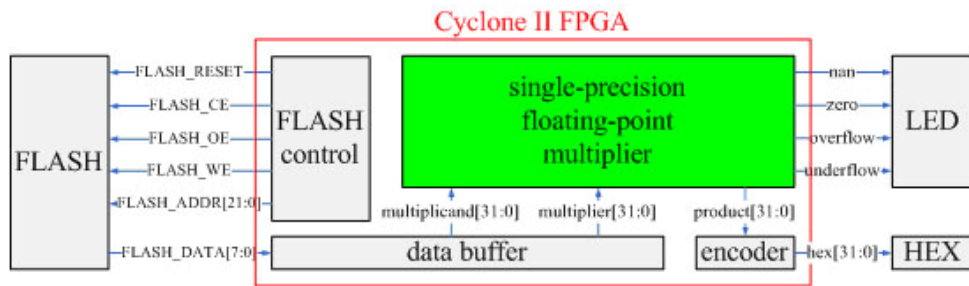


图 4.2 Altera DE2 硬件验证方案

FLASH 控制模块 FLASH\_control、数据缓冲模块 data\_buffer 和七段码编码模块 encoder。在 FLASH\_control 的控制下，data\_buffer 读取 FLASH 中的数据、缓冲后作为输入信号发送给乘法器。乘法器运算出的乘积通过 encoder 编码后由 DE2 开发板的七段数码管 HEX7~HEX0 以十六进制的形式显示，nan、zero、overflow 和 underflow 信号分别由 DE2 开发板的 LED 灯 LEDR3~LEDR0 显示，如图 4.3。

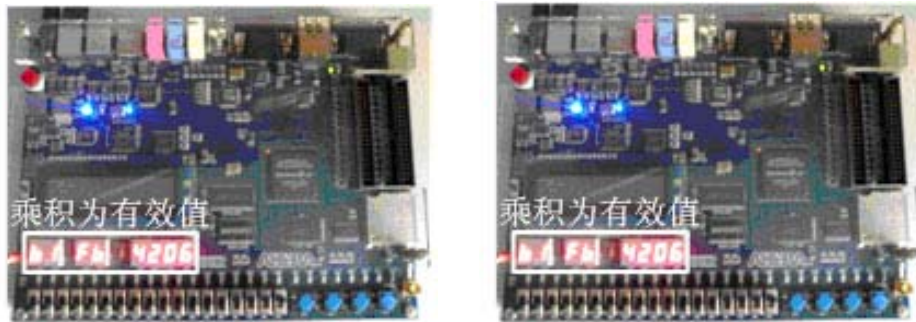


图 4.3 Altera DE2 硬件验证

左图为 multiplicand=32'hB6295A1C 和 multiplier=32'h3B3DE804 的运算结果，运算结果在有效范围内，product=32'hB1FB4206；右图为 multiplicand=32'h73923ABC 和 multiplier=32'h5E78A63C 的运算结果，运算结果上溢出，product=32'h7F800000（无穷大），且输出 overflow 标识信号。单精度浮点数乘法器在 Altera DE2 开发板上能够输出正确的运算结果，因此，其功能在硬件上得到了验证。

#### 四. 结束语

本文设计了一个基于 FPGA 的单精度浮点数乘法器。乘法器采用了改进的带偏移量的冗余 Booth3 算法和跳跃式 Wallace 树型结构，减少了部分积的数目，缩短了部分积累加的耗时；提出了对尾数定点乘法运算中 Wallace 树产生的 2 个伪和采用部分相加的处理方式，有效地提高了的运算速度；并且加入了对特殊值的处理模块，完善了乘法器的功能。本文的设计在 Altera DE2 开发板上进行了验证。乘法器在 5 个时钟周期内完成单精度浮点数乘法

运算, 在 Cyclone II EP2C35F672C6 器件上的最高工作频率达到 212.13 MHz。该单精度浮点数乘法器已应用于高速数字信号处理系统的设计中。

## 参考文献

- [1] 金美华, 宋万杰, 吴顺君。FPGA 中浮点乘法器的实现。火控雷达技术。2008.37(1)
- [2] Gong Renxi, Zhang Shangjun, Zhang Hainan. Hardware implementation of a High Speed Floating Point Multiplier Based on FPGA. Proceedings of 2009 4th International Conference on Computer Science and Education, ICCSE 2009. 2009
- [3] Gary W Bewick. Fast multiplication algorithms and implementation. Stanford University. 1994.2
- [4] 孙海珺, 邵志标, 迟晓明。基于冗余算法和跳跃式结构的 54 位乘法器的研究。西安交通大学学报。2006.40(2)
- [5] 李伟, 戴紫彬, 陈韬。基于跳跃式 Wallace 树的低功耗 32 位乘法器。计算机工程。2008.34(17)
- [6] ANSI/IEEE Std.754-1985. IEEE Standard for binary Floating-Point Arithmetic. IEEE, Inc, New York. 1985
- [7] Andrew D Booth. A signed binary multiplication technique. Quarterly Journal of Mechanics and Applied Mathematics. 1951, 4(2)
- [8] C S Wallace. A suggestion for a Fast Multiplier. IEEE Transactions on Electronic Computers. 1964, 13(2)
- [9] 常静波, 郭立。一种 3 级流水线 Wallace 树压缩器的硬件设计。微电子学与计算机。2005.22(1)
- [10] 赵忠民, 林正浩。一种改进的 Wallace 树型乘法器的设计。电子设计应用。2006.8
- [11] Shivaling S M, Poras T B. High Performance Low Power Array Multiplier Using Temporal Tiling. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 1999, 7(1)