河北工業大学

HEBEI UNIVERSITY OF TECHNOLOGY

# 硕士学位论文

## MASTER THESIS

学　　院：计算机科学与软件学院

学科专业：控制科学与工程

论文作者：罗淑贞

指导教师：耿恒山　教授

河北工业大学研究生院　2014　年　11　月

分类号：TP332.2　　密级：
U D C：004.31　　编号：201222101002

河北工业大学硕士学位论文

# 基于FPGA的浮点乘加融合部件的研究及算法

论 文 作 者：罗淑贞　　学 生 类 别：全日制
学 科 门 类：工学硕士　　学 科 专 业：控制科学与工程
指 导 教 师：耿恒山　　职　　称：教授

Dissertation Submitted to
Hebei University of Technology
for
The Master Degree of
Control Science and Engineering

# RESEARCH ON FLOATING-POINT MULTIPLY-ADD FUSED UNITS AND THE ALGORITHM BASED ON FPGA

by
LUO Shu Zhen

Supervisor: Prof. GENG Heng Shan

November 2014

# 摘　要

随着国内高性能 CPU 的快速发展，研究具有高精度的浮点乘加融合部件对推动高性能处理器的研究具有重要意义。然而国内对浮点乘加部件的研究和国外的水平仍存在一定差距，还有很大的发展空间。

本论文旨在降低浮点运算的延时，提升速度，通过深入分析现今浮点乘加融合思想与结构，完成了对浮点乘加融合体系结构的设计。论文通过对系统结构模块化，把系统分为以下主要模块：解码模块，乘法器模块，加法器模块，前导 1 预测模块，规格化和舍入模块等，且主要通过设计前导 1 预测环节中的关键算法来完成降低延时的目的，最后对各个模块进行综合仿真，并在 Altera 公司的 DE2 平台上进行仿真实现。

论文的重要创新点在于设计三操作数前导 1 预测算法。在这一模块先是分析了当前两操作数前导 1 预测算法的编码规则，并深入探讨了其存在的不足，并针对这一不足，在 FPGA 平台上设计了能够直接处理三操作数的前导 1 预测算法的完整实现方案，可以有效降低关键路径延时和功耗。论文重点设计出了三操作数的编码树结构和预测算法的预编码规则，通过在 FPGA 硬件验证平台上对系统结构合理模块化，且采用硬件描述语言 VerilogHDL 对部分功能进行编程，优化了设计过程，最后对仿真结果进行了分析。仿真结果表明，设计完成的算法结构较传统算法在关键路径延时上减少 36.15%，功耗降低 39.20%。

最后，在浮点乘加部件的基础上完成了浮点乘加融合系统结构的设计，并利用 FPGA 技术实现了乘加融合模块的仿真。通过仿真实现来验证各部件结果，由验证结果可知，由此设计出来的浮点乘加融合结构有效的降低了延时，提升了速度。

**关键词：**浮点乘加融合　前导 1 预测算法　三操作数　VerilogHDL

# ABSTRACT

With the rapid development of the domestic CPU, high-performance floating-point multiply fusion research,which has independent intellectual property, is significant for improving performance.

This paper has fulfilled the design of floating point multiply-add system to reduce the path delay through in-depth analysis of the throught and structure.The system has divided into the following main modules:decoding module, the multiplier module, adder module, leading-one prediction module, as well as normalization and rounding module, which the leading-one prediction is the key algorithm in the system to reduce the path delay.

The most important innovation is the design of leading-one prediction module. A method is adopted to deal directly with three-operand on FPGA platform and designed three-operands complete prediction algorithm implemention. This method can reduce the critical path delay and power consumption. The paper focused on the design of three-operands encoding tree structure, and based on the pre-encoding rules on FPGA hardware verification platform, reasonable modular for system architecture, using hardware description language VerilogHDL to program some function module to optimize the design process and finally the results are analyzed and verified. Compared with the design using the traditional algorithm,the one using the proposed algorithm can reduce the delay of the critical path by36.15%,and reduce power consumption by 39.20%.

Finally, the paper has completed the design of floating-point multiply-add integration system architecture in the basis of floating-point multiply-add components.The verification results showed that floating-point multiply-add fused structure effectively reduced the latency.

**Key words:** floating-point multiply-add fused structure;leading-one prediction；a three-operand;VerilogHDL

# 目　　录

# 第一章　绪论

## 1.1 国内外研究概况

具有高性能的浮点乘加部件在研究高性能CPU的关键技术中占有很重要的意义，目前在国内外已经取得了在性能上和在关键路径延时上的优化技术。本节首先对基本的浮点乘法器和浮点加法器进行简单的说明，最后对浮点乘加融合的结构和国内外的研究现状做出简要介绍。

### 1.1.1 浮点乘法器和浮点加法器

由于浮点数的数据格式以及加减乘除等运算标准已经在IEEE754-854标准中给出明确说明，所以在设计浮点运算的处理器时要支持IEEE754-854标准。

为了完成浮点数据的加法，在设计加法器时需要做的处理工作包括对尾数的加法操作，对阶移位以及规格化处理，求补码，前导1预测以及最后的舍入处理。由此以来运算速度会比进行整数加法的运算速度要慢很多，因此提高浮点加法器的运算时间对于不含浮点乘加融合结构的浮点单元具有很重要的意义[1-2]。

Sweeney通过研究浮点运算中浮点加法的结构和布局，研究得出浮点加法的运算在浮点运算中频率最高的结论。通过研究 IBM360/91 的浮点处理器的原理和结构，Waser 和 Flynn 又已有浮点运算的基础上新提出了一种运算的改进方法。而第一次提出采用双通路加法器设计的人是 Farmwald，这种算法至今在研究浮点加法中仍有很重要的指导意义[3-6]。

Gosling 又将现今已经在浮点运算体系中被采用的浮点加法结构进行了总结。这些结构现在多数已成为研究浮点加法的标准结构。Sit 和 Benschneider 第一次设计实现了双通路浮点加法器的结构，且与IEEE754运算标准是兼容的。至今依然有很多人对双通路浮点加法器有浓厚的兴趣，他们仍在继续深入研究着[1]。

通过研究已有的技术成果，人们在浮点运算的基础之上，为了提高浮点加法的运算速度，相继提出了许多优化的技术，主要包括：(1)设计 far 和 close 两条通路用来处理 A,B 浮点数之间指数差的比较。(2)浮点数的对阶移位与计算粘贴位同时处理；(3)通过设计前导0预测算法，将预测尾数结果中的前导0个数的与求和操作同时进行，

以此来提高运算速度。(4)通过采用复合加法器结构使舍入与求和步骤合并进行，并采用此结构完成快速转换的操作[5-7]。(5)通过采用注入算法将舍入模式转换为零舍入模式，从而提高运算速度。

浮点乘法结构包括进位传播加法，部分积产生与压缩、舍入处理等主要操作。且多数乘法结构会采用 Booth 这种扫描算法，这一点则与整数的乘法结构类似。传统的 Booth 算法的结构是利用乘数的每一位来产生一项部分积，那么此时实现 16*16 的操作，则最后需要做 16 项部分积相加的处理，但是若采用优化后的 Booth 算法，若要实现 16*16 的操作，则只需要做 9 项部分积相加的处理，减少了累加的次数，因为改进后的编码是利用乘数的连续三位来产生一项部分积，因为它可以表示出被乘数的 1 倍，2 倍，负 1 倍，负 2 倍或 0[8]，所以此部分积要经过部分积选择器进行确定。因此现在多数浮点乘法器都在采用优化后的 Booth 算法来提升运算效率。

前面已经提到过，在运用 Booth 算法之后，要完成乘法操作，还得对部分积进行累加，有人已经提出过设计进位存储加法器 CSA(carry-save-adder)，在设计上采用线性阵列和对数树型等结构，但是设计的电路比较繁杂，且经过性能分析后的路径延时大，性能低。因此 Wallace 在此基础上又提出了一种优化的 CSA 结构（3:2CSA），即 3 输入，2 输出,部分积经过 3:2CSA 结构，产生两个输出，而不是传统意义上运用行波进位来完成累加。研究表明，优化后的 CSA 结构显著提升了性能，减少了关键路径延时。

为了能使优化效果更明显，有人提出了并行计数器和压缩树的概念，计数器是通过组合逻辑设计目的是显示数据中 1 的个数，现在多被采用的有(3,2)、(5,3)计数器，表示的输入中有 2 个 1，输入中有 3 个 1，而（r，s）压缩树代表通过由上一级输入为 r 个，输出到下一级的信号有 s 个[2]。这样（4:2）压缩树相当于（5,3）计数器，因为第五个输入来自于同级前一位的第三个输出，因此从某种意义上来说压缩树是可以表示在部分积中计数器的功能，因此我们可以利用由计数器对来设计的 4 输入，2 输出的压缩树结构来实现 64*64 乘法器的设计。

乘法器根据部分积压缩运算的结构可划分三类：迭代乘法器、线性阵列乘法器和树形乘法器。近些年来人们设计出很多乘法器的优化结构，例如按数迭代结构，它是在按位迭代结构的基础上改进的，但是这种优化结构在很大程度上减少了完成运算所要进行的迭代次数，减少时间。也可以从硬件设计方面来减少延时的，例如，运用动态电路或者冗余码来优化电路结构，减少延时，在设计方面也得到了广泛的应用，但是在对功耗和速度方面有严格要求时采用最多的结构还是 Booth 算法编码和树型结构乘法器。

### 1.1.2 浮点乘加融合的发展

传统的不含浮点乘加融合结构的浮点运算单元, 若要完成 A×B+C 操作需要进行乘一次，舍入一次，求和，再舍入一次，这样就进行了两次舍入操作，如图 1.1 所示。在此基础上研究出来的浮点乘加融合是将 A×B+C 作为一个整体结构, 不进行中间舍入，这样就只进行最后一次舍入，提高了运算速度[1]，这样的结构也被广泛采用到低功耗和高速流水线设计中，如图 1.2 所示。

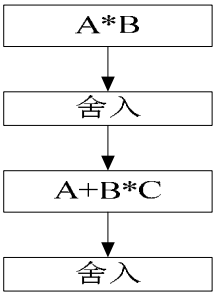



图 1.1 没有乘加融合的浮点运算

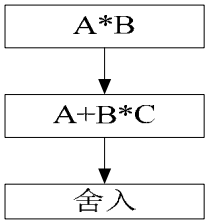



图 1.2 Montoye 和 Hokenek 提出的乘加融合运算

乘加融和运算跟不含乘加融合的浮点运算比较有以下方面的优点：

1)提高运算的速度和精度。

2)降低硬件成本，减少关键路径的延时，提高性能。

3)拥有乘加融合结构在实现其他较复杂的浮点运算指令更占有优势。

较之前的浮点乘加运算，容易实现其他浮点运算和整数乘加指令。

Montoye 和 Hokenek 提出了浮点乘加融合 MAF(multiply-add-fused)结构，这是最早被提出的结构，但是研究发现在舍入时会存在一定程度的延时，并且求和时的位数较大，比较复杂，效率有待改善。采用这种结构的处理器有 IBM 公司的 power3、PowerPC604eTM、PowerPC 603eTM 等[2]。它包括以下的计算步骤[2]：

1) 对 A，B，C 进行解码，B 与 C 尾数相乘，A 与 B,C 相乘的结果进行对阶。

2) 对阶后的加数 A 与 A，B 相乘后的结果 c 和 sum 同时进入 3:2CSA 进行相加，产生两个输出结果。

3) 使上述输出结果做相加处理，并且与 LZA 同时进行，得到规格化需要的移位数以及结果的符号位。

4) 把输出结果处理成标准化格式。

5) 利用得到的粘贴位，将 B×C+A 结果做舍入处理。

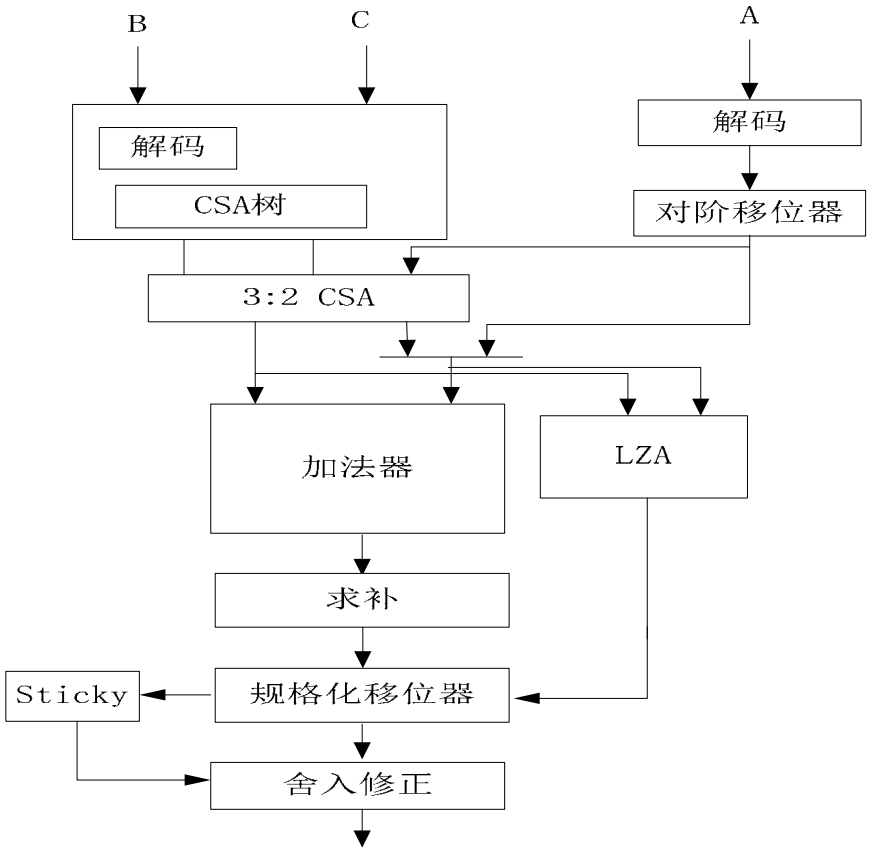



图 1.3　传统的乘加融合 MAF 结构

为了解决传统 MAF 结构的缺点，降低其延时，T.Lang 设计了一种低延时浮点乘加融合结构。这种优化后的乘加融合结构在两方面有所突破，一方面是合并舍入和加法操作，另一方面在加法之前进行规格化处理。在一定程度上降低了延时，提高运算能力，目前大部分通用处理器和科学研究均采用这种结构，这种低延迟 MAF 的特点是：

1) 采用复合加法器得到 sum 和 sum+1，设计选择器，判断 sum 的最高有效位和舍入位，两者之间选择一个作为舍入结果，最后进行后规格化操作。

2) 将 LZA 与规格化移位同时进行。利用 LZA 前导 1 预测得到首 1 位置，从而得到规格化移位量，同时进行规格化处理，两者并行，减少运算延时。

3）利用时间差值进行树型加法器的逻辑实现。当 LZA 预测得到结果的首 1 位置与进行规格化移位两者之间是存在一定时间差的，为了充分提升效率，可以在这个时间差值进行树型加法器的部分逻辑实现。

改进之后的低延迟的浮点乘加融合结构是同样存在缺陷的，其路径的延时与传统的 MAF 结构所差无几。当仅支持规格化数据输入时，那么采用浮点乘加融合的运算时间要比执行乘法操作的时间要多。

为了解决这一问题，T.Lang 和 J.D Bruguera 提出了双通路浮点乘加融合结构，是

在输入仅支持规格化数的浮点乘法又单独设计了一条通路，但是设计结构庞杂，硬件成本较高。纵古至今，国内外一直在对浮点乘加融合结构进行深入的探索和研究，并且在乘法器，加法器等各种环节上已经提出了很多优化的技术方法，包括 LZA 环节上也提出了各种预编码规则以及硬件实现，降低了关键路径延时，且各种优化技术也已经在 Intel、IBM 等大公司的通用处理器中得到了广泛的采用。而国内通常根据国外在著名刊物上所发表的结果进行半定制设计，其中西北工业大学、中科院以及国防科技大学都有人对高精度的科学计算加速方法以及高精度乘累加器，高性能全流水乘加部件等都进行了不同程度的探讨[3]。

所以，国内在浮点运算领域还有很大的发展空间，和国外的发展水平仍存在一定差距，且随着国内高性能 CPU 的快速发展，研究浮点乘加部件是针对浮点运算速度和精度的点睛之笔，对提高速度以及提升精度都具有重要的价值。

## 1.2 研究与实现意义

浮点乘加部件在满足 IEEE754-854 标准的条件下可完成浮点加、减、乘、除、平方根、求余数等操作。其功能强大，如若系统要求实现定点加/减，定点乘加，浮点逻辑等功能，可以通过增加相应的逻辑单元进行实现，其可塑性较强。

浮点输入为非规格化数的时候，Itaniu 有专门处理这种数的方法，即无软件协处理。但是由于 X 处理器不含中断机制以及无软件协处理，如若出现这种情况只能通过硬件结构来解决，所以针对 X 处理器只有规格化数输入才能进行浮点乘加融合操作[10]。

在设计浮点乘加部件时也要考虑常用的浮点指令，其必须增加一些例如指数调整指令和规格化指令等在实现各种算法中需要用到的，以此来解决查表不能解决非规格话数输入的问题。

我们之所以设计浮点乘加部件，目的是要减少关键路径的延时以及硬件资源的占用，但是除此之外，在设计时一定不要忽略计算的正确性和测试的充分性，否则设计将毫无意义，最后我们需要设计的是具有性能强，速度快，精度高，准确度高等浮点乘加部件。

之前已经详细的介绍过，我们可以采用很多技术来优化浮点乘加部件，基本包括减低关键路径的延时，或者在电路设计方面进行优化设计实现或者逻辑重组等方面。在电路设计方面，全定制设计和半定制设计都具有其优势，其中全定制设计具有高度的灵活性，其采用传输管逻辑和动态逻辑等半定制无法实现的功能，而且在设计中自动删除冗余逻辑来减小逻辑单元的面积，减少占用资源，实现优化，在延时和占用面积上都具有较大的优势。通过设计浮点乘加部件的结构以及算法，可以将此设计成可

复用 IP 核或者硬 IP 核，对高性能浮点系统中处理浮点运算具有很重要的作用。

## 1.3 论文研究内容

本文从研究背景出发，在已有技术的基础上，以降低延时和功耗为研究目的，把乘加融合这个结构为研究对象，通过设计结构中的三操作数的前导 1 预测算法，从而提高了浮点运算速度，减少关键路径的延时，并把乘加融合系统分为几个主要模块，包括解码模块，乘法以及加法模块，移位器模块，首 1 预测模块，标准化以及舍入模块等。其中乘加器中设计了乘法器结构、部分积求和，部分积选择器。在两操作数预测算法的基础上成功设计出三操作数的前导 1 预测算法以及完整硬件实现方案，通过与两操作数的前导 1 预测模块相比较，验证了三操作数算法的优势。最后，设计基于 FPGA 的高性能浮点乘累加器，通过仿真，在 Alter 公司的 DE2[11-14]实验板上实现。

## 1.4 论文主要贡献

1.深层次的剖析了浮点数的乘加融合运算，并将浮点乘加融合结构进行详细的介绍。

2.设计了三输入两输出的进位存储 CSA 加法器。

3.设计了三操作数的前导 1 预测算法，其算法结构中包括预编码规则的设计以及编码树的硬件结构实现，最后又通过 quartus II 仿真工具中测试，仿真结果表明设计的结构减少了关键路径的延时，降低了功耗，提高了运算速度，重点在第三章中详细介绍说明。

4. 完成了浮点乘加融合结构的整体设计实现，通过下载到 Altera 公司的 DE2 开发板上进行调试，完成综合仿真。

# 第二章　64 位浮点乘加融合体系结构

本论文研究的是浮点乘加部件，针对的是 64 位浮点数 A,B,C，处理对象是浮点数，即在处理过程中和处理整数运算的方法存在很大不同。在这里本章先介绍浮点运算的相关基础知识[2]，以及对 64 位浮点乘加融合体系结构中各个模块进行简单介绍。

## 2.1 浮点运算基础的介绍

下面将从浮点数据格式等浮点运算中的一些基本概念做简要介绍。

### 2.1.1 浮点数据格式

IEEE-754 标准定义了一些浮点运算的常用概念, 下面对这些概念作出简单的介绍[2]。

浮点数：包括三部分，分别是符号，有符号指数和尾数。

指数：表示浮点数中的二的整数幂。

零：为有符号的数值，分别是正零和负零，但两种形式在数值上为等价。

符号判定可由执行的操作和最后舍入模块中所采用的舍入模式决定。

NaN：为非数，且有两种讨论形式：1）被用作操作数时，表示非法操作发射信号；2）当进行算术分配时，会传输 NaN 信号，但不提示非法操作。

正无穷大：+∞为浮点格式中的最大的正实数。

负无穷大：-∞为浮点格式中最大的负实数。且和正无穷大一样均可以被表示为零尾数或者所允许的最大值。

浮点数格式为：

| 1 bit | M bits | N bits |
|---|---|---|
| Sign（符号） | Exponent(指数) | Mantissa(尾数) |

值= $(-1)^{sign}\times 1.mantissa\times 2^{expoent-bias}$，bias 为偏移量；其中，对于双精度 64 位的浮点数，IEEE-754 标准规定，M 为 11，N 为 52，即共占用 64 位的存储空间。

对于双精度数的具体值规定如下：

当 e=2047 且 f<>0 时，则不论 s 为何值，v=NaN ;

当 e=2047 且 f=0，s=0 时，则 v=—∞;

当 e=2047 且 f=0,s=1 时，则 v=+∞;

当 0<e<2047 时，则 $v=(-1)^s\times2^{-1022}\times0.f$(为非规格化的数)；

当 e=0 且 f=0 时，$v=(-1)^s\times0$,即为零。

当 e<2047 时，N 虽然为 52 位，但只是表示小数点之后的二进制位数，也就是说，假定 N 为 01011111000…，在二进制上其实是.01011111000…，因为 IEEE 标准上是有一位隐含位的，位于小数点左侧，但针对 64 位浮点数据格式是没有直接显示出来该位，但是偏置时数字段的值却表示了该值，所以将 64 位数据格式进行解码时应该是 52 位尾数加上一位隐含位供提供 53 位精度。

### 2.1.2 浮点运算中基本概念

**规格化：**

浮点数据经过运算得到的输出必须要进行规格化处理，由于在采用规格化之后数据还要进行舍入操作，那么产生的误差就会小于最低有效位数值的一半[3]。

**舍入：**

浮点数进行运算之后的结果要变成标准浮点数据格式，那么就要进行舍入操作。现在目前被广泛采用的舍入模式主要有四种，即零舍入，偶数舍入，正无穷舍和负无穷舍入四种模式。其中零舍入为把最低有效位之后的所有位舍掉。偶数舍入指的是舍入后的结果必须是偶数，例如，0.25 和 0.15 均舍为 0.2。正无穷舍入指的是当符号位为正且最低有效位后出现 1 时，在有效位后加 1，否则就要删去有效位后的所有位。负无穷舍入指的是当符号位为负，在有效位后面出现 1 时，则在有效位加 1，否则就要删去有效位后的所有位[3]。

**浮点运算的异常情况：**

在进行浮点运算时，有时可能会出现错误，统称为异常。在 IEEE-754 标准中定义了以下几种异常情况：

1）无效操作。这种情况有可能是多种情况，可以分为两个方面，一方面是因为操作数中存在非有限数。另一方面是因为结果无效。当操作数为非数、加减无穷、0 乘无穷、0/0 等这些无意义的操作被列为无效操作。或者当进行求余处理的时候，如果除数为非规格化的，或者被除数为无穷大的，或者是开平方根处理时，如果被开方数为小于零的数和非规格化数的，或者是在进行进制转换时发生的溢出现象。

2）上溢或者下溢。指结果太大或太小。

3）不精确。浮点数据和整型之间进行转化时可能会出现[3]。

## 2.2 浮点乘加融合系统结构

本文是以 64 位浮点数据为输入来实现浮点乘加融合体系结构 A+ B×C 的设计。总体结构包括操作数的解码，三个操作数的对阶，加数 A 的移位，乘加器，三操作数的前导 1 预测算法，以及最后的规格化和舍入。如图 2.1 所示为浮点乘加融合的系统结构。

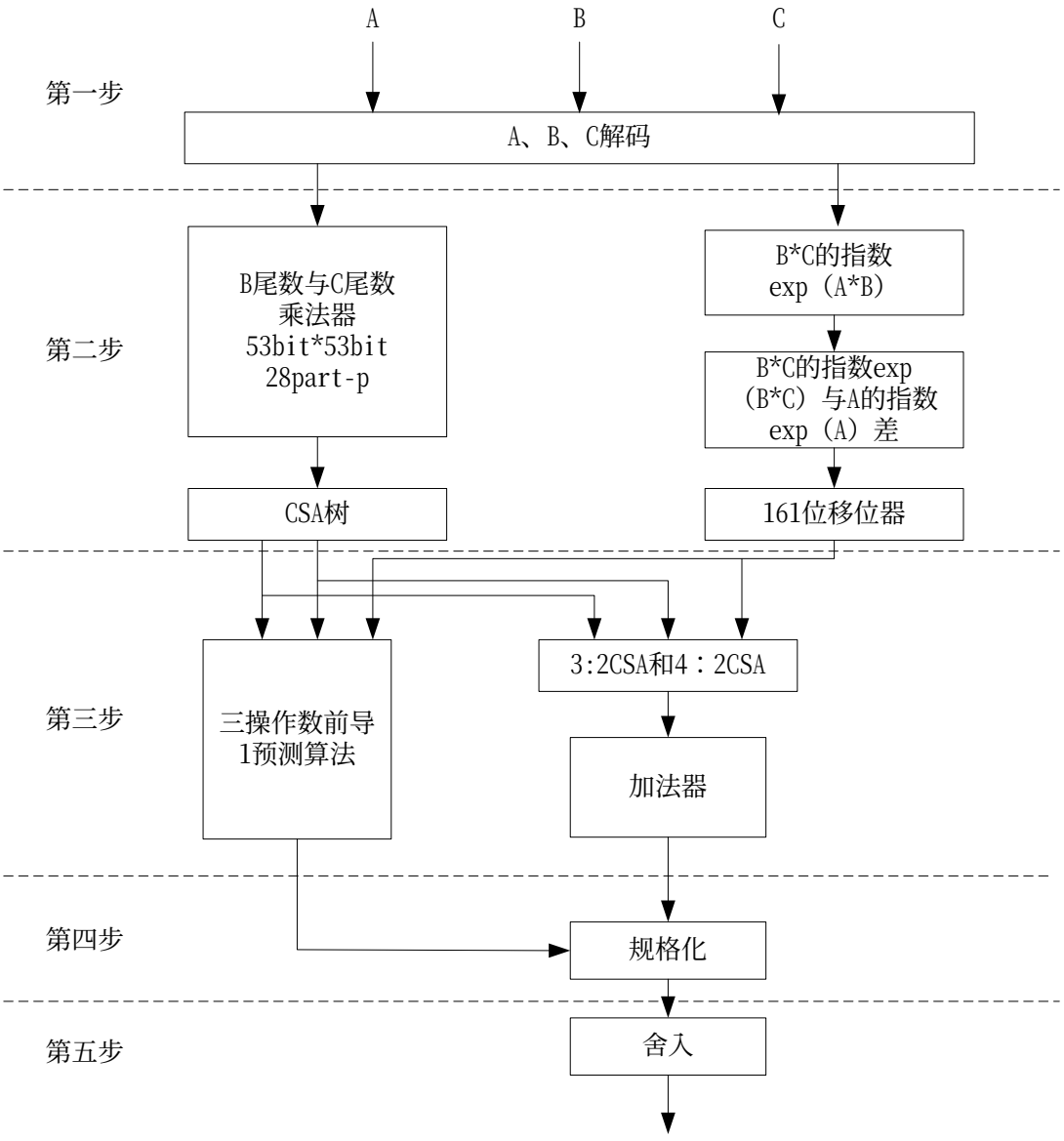



图 2.1　浮点乘加融合系统结构

1）首先对操作数 A,B,C 进行解码，分离出符号，指数和尾数。

2）并行完成 B×C 尾数的相乘以及加数 A 与 B×C 阶码对齐移位的操作，其中包括部分积选择器，进位存储加法器，部分积形成器等。

3)同步完成 B×C 与 A 相加以及前导 1 预测, 利用进位存储加法器实现相加操作,

并通过前导 1 预测提前得出运算结果的首 1 位置，以便进行规格化处理。

4）进行规格化和舍入操作。

## 2.3 本章小结

本章首先介绍了浮点运算的一些基础概念，并给出了浮点乘加融合体系结构，分析说明了体系结构的各个功能模块和本论文需要设计实现的内容。

# 第三章　三操作数前导 1 预测算法的设计与性能分析

前导 1 预测(Leading-One Prediction)算法是对浮点加法运算后的结果进行预测从而控制规格化移位的一种算法，其功能是在加法运算同时并行预测出运算结果中的首 1 的位置，从而得到移位量来进行规格化处理[8]。前导 1 预测模块在浮点加法处理器中占比重较大，其速度性能的高低，也是影响浮点加法模块性能的关键所在。在 Lang 等人设计的浮点乘加系统结构中，前导 1 预测电路处在整个系统结构中的重要环节，因此要想充分改善浮点乘加部件的性能，降低前导 1 预测电路所产生的延时是当务之急[7]。

传统的前导 1 预测算法是不能直接处理三个操作数，需要增加一个 3:2CSA 部件合并成两个操作数。相关文献表明，传统的前导 1 预测算法会增加硬件电路面积，增大延迟，故在本章中设计出了一种在 FPGA 平台上直接处理三个操作数的算法完整实现方案，提高运算效率。本章首先对前导 1 预测的结构以及三操作数的预测流程进行了简单了介绍，其次是对预编码规则的设计，重点是对预编码模块和编码树模块的整体结构设计实现，最后，采用 Quartus Ⅱ进行仿真验证，由仿真结果可得出设计的三操作数预测算法能够减少关键路径的延时和功耗。

## 3.1 前导 1 预测算法的结构

如图 3.1 所示为前导 1 预测的结构[6]，主要包含两个部分，一部分做基本的前导 1 预测，包括预编码和编码树（LOD），另一部分负责对前导 1 位置的纠错。首先预测部分，包括预编码和检测的编码树，其中在预编码模块，以预编码逻辑规则为基础，对 X,Y,Z 进行预编码，针对 X+Y-Z 的结果的正负分别采取两种不同的编码方式，且并行的进行正编码和负编码两种情况，由符号的结果得到后再进行选择，通过预编码规则生成一串“ 0 ”，“ 1 ”序列，然后前导 1 检测单元计算出“ 0 ”，“ 1 ”序列中第一个“ 1 ”的所在位置，并通过第二部分的纠正树产生 1 位纠正位来判断前导 1 位置是否需要进行纠正，纠正树与前导 1 检测模块并行执行，节省时间。

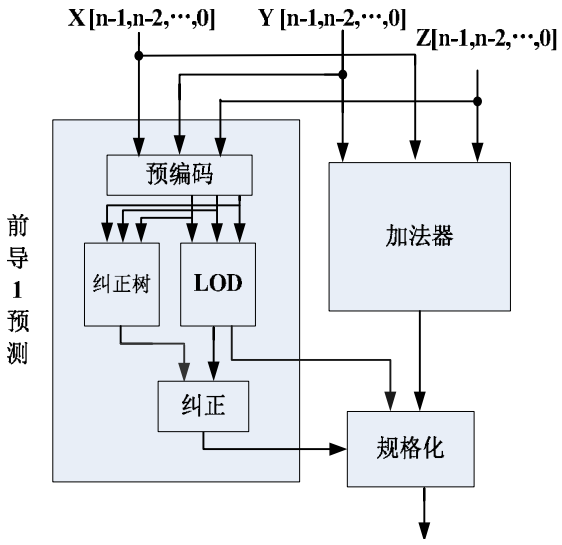
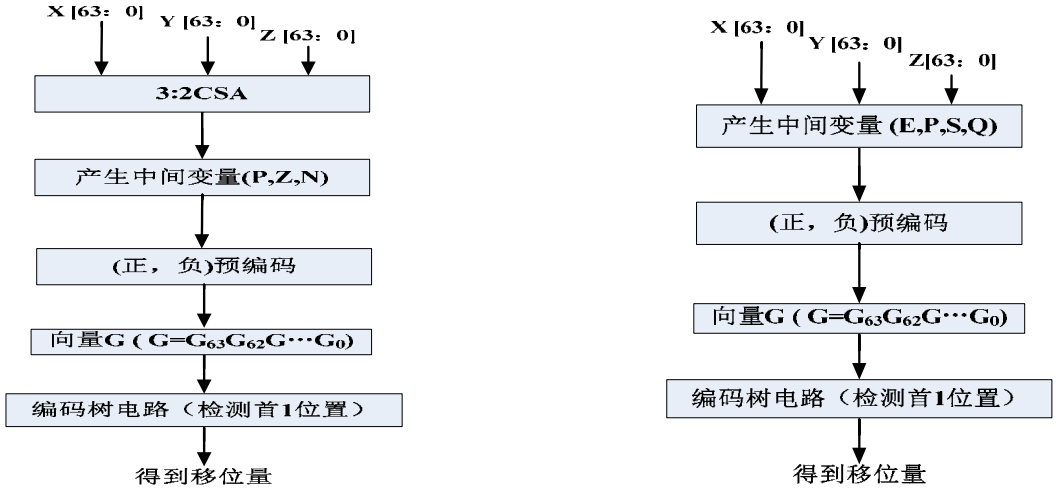



图 3.1　前导 1 预测的结构图

## 3.2 浮点乘加运算中传统预测算法

前导 1 预测模块是完成预测$|SUM_{B\times C}|+|CARRY_{B\times C}|-|A|$的首 1 位置的功能。那么如若提升效率，快速进行预测得到首 1 位置必须要优化前导 1 预测算法。

传统的前导 1 预测算法都是针对两个操作数来处理的，如图 3.2(a)所示的，其需要将三个操作数通过 3:2 进位存储加法器 CSA 合并为两个操作数, 再通过前导 1 预测算法的预编码规则从而生成向量$G$ ,那么向量$G$ 的首 1 位置也应与最终运算结果的首 1 位置相匹配，然后利用编码树电路对向量$G$ 的首 1 位置进行检测，对位置进行编码输出，得到移位量供规格化处理。




(a)　传统的前导 1 预测算法流程　　(b)　三操作数前导 1 预测算法流程

图 3.2　前导 1 预测算流程比较

在传统前导 1 预测算法中的预编码规则是把得到的$SUM_{B\times C}$ ,$C_{B\times C}$ ,和经过移位器

的操作数 A 通过 3:2 进位存储加法器 CSA 合并为两个操作数，在这里表示成 X,Y。$X = X_0X_1.....X_{n-1}, Y = Y_0Y_1.....Y_{n-1}, X_i, Y_i \in \{0,1\}$ 预编码规则如下。

通过对 X,Y 的每一位定义 $w_i = |x_i| - |y_i|, w_i \in \{-1,0,1\}$ 且在这里通过引入中间变量 P,Z,N 来分别表示 $w_i$ 中的-1,0,1.即可得出 $P_i, Z_i, N_i$ 的组合逻辑表达式。

$p_i = \overline{x_i} y_i$，$z_i = \overline{x_i \oplus y_i}$，$n_i = x_i \overline{y_i}$

从而可得到关于 $P_i, Z_i, N_i$ 的位串，然后对此位串中的每一位再进行编码，则为预编码串 $G$，$G$ 中的首 1 所在的位置便是前导 1 的位置。最后利用编码树对首 1 位置进行编码输出，其中预编码串 $G$ 的每一位定义如公式 3.1 所示。

$$G_i = p_{i-1}(z_i \overline{n_{i+1}} + n_i \overline{z_{i+1}}) + \overline{p_{i-1}}(n_i \overline{n_{i+1}} + z_i \overline{z_{i+1}}) \quad \text{.....(3.1)}$$

然后利用编码树电路来检测 $G$ 的首 1 位置。根据 $G$ 的首 1 位置就可得到规格化移位量。

## 3.3 三操作数前导 1 预测算法设计与实现

本节依据三操作数的预编码逻辑规则[4]，设计了预编码模块的硬件实现方案，并完成了相应各子模块的设计，最后通过 Quartus II 进行仿真。由仿真结果分析可得到，设计的预编码实现方案可以有效的完成预测。

### 3.3.1 三操作数的预编码规则

前导 1 预测是对浮点乘加完成后的结果进行提前预测，预测出小数点后的第一个 1 的位置，此预测过程要与加法运算并行完成，从而提高运行效率。

本论文设计的是三操作数的前导 1 预测算法，输入为三个操作数，分别为 $B \times C$ 运算得出的 $SUM_{B\times C}$ 和 $C_{B\times C}$，以及经过移位器的操作数 A。为了设计方便，在这里用 X,Y,Z 且均为 64 位，分别代表此三个操作数，$X = X_{63}X_{62}X_{61}.....X_0$，$Y = Y_{63}Y_{62}Y_{61}....Y_0$，$Z = Z_{63}Z_{62}Z_{61}.....Z_0$。首先对 X,Y,Z 进行预编码，针对 $X + Y - Z$ 的结果的正负分别采取两种不同的编码方式，且并行的进行正编码和负编码两种情况，由符号的结果得到后再进行选择，通过预编码部分得到一个 64 位的向量 $G$，然后利用编码树电路来对这个向量 $G$ 的首 1 位置进行检测和编码，从而得到了这个首 1 位置的二进制编码。

三操作数预编码规则具有正编码和负编码两种编码模式，根据结果的符号来判断正负再进行选择，且带有最多一位的误差。$X_i, Y_i, Z_i \in \{0,1\}$，令 $R_i = X_i + Y_i - Z_i \in \{-1,0,1,2\}$，然后对 $R_i$ 可能出现的结果逐一研究和分析[4]，探讨算法规则。

**1.** $R > 0$ **即为 R 为 1 或 2**

(1)R 以 $0^k2$ 开头，R=$0^k2$(f),即(f)表示-1,0,1,2 中任意的位串。

**①R=$0^k2$(f)**

若(f)为非负数(0,1,2)， $R=\underbrace{000....0}_{k}2$(f),即可举例 X=$0^k1$(f)， Y=$0^k1$(f)， Z=$0^{k+1}$(f),则 X+Y-Z 的首 1 位置会进位， $R=\underbrace{000....1}_{k}0$(f), 则前导 1 的位置就在第 k 位。

若(f)为负数( -1)， $R=\underbrace{000....0}_{k}2$(f),可举例 X=$0^k10$(f)， Y=$0^k10$(f)， Z=$0^{k+1}1$(f),因为(f)的负数会向前借位， $R=\underbrace{000....0}_{k}11$(f),则前导 1 的位置就在第 k+1 位。

无论(f)在这里是非负数或这是负数，总假设前导 1 的位置在第 k 位，进而可得到首 1 的位置在第 i 位的预测规则为：$R_iR_{i+1}=0_i2_{i+1}$。

(2)R 以 $0^k1$ 开头，R=$0^k1$(f), (f)表示-1,0,1,2 中任意的位串。

**①R=$0^k12$(f)**

若(f)为非负数(0,1,2)，即可举例 X=$0^k11$(f)， Y=$0^{k+1}1$(f)， Z=$0^{k+2}$(f),则 X+Y-Z 的首 1 位置会进位， $R=\underbrace{000....1}_{k}00$(f), 则前导 1 的位置就在第 k 位。

若(f)为负数( -1)，即可举例 X=$0^k110$(f)， Y=$0^{k+1}10$(f)， Z=$0^{k+1}1$(f),因为(f)的负数会向前借位， $R=\underbrace{000....0}_{k}111$(f),则前导 1 的位置就在第 k+1 位。

无论(f)在这里是非负数或这是负数，总假设前导 1 的位置在第 k 位，进而可得到首 1 的位置在第 i 位的预测规则为：$R_{i+1}R_{i+2}=1_{i+1}2_{i+2}$。

**②R=$0^k10$(f)**

若(f)为非负数(0,1,2)，即可举例 X=$0^k101$(f)， Y=$0^{k+2}1$(f)， Z=$0^{k+3}$(f),则 X+Y-Z 的首 1 位置会进位， $R=\underbrace{000....0}_{k}11$(f), 则前导 1 的位置就在第 k+1 位。

若(f)为负数( -1)，即可举例 X=$0^k100$(f)， Y=$0^{k+3}$(f)， Z=$0^{k+2}1$(f),因为(f)的负数会向前借位， $R=\underbrace{000....0}_{k}011$(f),则前导 1 的位置就在第 k+2 位。

无论(f)在这里是非负数或者是负数，总假设前导 1 的位置在第 k+1 位，进而可得到首 1 的位置在第 i 位的预测规则为：$R_{i+1}R_{i+2}=1_i0_{i+1}$。

**③R=$0^k11$(f)**

R=$0^k11$(f)=$0^k11$(0， -1， 1， 2)(f) 。(0,-1,1,2)表示这一位可以为 0 或-1，或者是 1 或 2。

若(0,-1,1,2)这一位为 1，则 R 可以写成 $0^k11^m$(-1,0,2)(f)。

若(-1,0,2)这一位为 0 或-1，则前导 1 的位置在第 k+1 位。

若(-1,0,2)这一位为 2，且(f)为非负数，则前导 1 的位置在第 k 位；若(f)为负，则

前导 1 的位置在第 k+1 位。

即假设前导 1 的位置在第 k 位，进而可得到首 1 位置在第 i 位的预测规则：$R_{i+1}R_{i+2}=1_{i+1}1_{i+2}$。

**④R=$0^k$1(-1)(f)**

同上述 R=$0^k$11(f)的情况，R=$0^k$1-1(f)= $0^k$1-1$^m$(f)= $0^k$1-1 $^m$(0,1,2)(f)。(0,1,2)表示这一位可以为 0，或者为 1 或者为 2。

若(0,1,2)这一位为 0，即 R= $0^k$1(-1) $^m$0(f) 。(f)为正，则前导 1 的位置在第 k+m+1 位；若(f)为负，则前导 1 的位置在第 k+m+2 位。

假设前导 1 的位置在第 k+m+1 位，则首 1 的位置在第 i 位的预测规则为：$R_{i+1}R_{i+2}=-1_i0_{i+1}$。

若(0,1,2)这一位为 1，即 R= $0^k$1(-1) $^m$1(f) 。若(f)为非正数（-1 或 0），则前导 1 的位置在第 k+m+1 位；若(f)为正数（为 2），则前导 1 的位置在第 k+m 位。

假设首 1 位置位于第 k+m 位，则首 1 的位置在第 i 位的预测规则为：$R_{i+1}R_{i+2}=-1_{i+1}1_{i+2}$。

若(0,1,2)这一位为 2，即 R= $0^k$1(-1) $^m$2(f)。若(f)为非负数，则首 1 位置位于第 k+m 位；若(f)为负数，则前导 1 的位置在第 k+m+1 位。假设前导 1 的位置在第 k+m 位，则首 1 的位置在第 i 位的预测规则为：$R_{i+1}R_{i+2}=-1_{i+1}2_{i+2}$。

表 3.1　W>0 首 1 位置预测规则

| R=X+Y-Z | 首 1 的位置 | 前导 1 在第 i 位的预测规则 |
|---|---|---|
| $0^k$2 (f) | k 或 k+1 | $R_iR_{i+1}=0_i2_{i+1}$ |
| $0^k$12 (f) | k 或 k+1 | $R_{i+1}R_{i+2}=1_{i+1}2_{i+2}$ |
| $0^k$10 (f) | k+1 或 k+2 | $R_iR_{i+1}=1_i0_{i+1}$ |
| $0^k11^m$(-1,0,2)(f) | k 或 k+1 | $R_{i+1}R_{i+2}=1_{i+1}1_{i+2}$ |
| $0^k$1(-1) $^m$0(f) | k+m+1 或 k+m+2 | $R_iR_{i+1}=(-1)_i0_{i+1}$ |
| $0^k$1(-1) $^m$1(f) | k+m 或 k+m+1 | $R_{i+1}R_{i+2}=(-1)_{i+1}1_{i+2}$ |
| $0^k$1(-1) $^m$2(f) | k+m 或 k+m+1 | $R_{i+1}R_{i+2}=(-1)_{i+1}2_{i+2}$ |

**2.** $R<0$

即 X+Y-Z<0 时，即 Q=Z-X-Y$\in\{0,-1,-2,1\}$，因为 Q>0，则 Q 的最高有效位为 1，即 Q=$0^k$1(f)。

**①Q=$0^k$11(f)**

若(f)为(0,1，-1)，则前导 1 的位置在第 k+1 位。

若 Q= $0^k11(-2)^m(f)$，m=1 时，前导 1 的位置在第 k+1 位。m>1 时，则前导 1 的位置在第 k+2 位。假设首 1 的位置在第 k+1 位，则首 1 位置位于第 i 位的预测规则为：$Q_iQ_{i+1}=1_i1_{i+1}$。

**②Q=$0^k10(f)$**，分为两种情况：

若(f)为（-1,0,1）,则前导 1 的位置在第 k+1 或 k+2 位，则首 1 位置位于第 i 位的预测规则为：$Q_iQ_{i+1}=1_i0_{i+1}$。

若(f)为-2, 即 Q=$0^k10(-2)^m(f)$,当 m=1 时, (f)为非负数时, 即首 1 位置位于在第 k+m+1 位；(f)为负数(-1)时，即前导 1 的位置在第 k+m+2 位。假设前导 1 的位置在第 k+m+1 位，则前导 1 在第 i 位的预测规则为：$Q_iQ_{i+1}=0_i(-2)_{i+1}$。

当 m>1 时，(f)为非负数，前导 1 的位置在第 k+m+1 位；

当(f)为负数(-1)时, 首 1 位置位于第 k+m+2 位。假设前导 1 的位置在第 k+m+1 位, 则首 1 位置在第 i 位的预测规则为：$Q_iQ_{i+1}=(-2)_i(-2)_{i+1}$。

**③Q=$0^k1(-1)^m(f)$**

当(f)为(1,0)时，Q=$0^k1(-1)^m(1,0)(f)$，与上述情况相同，不再赘述。

当(f)为-2 时，Q=$0^k1(-1)^m(-2)(f)$，与上述情况相同，不再赘述。

**④Q=$0^k1^m(-2)^m(f)$**

与上述情况相同，不再赘述。

表 3.2　W < 0 时前导 1 的位置

| R=X+Y-Z | 首 1 的位置 | 前导 1 在第 i 位的预测规则 |
|---|---|---|
| $0^k11(f)$ | k+1 或 k+2 | $R_iR_{i+1}=(-1)_i(-1)_{i+1}$ |
| $0^k10(-1,0,1)(f)$ | k+1 或 k+2 | $R_iR_{i+1}=(-1)_i0_{i+1}$ |
| $0^k10(-2)^m(f)$ | k+m+1 或 k+m+2 | $R_iR_{i+1}=0_i2_{i+1}$ 或 $2_i2_{i+1}$ |
| $0^k1(-1)^m(1,0)(f)$ | k+1 或 k+2 | $R_iR_{i+1}=(-1)_i0_{i+1}$ |
| $0^k1(-1)^m(-2)(f)$ | k+1 或 k+2 | $R_iR_{i+1}=(-1)_i0_{i+1}$ |
| $0^k1^m(-2)^m(f)$ | k+1 或 k+2 | $R_iR_{i+1}=(-1)_i0_{i+1}$ |

故综上所述，当 R≥0 时，前导 1 在第 i 位的条件：

$$G_i(pos)=0_i2_{i+1}+(1_i+(-1_i))0_{i+1}+(1_{i+1}+(-1_{i+1}))(1_{i+2}+2_{i+2}) \quad \text{……(3.2)}$$

当 R≤0 时，前导 1 在第 i 位的条件：

$$G_i(neg)=-1_i(-1_{i+1}+0_{i+1})+(0_i+2_i)2_{i+1} \quad \text{……(3.3)}$$

本文在这里引入中间变量 E，P，S，Q 来分别表示 $R_i$ 中的 0，-1，1，2，并把中间变量 E，P，S，Q 代入到 $G_i(pos),G_i(neg)$。如表 3.3 所示为生成规则。

表 3.3　中间变量的生成规则

| $X_i$ | $Y_i$ | $Z_i$ | $W_i$ | E,P,S,Q |
|---|---|---|---|---|
| 0 | 0 | 0 | | |
| 0 | 1 | 1 | 0 | E |
| 1 | 0 | 1 | | |
| 0 | 0 | 1 | -1 | P |
| 0 | 1 | 0 | | |
| 1 | 0 | 0 | 1 | S |
| 1 | 1 | 1 | | |
| 1 | 1 | 0 | 2 | Q |

从上述表中可得出 E,P.S,Q 的组合逻辑表达式：

$$E_i = \overline{X_i}\,\overline{Y_i}\,\overline{Z_i} + (X_i \oplus Y_i)Z_i \quad \text{.....(3.4)}$$

$$P_i = \overline{X_i}\,\overline{Y_i}\,Z_i \quad \text{.....(3.5)}$$

$$S_i = X_i Y_i Z_i + (X_i \oplus Y_i)\overline{Z_i} \quad \text{..... (3.6)}$$

$$Q_i = X_i Y_i \overline{Z_i} \quad \text{.....(3.7)}$$

把中间变量 E,P,S,Q 代入到$G_i(pos), G_i(neg)$中可得出：

$$G_i(pos) = E_i Q_{i+1} + (S_i + P_i)E_{i+1} + (S_{i+1} + P_{i+1})(S_{i+2} + Q_{i+2}) \quad \text{.....(3.8)}$$

$$G_i(neg) = P_i(P_{i+1} + E_{i+1}) + (E_i + Q_i)Q_{i+1} \quad \text{.....(3.9)}$$

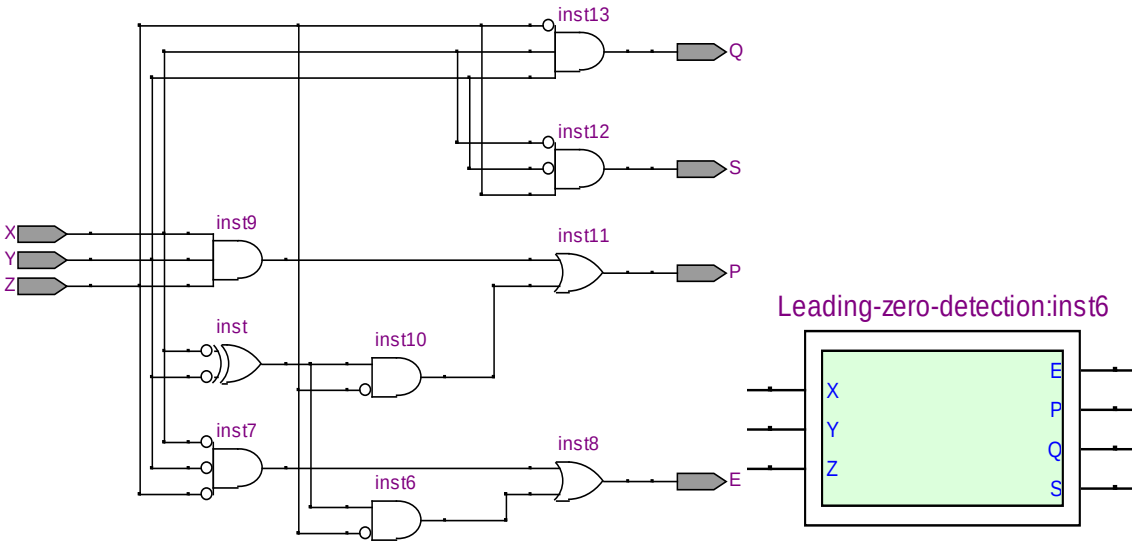
### 3.3.2 预编码规则的硬件实现方案

由上一小节得知，向量 G 的首 1 位置是我们预测的对象，只要仿真结果的首 1 位置与理论上 X，Y，Z 经过运算的结果的首 1 位置是一致的，即可说明设计的方案是切实可行的。

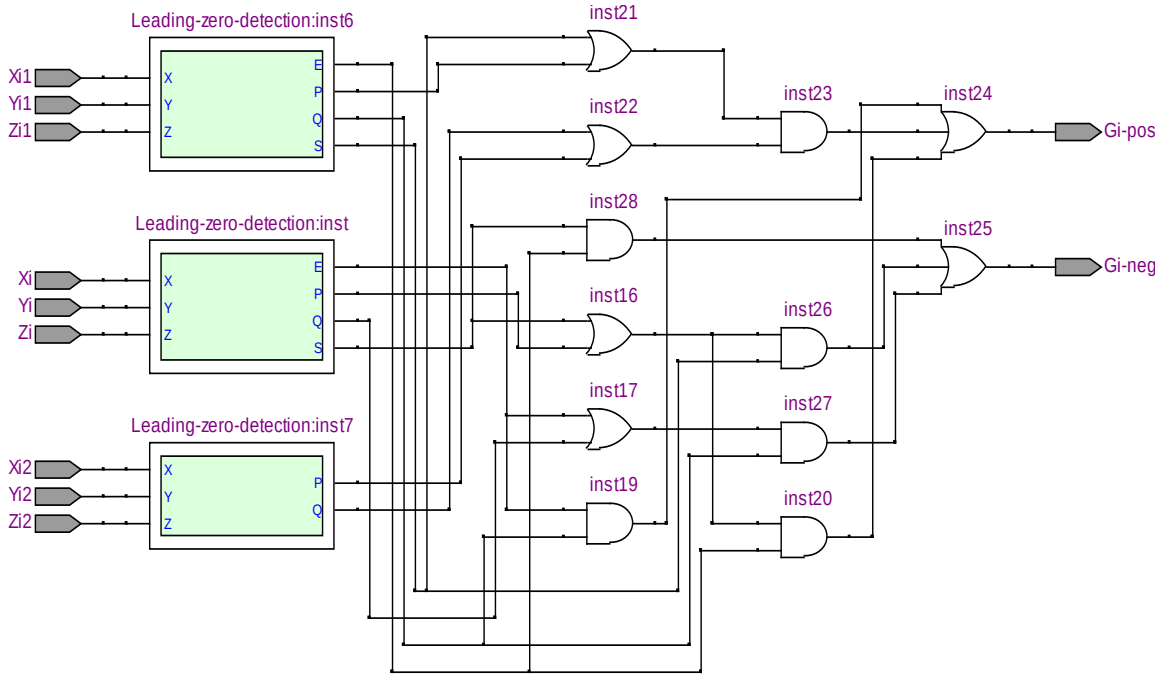
在这里，首先以 8 位为输入设计预编码子模块，从上述编码规则可看出$G_i$只与第 i，i+1 位，i+2 位有关，并由上述规则设计出子模块的组合逻辑电路，如图 3 所示为经 Quartus Ⅱ综合后的 RTL 结构图。

如图 3.3(a)所示，其中 Xi1 表示$X_{i+1}$(第 i+1 位)，Xi2 表示$X_{i+2}$(第 i+2 位)，同理 Y,Z 也是。Xi1~Xi3,Yi1~Yi3,Zi1~Zi3 为输入，生成中间变量$E,P,S,Q$（第一级），并经由组合逻辑电路得到输出$G_i(pos)$，$G_i(neg)$（第二级）如图 3.3(b)所示。最后通过调

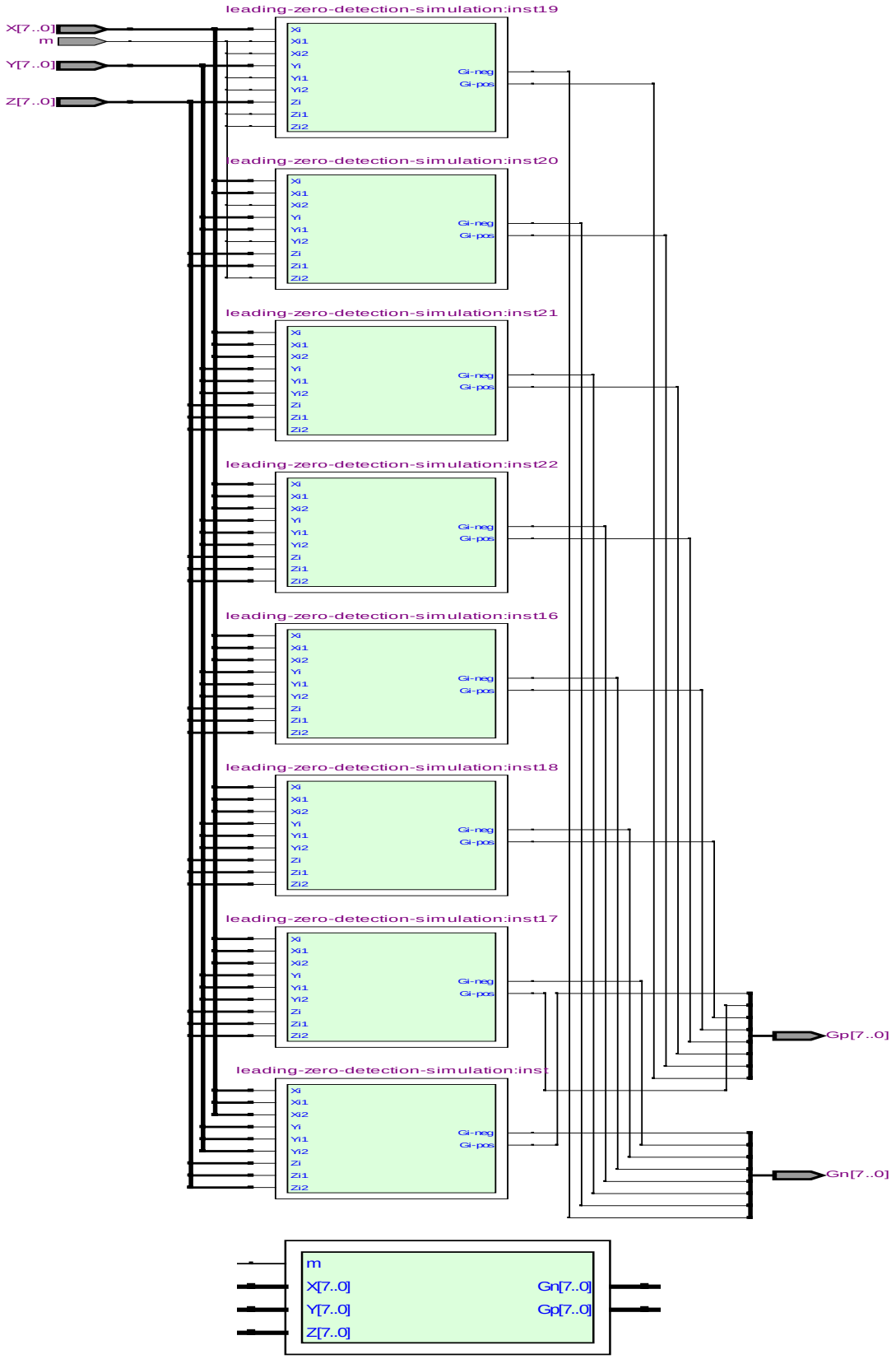
用子模块的方式，生成输出$G_p$[7..0]和$G_n$[7..0](第三级)，如图 3.3(c)所示。




(a) 第一级（RTL 结构图以及生成的子模块）




(b) 第二级(RTL 结构图)




(c)　第三级(RTL 结构图以及生成的子模块)

图 3.3　经 Quartus Ⅱ综合后的 RTL 结构图

仿真分析结果如表 3.4 所示，仿真输出如图 3.4 和图 3.5 所示。仿真的输入为 X,Y,Z 和 m ,其中 m 是一个低电位常量，输出为$G_P$,$G_n$。

表 3.4　仿真结果分析

| R≥0 | | | | R≤0 | | |
|---|---|---|---|---|---|---|
| 输入 | X | 01010010 | 10110101 | X | 00010100 | 00101010 |
| | Y | 00000010 | 01000110 | Y | 00001001 | 00010101 |
| | Z | 00001100 | 01010001 | Z | 01011101 | 11011011 |
| 处理公式 | X+Y-Z | | | X+Y-Z | | |
| 理论输出 | $G_p$ | **01**000100 | **1**0101010 | $G_n$ | **01**000000 | **1**0011100 |
| 仿真输出 | $G_p$ | **01**001000 | **1**0101010 | $G_n$ | **01**000000 | **1**0000000 |

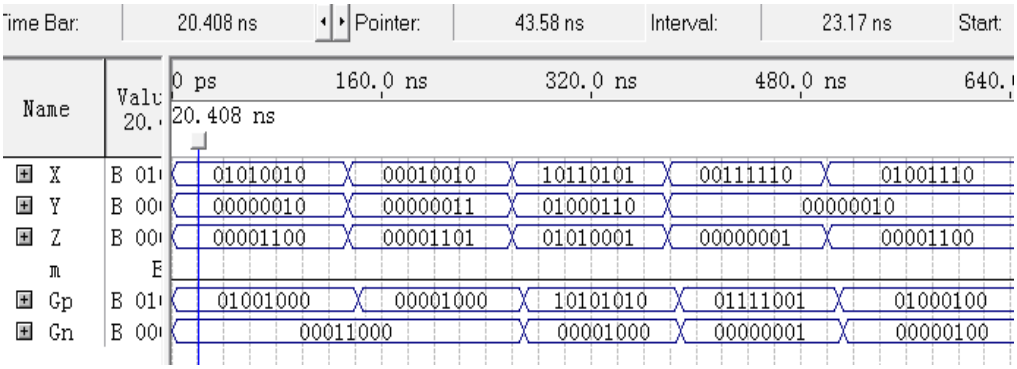



图 3.4　R>0 ($G_P$)仿真波形

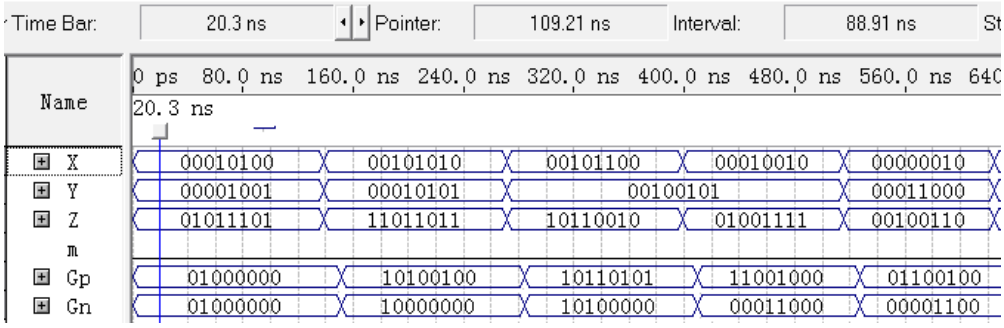



图 3.5　R<0 ($G_n$)仿真波形

由仿真结果分析得知，当 R≥0 和 R≤0 时，理论结果和仿真结果的首 1 位置是一致的，即方案是准确有效的。如图 3.4 所示，当 X=00111110, Y=00000010, Z=00000001 时，得到的结果应该是 00111111，即首 1 的位置在第二位，仿真得到的$G_P$为 01111001，首 1 的位置在第一位。因此，这种预编码规则是可能存在 1 位的误差的。

8 位的预编码部分已经仿真成功，综合编译使其生成子模块，即可对 64 位数据，按照从高到低的顺序，由八位一组进行设计，通过调用子模块，最后可得到一个 64 位的向量$G$。如图 3.6 所示为经过 Quartus Ⅱ综合后的 RTL 结构图，图 3.7 为相应的

仿真结果。由仿真结果可得出，此硬件设计方案可以正确，有效的完成前导 1 预测，实现预期目标。

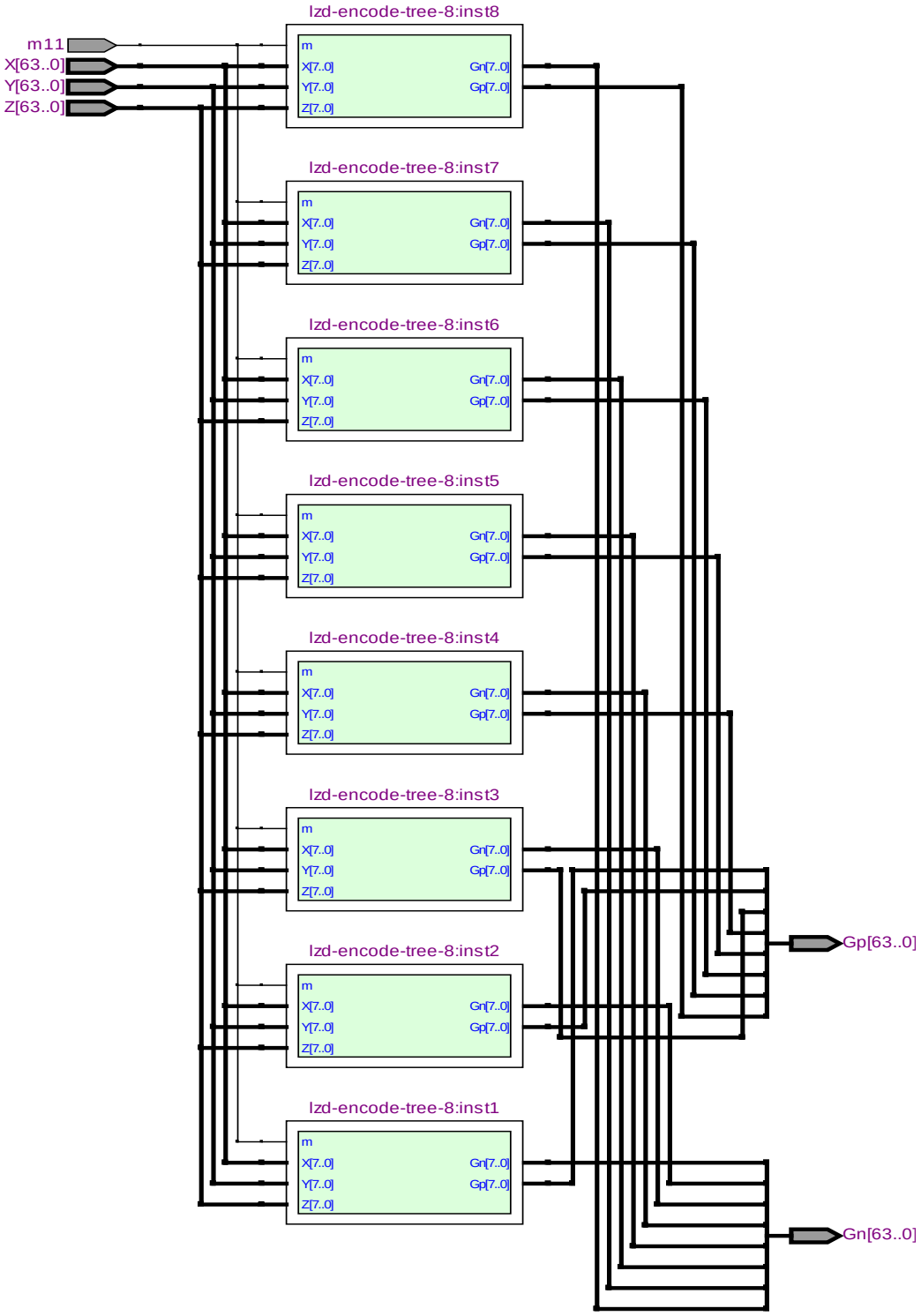



图 3.6　经 Quartus Ⅱ综合后的 RTL 结构

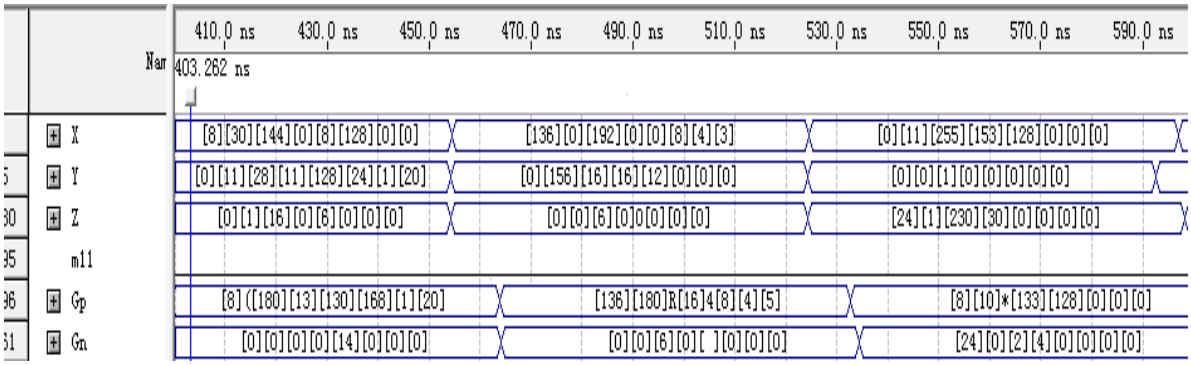



图 3.7　64 位的仿真结果

### 3.3.3 编码树电路的逻辑规则

在上一节中通过实现预编码模块得到 64 位的向量$G$,在$G$中首 1 出现的位置$i$就是前导 1 的位置。因此需要通过编码树电路来对$G$的首 1 位置进行检测输出，从而得到移位量来进行规格化处理[6]。

在这节中，本文首先设计出了基本单元—2 位的前导 1 检测结构(LZD2)，然后通过结构模块化对 4 位，8 位和 32 位进行设计，最后再通过对 64 位进行分组实现，且部分功能运用 Verilog 硬件描述语言进行编程，最后在 Quartus Ⅱ上仿真实现。

电路结构共有 5 级逻辑，其基本单元是 2 位的前导 1 检测结构（LZD2）。

在编码树中，以输入数据$G$的每 2 位输入作为一个基本组合（LZD2），产生 1 位二进制代码，如表 3.5 所示，其中 B[1:0]作为输入, V 表示输入中是否存在 1，为有效位。P 代表首 1 位置也是移位量的二进制编码。

表 3.5　LZD2 真值表

| B[1:0] | P | V |
|---|---|---|
| 00 | X | 0 |
| 01 | 1 | 1 |
| 10 | 0 | 1 |
| 11 | 0 | 1 |

可得出逻辑表达式如下：$P=\overline{B[1]}B[0]$,$V=B[1]+B[0]$。由 V 来决定其是否存在前导 1，若存在再判断首 1 的位置。

其 LOD2 仿真生成的子模块如图 3.8 所示。

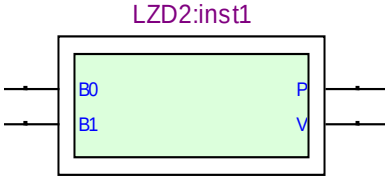


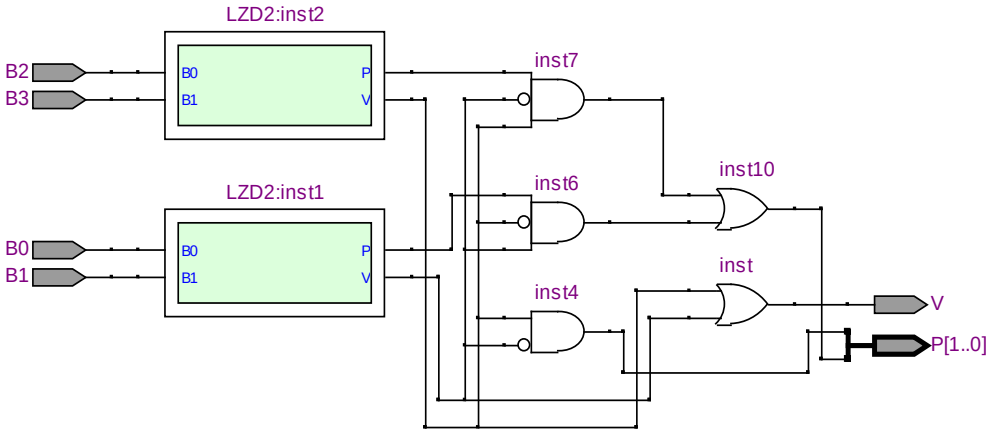

图 3.8　LZD2 子模块

若以输入数据$G$的每 4 位输入作为一个基本组合（LZD4），则产生 2 位二进制代码。如表 3.6 所示，其中 B[3:0]作为输入，$P_0$对应 B[3]B[2]两位的首 1 位置，$P_1$对应 B[1]B[0]两位的首 1 位置，$V_0$,$V_1$代表 $P_0$，$P_1$中是否存在有效 1，P 表示标准化需要的移位量，V 表示输入中是否存在 1，为有效位。P[1：0]代表首 1 位置也是移位量的二进制编码。

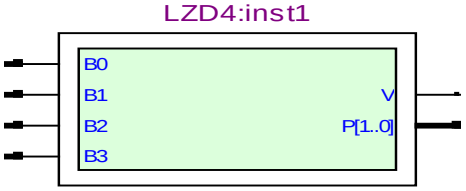
表 3.6　LZD4 真值表

| B[3:0] | $P_0$ | $P_1$ | $V_0$ | $V_1$ | P | V | P[1:0] |
|---|---|---|---|---|---|---|---|
| 0000 | * | * | 0 | 0 | X | 0 | ** |
| 0001 | * | 1 | 0 | 1 | 3 | 1 | 11 |
| 001X | * | 0 | 0 | 1 | 2 | 1 | 10 |
| 01XX | 1 | * | 1 | * | 1 | 1 | 01 |
| 1XXX | 0 | * | 1 | * | 0 | 1 | 00 |

表中的*号代表不存在 1 或者不定态，不影响对首 1 位置的检测，由上表可发现，$V=V_0+V_1$，$P[1]=\overline{V_0}V_1$，$P[0]=P_0V_0\overline{V_1}+P_1\overline{V_0}V_1$。$P_0$和 $P_1$中间变量可通过调用 LZD2 子模来实现，由此我们可设计出 LZD4 的逻辑电路结构，RTL 级电路图和生成的子模块如图 3.9 所示，仿真结果如图 3.10 所示。从仿真结果可得出，输入为 1011，输出 P=00；输入为 0111，输出 P=01；通过仿真，结果正确。




(a) LZD4　RTL 级电路图




(b)　生成的子模块

图 3.9　RTL 级电路图和子模块

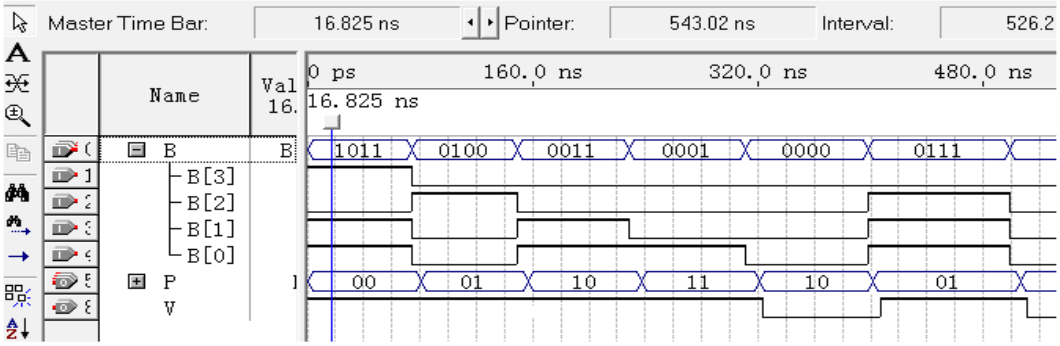
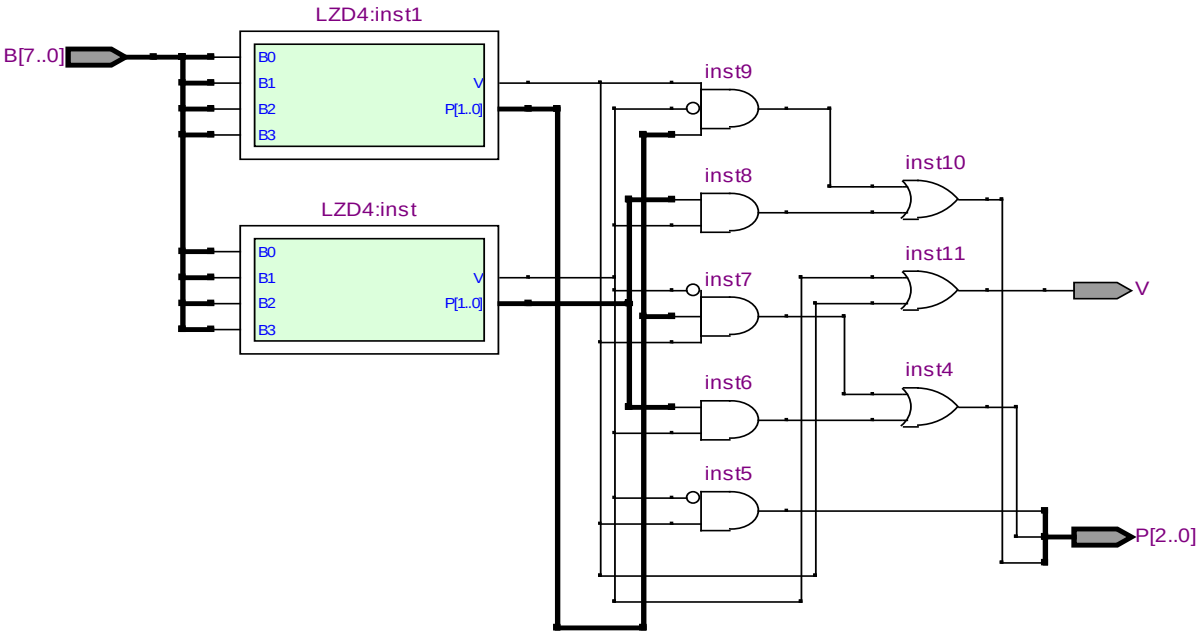
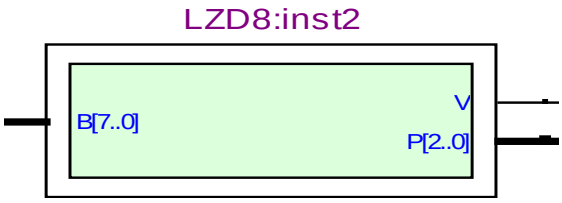



图 3.10　输入为 4 位的仿真结果

我们由上一节设计出编码树电路的基本模块(LZD4)，那么两个 LZD4 单元的输出信号就可作为一个 8 位编码树的输入信号，输出则产生一个 3 位的二进制代码，依据 LZD4 的编码树逻辑规则，即可得到 8 位的编码树组合逻辑表达式：$V = V_0 + V_1$，$P[1] = P_0[1]V_0 + P_1[1]V_1\overline{V_0}$，$P[0] = P_0[0]V_0 + P_1[0]\overline{V_0}V_1$。这样就可以设计出输入是 8 位的子模块(LZD8)，如图 3.11 所示。




(a)　8 位(LZD8)的 RTL 结构




(b)　生成子模块

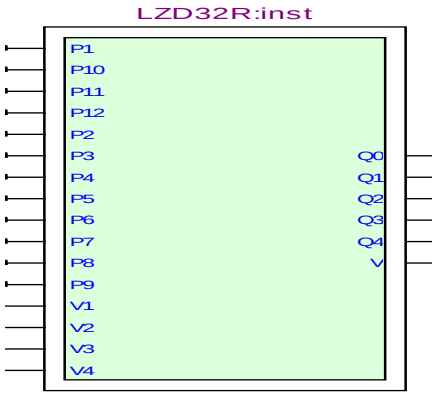
图 3.11　LZD8 结构图

若输入数据为 32 位，结果就要产生一个 5 位的二进制代码来表示输入数据的首 1

位置。通过对 32 位数据进行 8 位一分组，调用 4 个 LZD8 子模块，且本文在这里设计了一个转化模块LZD32R, 此模块的功能是将4个LZD8子模块输出的中间变量P[2：0]转化为 X+Y-Z 的首 1 位置二进制编码 LZC[4：0]，若 LZC[4：0]=01001，则首 1 位置在第 9 位，标准化需要的移位量是 9。在这里子模块 LZD32R 我们采用 Verilog 语言编程实现，程序描述如图 3.12 所示（在这里输出 LZC 用 Q 来表示）。总 RTL 结构图如图 3.13 所示。

```
module LZD32R(V1,V2,V3,V4,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,Q1,Q2,Q3,Q4,Q0,V);
input V1,V2,V3,V4;
input P1, P2, P3;
input P4, P5, P6;
input P7, P8, P9;
input P10, P11, P12;
output reg Q0, Q1, Q2, Q3, Q4;
output reg V;
  always @(V1,V2,V3,V4,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,P11,P12,Q1,Q2,Q3,Q4,Q0)
       begin
          V=V1|V2|V3|V4;
        if(V==1)
         begin
             if(V1==1) begin Q4<=0; Q3<=0;Q2<=P3;Q1<=P2;Q0<=P1;end
                 else if (V2==1)begin Q4<=0;Q3<=1;Q2<=P6;Q1<=P5;Q0<=P4;end
                   else if (V3==1) begin Q4<=1;Q3<=0;Q2<=P9;Q1<=P8;Q0<=P7;end
                   else    begin Q4<=1;Q3<=1;Q2<=P12;Q1<=P11;Q0<=P10;end
            end
        end
endmodule
```

(a)　LZD32R 程序描述




(b)　生成的 LZD32R 子模块

图 3.12　LZD32R 程序描述和生成的子模块

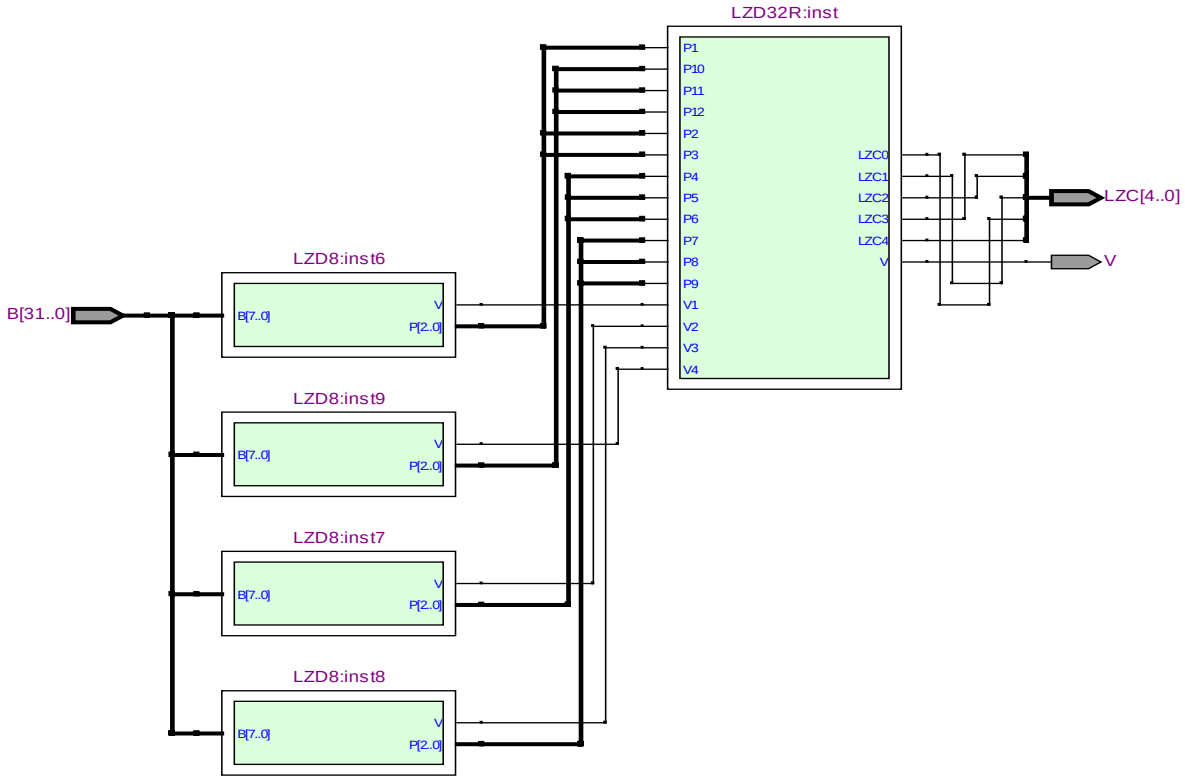
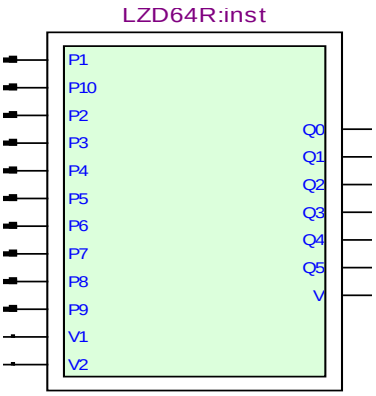



图 3.13　输入为 32 位时的 RTL 电路图

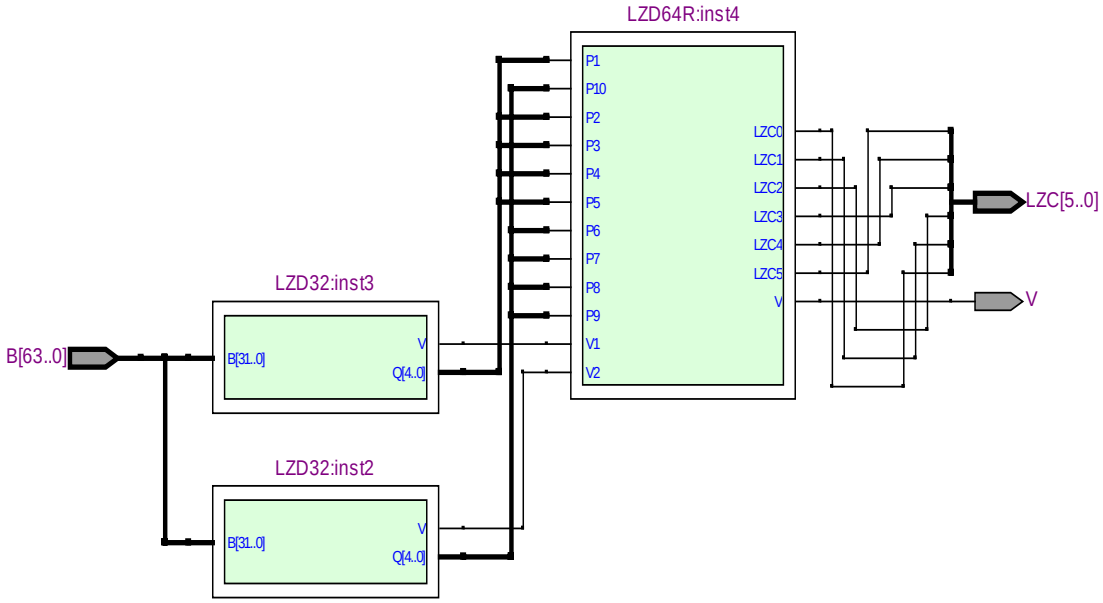
通过上述电路设计，综合编译生成 32 位子模块 LZD32，再调用两个 LZD32 子模块，同理用 Verilog 语言编写转换模块 LZD64R，由此我们可得到六位输出 LZC[5：0]来表示 64 位输入数据的首 1 位置，电路设计图如图 3.14(c)所示。LZD64R 的程序代码如图 3.14(a)所示。通过编写代码编译成功可生成子模块，如图 3.14(b)所示。

```
module LZD64R(V1,V2,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,Q0,Q1,Q2,Q3,Q4,Q5,V);
input V1,V2;
input P1,P2,P3,P4,P5,P6,P7,P8,P9,P10;
output reg Q0,Q1,Q2,Q3,Q4,Q5;
output reg V;
 always @(V1,V2,P1,P2,P3,P4,P5,P6,P7,P8,P9,P10,Q0,Q1,Q2,Q3,Q4,Q5)
    begin
     V=V1|V2;
        if(V==1)
            begin
              if(V1==1) begin Q5<=0;Q4<=P5;Q3<=P4;Q2<=P3; Q1<=P2;Q0<=P1; end
              else if (V2==1) begin Q5<=1;Q4<=P10;Q3<=P9;Q2<=P8;Q1<=P7;Q0<=P6; end
           end
       end
endmodule
```

(a)　LZD64R 设计程序




(b)　生成的子模块 LZD64R




(c)　32 位的编码树电路图

图 3.14　经 Quartus Ⅱ综合后的 RTL 结构图

## 3.4 仿真验证

以下均是 Quartus Ⅱ时序仿真的结果，经过和理论结果的对照，证明仿真的结果是完全正确的。表 3.7 为仿真结果分析。仿真输出如图 3.15,3.16,3.17 和 3.18 所示。

表 3.7　仿真结果分析

| 输入 | 8 位 | 32 位 | 64 位 | 64 位 |
|---|---|---|---|---|
| | 01000110 | [0][5][252][16] | [0][0][0][0][14][0][192][224] | [1][192][0][0][3][125][48] |
| 理论输出 | 001 | 01101 | 100100 | 000111 |
| 仿真输出 LZC | 001 | 01101 | 100100 | 000111 |

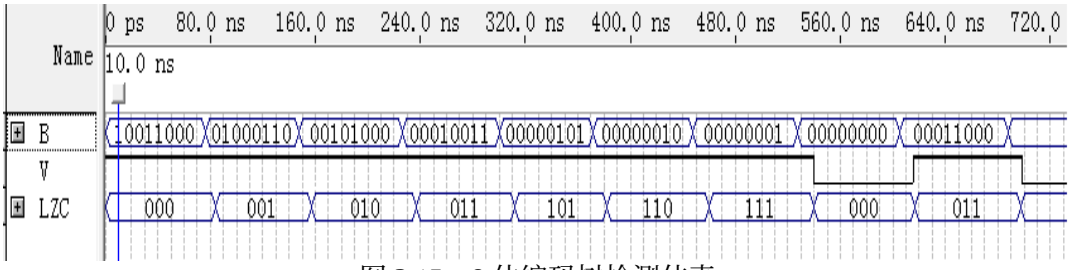



图 3.15　8 位编码树检测仿真

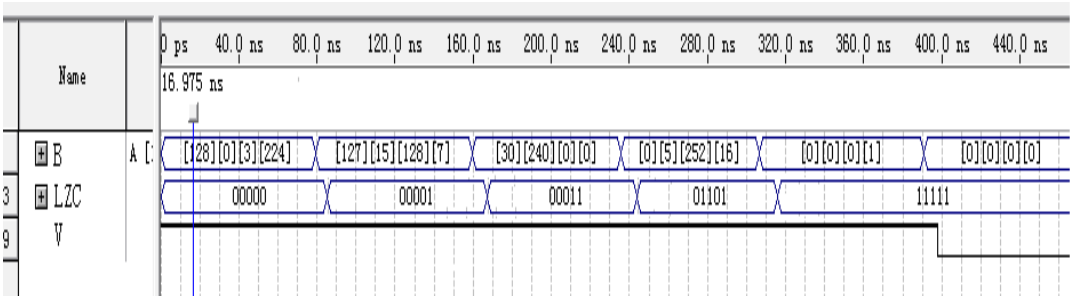



图 3.16　32 位编码树检测仿真

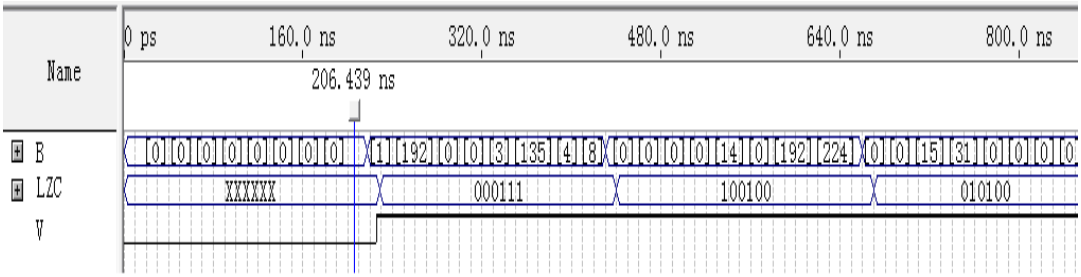



图 3.17　64 位编码树检测仿真

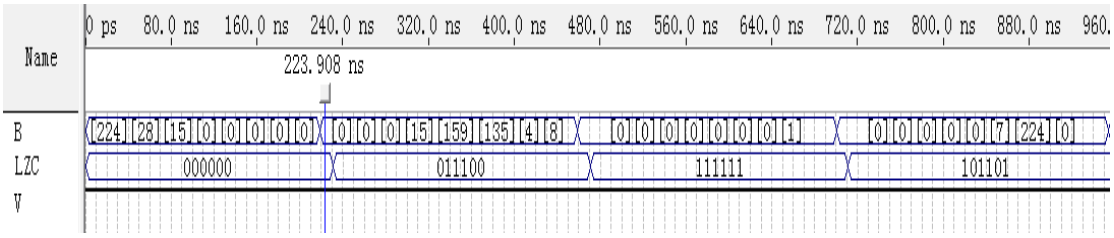



图 3.18　64 位编码树检测的仿真结果

由仿真结果分析得知，当 X=[0][0][0][0][14][0][192][224]，即首 1 的位置理论结果为第 36 位，仿真输出为 100100，经仿真验证，编码树电路的检测结果是完全正确的。

## 3.5 两操作数与三操作数的前导 1 预测算法的性能分析

以关键路径延时、静态功耗和占用逻辑单元为指标，采用 FPGA 逻辑综合工具对传统预测算法和三操作数前导 1 预测算法进行进行性能分析,由于传统前导 1 预测算法需要添加 3:2CSA，故又对其进行单独性能分析，图 3.19 为三操作数的性能分析报告。从图中我们得出：电路关键路径延时是 16.58ns、电路的动态功耗（Toltal Thermal Power Disspation）为 147.93mw。

而通过与传统预测算法进行比较，传统预测算法包括两部分，一部分是两操作数预测，一部分是 3:2CSA 的占用部分，分别对这两部分进行仿真性能分析，3:2CSA 部分占用的路径延时为 11.96ns,功耗为 110.43mw，两操作数预测部分，占用的路径延时为 14.01ns,功耗为 132.90mw，两部分合起来总共占用延时为 25.97，功耗为 243.33mw，由此可得到优化后的预测算法在效果上有明显改善。


图 3.19 三操作数预测算法分析报告

表 3.8 两操作数与三操作数的性能分析比较

| 性能指标 | 占用逻辑单元/个 | 功耗/mw | 关键路径延时/ns |
|---|---|---|---|
| 两操作数预测 | 160 | 132.90 | 14.01 |
| 3:2CSA | 128 | 110.43 | 11.96 |
| 传统前导 1 预测 | 288 | 243.33 | 25.97 |
| 三操作数预测 | 479 | 147.93 | 16.58 |

由表 3.8 分析结果可以看出，与二操作数 LOP 算法相比，占用逻辑单元虽然增加了，但三操作数前导 1 预测算法在功耗和关键路径延时方面较传统前导 1 预测（添加上 3:2CSA 后）有明显的优势，其中三操作数预测模块的关键路径延时减少了 36.15%，功耗降低了 39.20%，所以采用本文设计的三操作数前导 1 预测算法结构将缩短整个浮点乘加融合部件的延时。

## 3.6 本章小节

论文依据三操作数前导 1 预测的预编码规则，重点设计出了三操作数前导 1 预测算法编码树的结构，通过编写可综合的代码、设计编码树逻辑规则，生成子模块等实现了三操作数的前导 1 预测算法的整体结构。其中 LZD32R 转换模块使用 VerilogHDL 语言作为设计输入，优化了设计过程，且仿真数据均在 Quartus II 下进行综合得到，通过仿真结果和性能分析得出，此算法结构可以正确地完成预测的功能，且在很大程度降低了关键路径延时和功耗，具有很大的可行性。

# 第四章　浮点乘加融合体系结构的设计

乘加融合操作作为一种基本的操作，被广泛应用于许多科学计算和工程应用领域中。乘加融合操作是指将乘法运算得出的结果与另外一个操作数融合相加，即完成A+(B×C)操作，从而得到最终结果，这样就可节省整个乘加操作的执行延迟[15-17]，所以现在处理各种乘加运算时均把其作为一项单独的指令去执行。现在设计处理器都习惯将浮点双精度乘加体系包含在浮点运算单元中。

本章以典型浮点乘加融合结构体系为基础，依次完成主要模块的设计，主要包括操作数解码，乘加器的实现，前导 1 预测部分，规格化处理，舍入处理。其中乘加器模块主要有以下部分组成，部分积产生，部分积选择，部分积求和，161 位右移对阶，加法器部分。

## 4.1 解码模块

首要步骤就是对操作数 A,B,C 进行解码。首先从寄存器[18]取出 A、B、C 三个操作数，其操作数的数据格式如表 4.1 所示。在第二章中我们提到在 64 位浮点数中 52 位小数（尾数）加上隐含位可提供 53 位精度，因此我们要通过解码来提取出尾数与隐含位 1，尾数部分用原码来表示，因为原码规格化数的尾数第一位为 1，即纠正解码 E 来使小数点右移一位，由此以来尾数可表示成 1.M，这样尾数可省略在第一位的 1。表 4.1 为操作数解码格式。图 4.2 为 Verilog 程序描述。

表 4.1　A,B,C 数据格式

| 名称 | 长度 | 比特位置 |
|---|---|---|
| 符号位 Sign(S) | 1bit | b63 |
| 指数部分 Exponent(E) | 11bit | b62-b52 |
| 尾数部 Mantissa (M) | 52bit | b51-b0 |

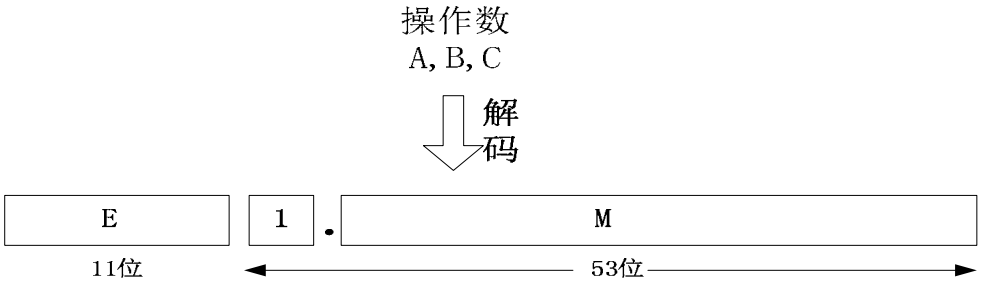



图 4.1　解码需要做的工作

```
module decode2(A,B,C,MA,MB,MC,EA,EB,EC,SA,SB,SC);
input [63:0]A;input [63:0]B;input [63:0]C;
output [52:0]MA;output [52:0]MB;output [52:0]MC;
output [10:0]EA;output [10:0]EB;output [10:0]EC;
output SA,SB,SC;
reg [52:0]U;reg [52:0]W; reg [52:0]Y;
reg [10:0]V; reg [10:0]X;reg [10:0]Z;
reg Q,R,S;
integer P;
   always@(A,B,C):
      begin
         for(P=0;P<52;P=P+1)
            U[P]=A[P];U[52]=1;
         for(P=0;P<52;P=P+1)
              W[P]=B[P];W[52]=1;
         for(P=0;P<52;P=P+1)
              Y[P]=B[P];Y[52]=1;
         for(P=52;P<63;P=P+1)
            V[P-52]=A[P]; Q=A[63];
         for(P=52;P<63;P=P+1)
            X[P-52]=B[P];    R=B[63];
         for(P=52;P<63;P=P+1)
              Z[P-52]=C[P];S=C[63];
      end
  assign MA=U; assign MB=W; assign MC=Y;
  assign EA=V; assign EB=X; assign EC=Z;
  assign SA=Q; assign SB=R; assign SC=S;
endmodule
```

图 4.2　解码程序描述

## 4.2 乘加器设计

浮点乘加融合是把乘法和加法当做一条不可分割的指令来进行，其实现过程是通过一个合并的乘法和加法结构完成的。乘法器的输出是 $B \times C$ 尾数相乘得到的 SUM 和进位 C 来表示的，然后与加法 A 共同输入到 3：2CSA 进位存储加法器，产生两个新的 SUM 和进位 C,从而再求它们的和。这样可减少了硬件面积和延时，节省了成本。

在本文的结构中，乘法器的部分采用优化后的 Booth 算法进行部分积的乘法，而部分积的求和采用的是 4:2CSA 进位存储加法器来实现的。如图 4.3 为操作数 B,C 尾数相乘的乘法器结构。其中 part1~part28 为部分积[2,18] 。

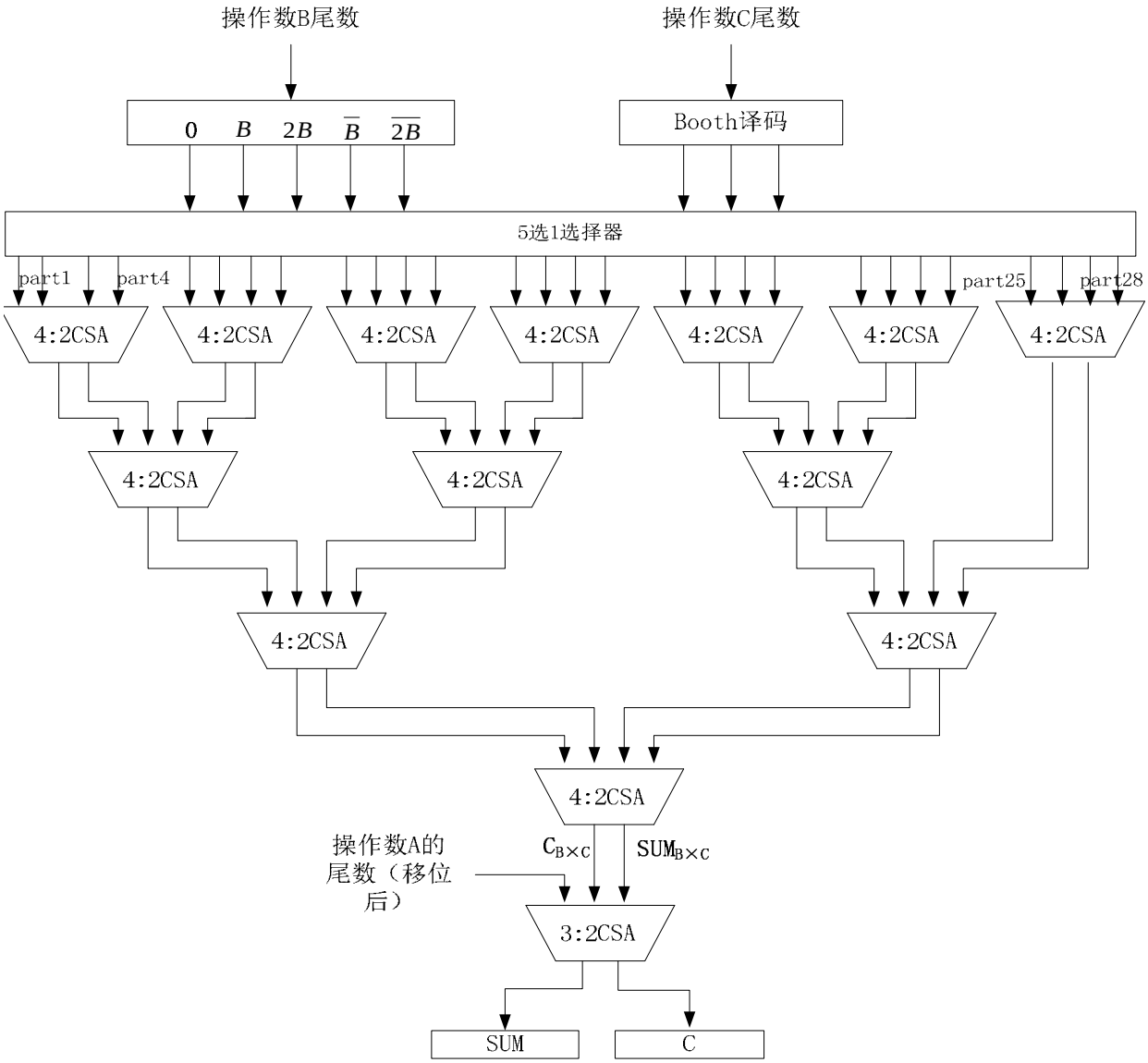



图 4.3　B×C（尾数）乘加器结构

### 4.2.1 部分积符号扩展和部分积的形成

在这里参与乘法运算的是 B 和 C 的尾数部分，且均以补码进行运算。依据 Booth 补码两位乘运算规则，如表 4.2 所示。

表 4.2　优化后的 Booth 补码运算

| $C_{n-1}$ | $C_n$ | $C_{n+1}$ | $C_{n+1}+C_n-2C_{n-1}$ | 说　明 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 右移两位 |
| 0 | 0 | 1 | 1 | 部分积加$[B]_{补}$，　右移两位 |
| 0 | 1 | 0 | 1 | 部分积加$[B]_{补}$，　右移两位 |
| 0 | 1 | 1 | 2 | 部分积加$[2B]_{补}$，右移两位 |
| 1 | 0 | 0 | -2 | 部分积加$[-2B]_{补}$，右移两位 |
| 1 | 0 | 1 | -1 | 部分积加$[-B]_{补}$，右移两位 |
| 1 | 1 | 0 | -1 | 部分积加$[-B]_{补}$，右移两位 |
| 1 | 1 | 1 | 0 | 右移两位 |

由上表所示，我们需要得到$[B]_{补}$, $[2B]_{补}$, $[-2B]_{补}$，$[-B]_{补}$, 0 五个部分积，针对$[-B]_{补}$，可对 $M_B$ 取反加 1，$[-2B]_{补}$是对 $2M_B$ 取反加 1，而在这里则需要考虑这个加 1 的问题，若是为其单独再设一个加法器，则会增大硬件电路面积，增大没必要的延时，为了提高效率，当遇到求负数补码时，我们把加 1 的操作放在与部分积求和一起进行[19]。加 1 的示意图如图 4.4 所示。

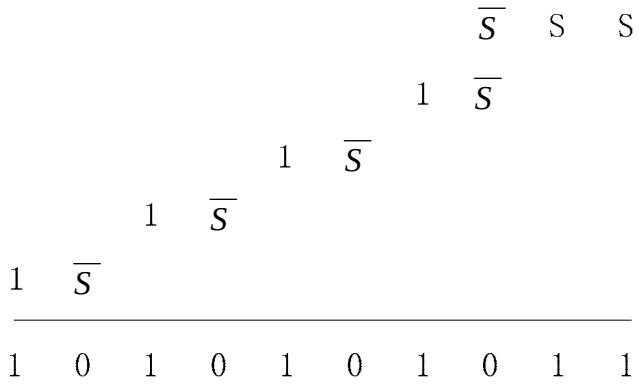



图 4.4　部分积的符号扩展

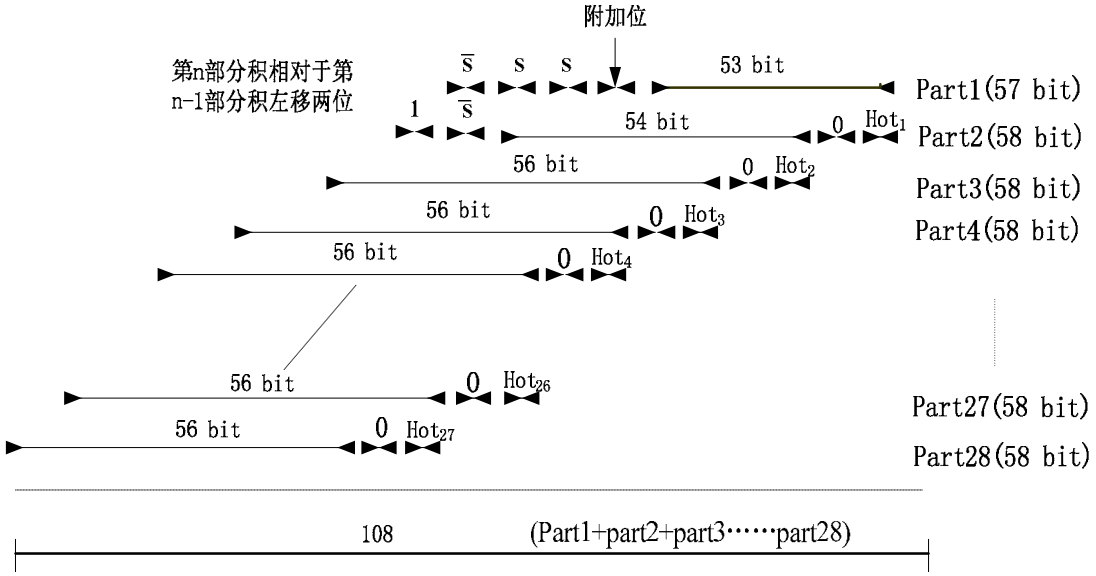



图 4.5　部分积（移位-添位）示意图

按照部分积的符号扩展，部分积为 53 位，但是由于在求[2B]补，[-2B]补,防止将有效位移出，即增添一位附加位，为 54 位，再加上需要进行符号扩展，如上图所示。即 part1 要再添置三位，为 57 位。其余部分积位 56 位。如图 4.5 所示为对部分积 part1~part28 进行添位操作结构图。

由于 B×C（53bit×53bit），因此我们得到的部分积实际总数应该是 $\frac{53+1}{2}=27$ 个，但是第 27 个部分积需要求补，因此，在这里本论文加上一个 part28。由改进的 booth 运算法则可得到，每一次与选择后的部分积相加之后，需要右移两位，即第 n 个部分积相对第 n-1 个部分积向左移两位，如上图所示。只有在第 n-1 个部分积为负数的时候,此时 Hot 位就是对应的“1”，当是正数的情况，Hot 位就是“0”，这样就可实现把加 1 的操作放到了部分积求和的过程，从而可提高效率，最后得到的 part1+part2……+part28 为 108 位[20]。

## 4.2.2 部分积的形成设计

部分积的形成是通过对乘数 C 的每相邻三位来判断进行的。在这里，本文以产生 6 个部分积（part1~part6）为例，如图 4.6 所示为部分积形成的程序流程图,以及产生部分积的程序代码见附录。

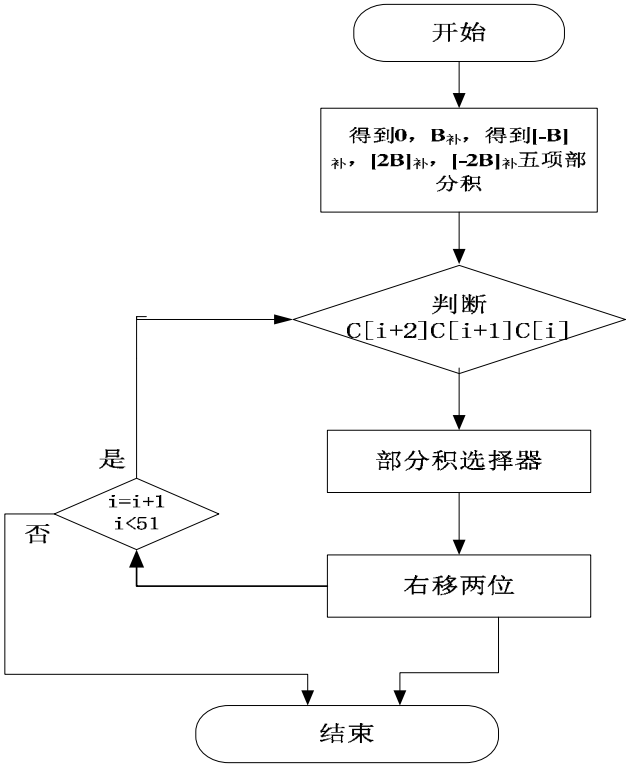



图 4.6　部分积形成流程图

### 4.2.3 部分积选择器

由上图 4.3 所示的乘加结构可知，需要设计选择器来对五项部分积进行选择，依据 Booth 补码两位乘运算规则，根据 $C_{n-1}$，$C_n$，$C_{n+1}$ 进行判断。本文在这里假定 $C_{n-1}$、$C_n$、$C_{n+1}$ 用 $C_0$、$C_1$、$C_2$ 表示，$d_1$、$d_2$、$d_3$、$d_4$、$d_5$ 分别表示部分积 0，$[B]_{补}$，$[2B]_{补}$，$[-B]_{补}$, $[-2B]_{补}$，Y 表示输出的部分积。如表 4.3 所示为选择器的真值表。

表 4.3　部分积选择器真值表

| $C_0$ | $C_1$ | $C_2$ | Y |
|---|---|---|---|
| 0 | 0 | 0 | $d_1$ |
| 1 | 1 | 1 | $d_1$ |
| 0 | 0 | 1 | $d_2$ |
| 0 | 1 | 0 | $d_2$ |
| 0 | 1 | 1 | $d_3$ |
| 1 | 0 | 1 | $d_4$ |
| 1 | 1 | 0 | $d_4$ |
| 1 | 0 | 0 | $d_5$ |

表 4.3 为部分积选择器真值表，输入 $C_0$、$C_1$、$C_2$ 为不同的值时，输出为 5 种不同的部分积，由上表所示，部分积选择器的逻辑表达式为：

$$Y=(\overline{C_0}\,\overline{C_1}\,\overline{C_2}+C_0C_1C_2)d_1+(\overline{C_0}\,\overline{C_1}C_2+\overline{C_1}C_2\overline{C_3})d_2+\overline{C_0}C_1C_2d_3+(C_0\overline{C_1}C_2+C_0C_1\overline{C_2})d_4+..$$

$$C_0\overline{C_1}\,\overline{C_2}d_5 \qquad (4.3)$$

图 4.7，4.8 为部分积选择器 RTL 电路结构图及生成的子模块。

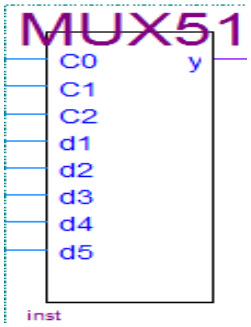



图 4.7　选择器子模块

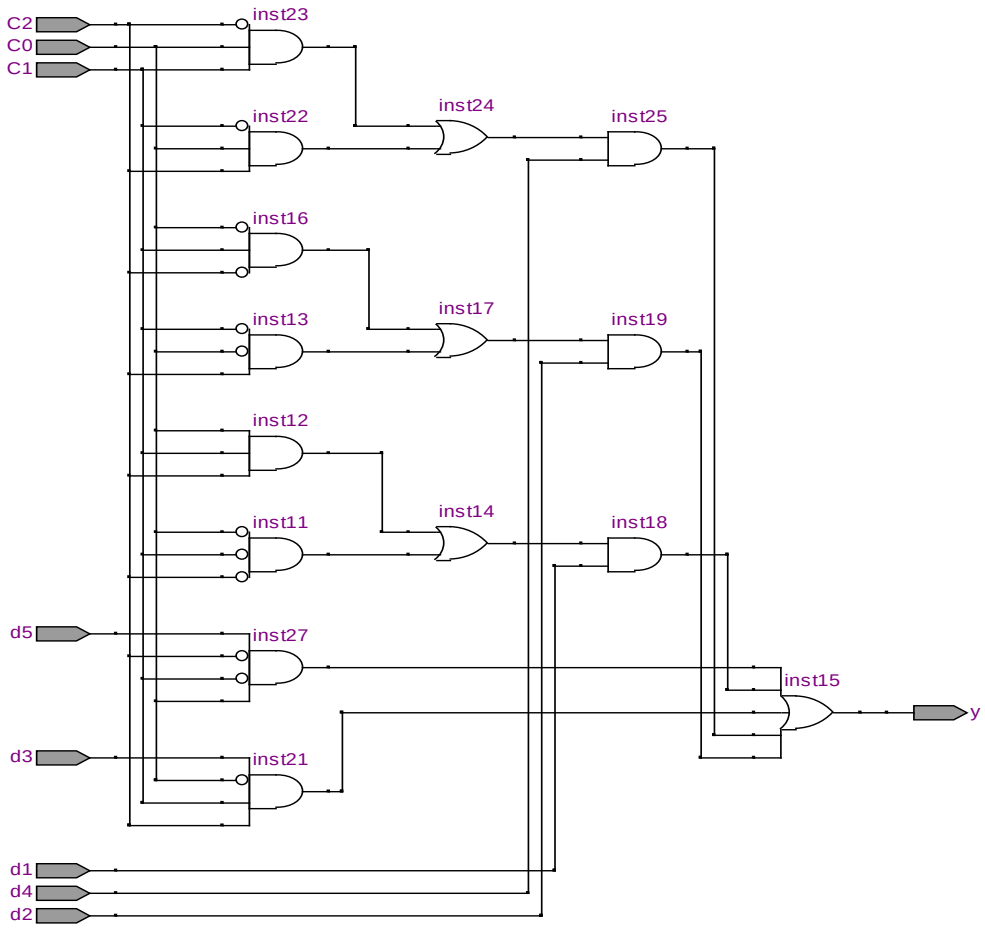



图 4.8　选择器逻辑电路图

### 4.2.4 3：2CSA 和 4：2CSA 设计

由图 4.3 乘加结构所示，需要将部分积进行求和操作，为了减少求和过程造成的延时，针对多数的加法结构采用进位存储加法器 CSA。其是采用并行的方法，多个数同时相加，求和结果化为本位和 SUM 和进位 C。本位和只与参与相加的位有关，就样就可以避免串行加法器中进位传递所造成的延时。

部分积求和结构采用 CSA 加法器是可以有效提高速度。现在通常采用的是 3：2CSA 和 4:2CSA 结构。3:2CSA 是针对三个输入，两个输出，4：2CSA 是针对 4 个输入，两个输出的[21-25]。

**1. 3：2CSA 的设计**

相加运算时，每一位的运算由相加数的本位值与上一位进位得来的值相加，即最后运算可采用本位数相加的本位和和进位进行全加运算，即可得出最后的结果。故分成两个支路，一个支路计算出本位和，另一个支路计算出进位。最后本位和和进位和进行相加。但是由于加数和被加数第 i 位的进位看做第 i+1 位的值，故需将最后的值左移一位即可。

当输入为三操作数 X,Y,Z 时，进行 X+Y+Z 的运算，产生本位和 Sum 和进位 C，$SUM_i$ 与三数的相应位 $X_i$,$Y_i$,$Z_i$ 有关，在这里以 X,Y,Z 四位为例。

表 4.4　3:2CSA 真值表

| 输入 | | | 输出 | |
|---|---|---|---|---|
| $X_i$ | $Y_i$ | $Z_i$ | $SUM_i$ | $C_i$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

故由上述真值表，可得到组合逻辑表达式：$SUM_i = \overline{X_i}\,\overline{Y_i}Z_i + \overline{X_i}Y_i\overline{Z_i} + X_i\overline{Y_i}\,\overline{Z_i} + X_iY_iZ_i$

$C_i = \overline{X_i}Y_iZ_i + X_iY_i\overline{Z_i} + X_i\overline{Y_i}Z_i + X_iY_iZ_i$。

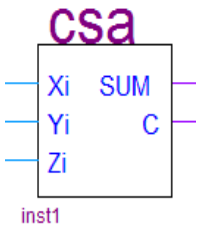



图 4.9　CSA 子模块

上一步可得到进位 C[3:0]。下一步就要进行进位的移位操作，因为由于加数和被加数第 i 位的进位要看成第 i+1 位，故左移一位即可。

设置$C'[0]=0$；$C'[1]$=C[0]；$C'[2]$=C[1]；$C'[3]$=C[2]；$C'[4]$=C[3]；本论文在这里采用编写 Verilog 代码来实现。如图 4.10 为程序描述。在这里 A 表示成 C,B 为左移之后的$C'$。

```
module ONESHIFT(A,B);
input [4:0]A;
output [4:0]B;
reg [4:0]Q;
integer P;
always@(A)
begin
      Q[0]=0;
      A[4]=0;
 for(P=0;P<4;P=P+1)
      Q[P+1]=A[P];
      end
   assign B=Q;
endmodule
```

图 4.10　左移一位--设计程序

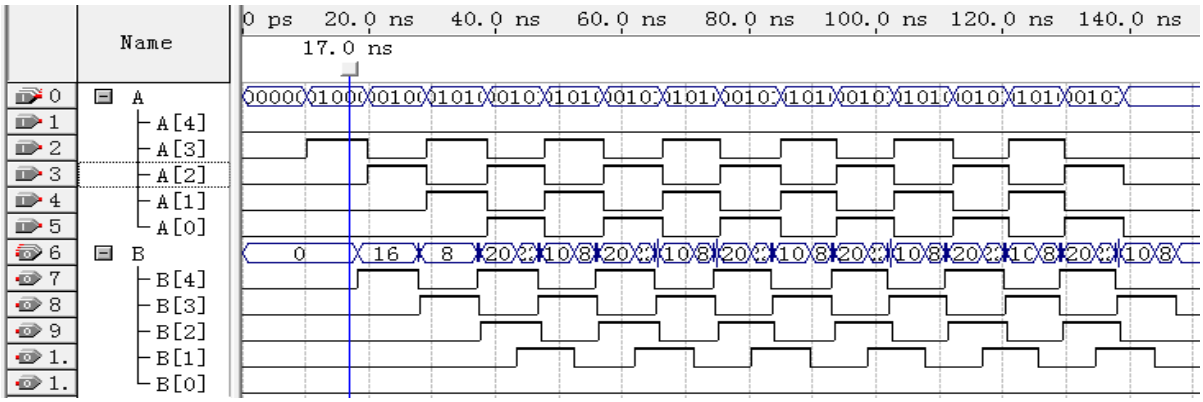



图 4.11　进位 C 左移一位仿真图

进位移位之后得到$C'[4:0]$，第二步进行 SUM[4：0]和$C'[4:0]$的全加运算得到 F 和 CARRY。如图 4.12 所示为 RTL 级综合电路图。

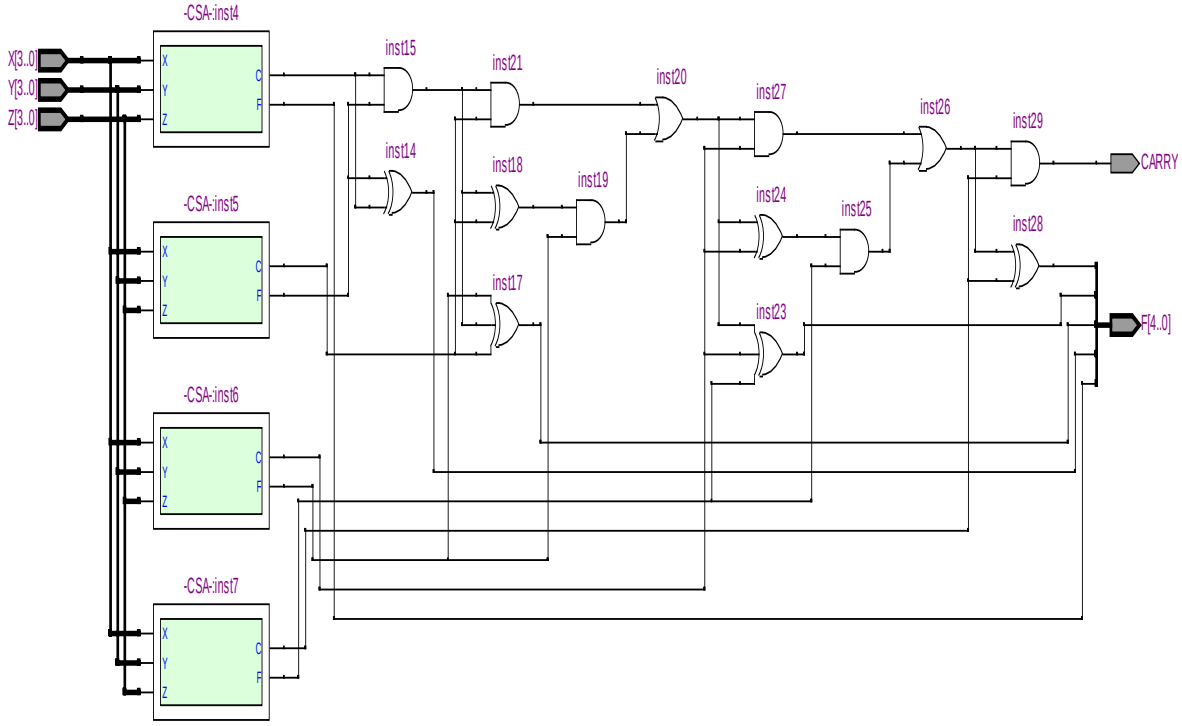



图 4.12　RTL 级电路图

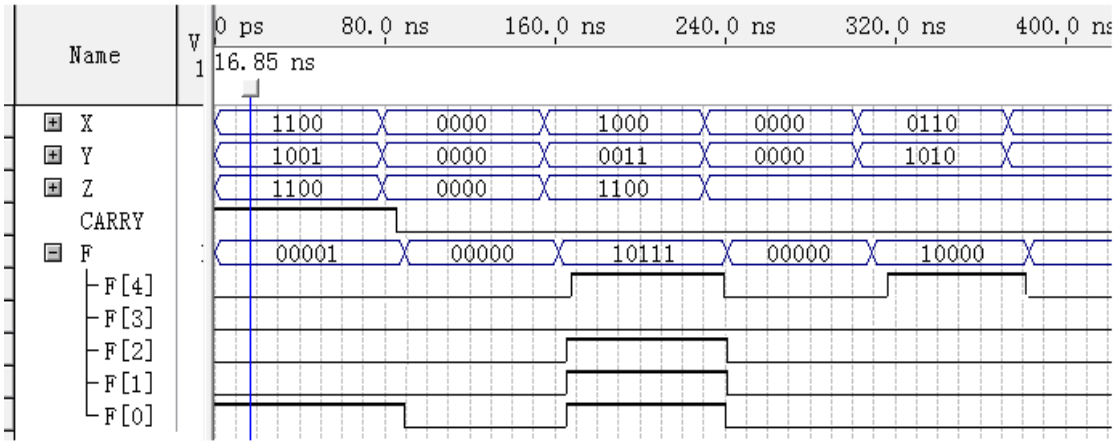



图 4.13　X+Y+Z(3:2CSA)仿真波形

2. 4:2CSA 加法器设计

由乘加器结构所示，需要 4：2CSA 来进一步提高速度，减少延时，4：2CSA 是通过两个 3:2CSA 来设计实现的，如图 4.14 所示，输入数据 $X_4$ $X_3$ $X_2X_1$，Cin 是从低位产生的进位，Cout 是第一个 CSA 产生的进位信号，Sum 为 4:2CSA 输出的和，C 为 4:2CSA 输出的进位。

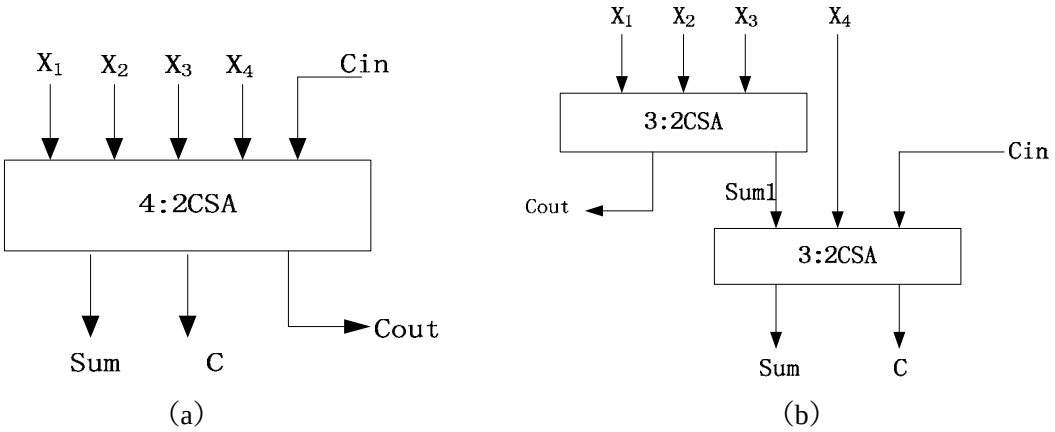



图 4.14　由两个 CSA 形成的 4:2CSA 结构图

表 4.5　4:2CSA 真值表

| 输入 | | | 输出 | |
|---|---|---|---|---|
| M | Cin | Cout | Sum | C |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 3 | 1 | 1 | 0 | 1 |
| 4 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 1 |

由上述 4:2CSA 真值表所示，X 表示不定态，M 代表输入数据中 1 的个数。由上述表中可得到组合逻辑表达式。在这里本论文采用把组合逻辑表达式进行变换，选择用异或门和二选 1 选择器来实现，从而简化组合逻辑电路，减少延时。如图 4.15 所示。

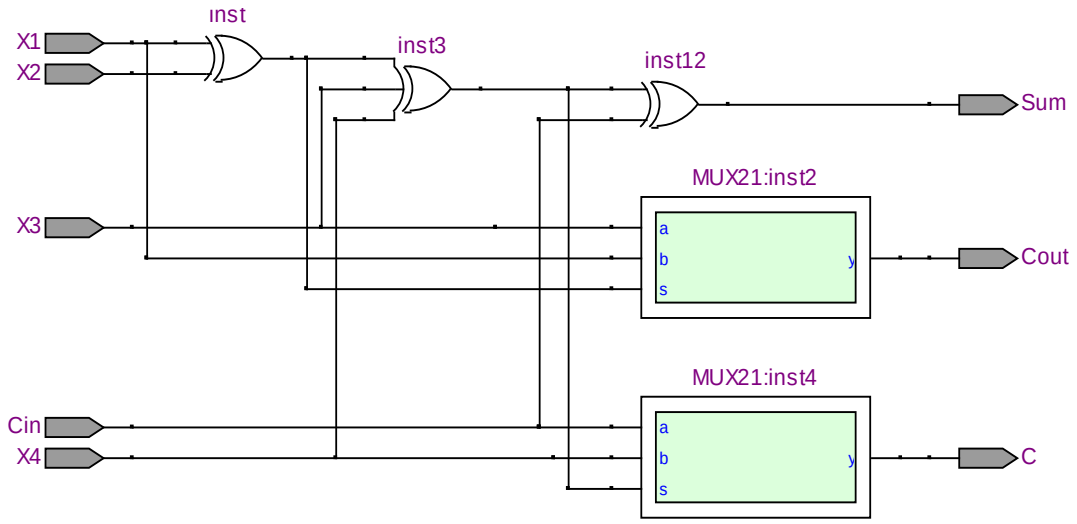



图 4.15　4:2CSA 实现 RTL 级结构图

## 4.3 对阶移位

要使 A 的尾数与 B(尾数)×C(尾数)指数对齐，必须要完成对阶操作，才即可开始运算。一般情况下，对阶操作我们采用一个 161 位的移位器进行实现[24-26]。由上节我们得到 B(尾数) ×C(尾数)得到是 108 位(包含隐含位和符号位)，如图 4.16 所示。

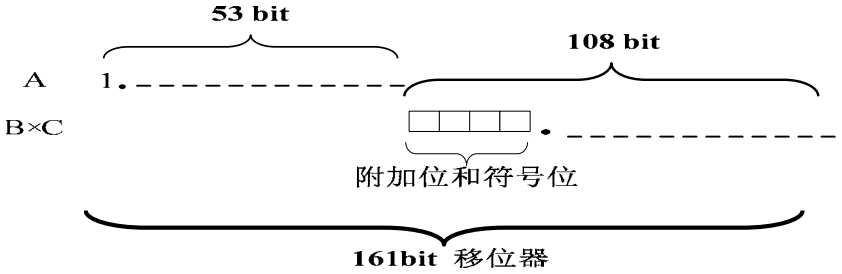



图 4.16　移位器结构图

通过移动加数 A 来实现与 B×C 小数点对齐，由上图所示，可知 A 移动的位数 shift Count=exp(B×C)-exp(A)+56，其中 exp(B×C)为 B 和 C 的指数和，exp(A)为 A 的指数，其中 56 位包含两部分，一个是 A 的尾数为 52 位（除去隐含位），二是 B×C 的符号位和附加位。在这里本章采取编写 Verilog 代码来实现移位器的设计。程序代码如图 4.17 所示。

在这里如果 count<0,说明 BC 的指数和远小于 A, BC 指数和结果对运算的精度产生较大的影响，此时舍弃掉 BC 相乘得到的 Sum 和 C。如果 count>161，即 BC 的指数和远大于 A，此时舍弃掉 A 即可。

```
module shiftcount(EA,EB,EC,S1,S2);
input [10:0]EA;input [10:0]EB; input [10:0]EC;input [160:0]S1;
output reg [160:0]S2;
integer i,shift;
integer AA,BB,CC;
reg [10:0]A; reg [10:0]B; reg [10:0]C;reg [160:0]M1; reg [160:0]M2;
always@(EA,EB,EC,S1,S2)
   begin
      A=EA;
      B=EB;
      C=EC;
      M1=S1;
      AA=A;
      BB=B;
      CC=C;
      shift=BB+CC-AA;
   if(shift>105)
      begin
          for(i=0;i<53;i=i+1)
              begin
                 M2[i+107-shift]=M1[i+107];
                 M2[i+107]=0;
               end
      end
    else if((shift>-105) &(shift<105))
       begin
           for(i=0;i<53;i=i+1)
              begin
                 M2[i+107-shift]=M1[i+107];
                 M2[i+107]=0;
              end
         end
   else M2=M1;
   S2=M2;
   end
end module
```

图 4.17　移位操作程序代码

## 4.4 舍入模块

经过 A+B×C 得到的运算结果做规格化移位之后，还要经过舍入环节做最后处理。根据浮点数据的标准格式中，浮点数据的尾数总可以用确定的尾数来表示，但是若经过加减乘除等最后的运算结果超出给定的位数时，计算机在处理时往往把超出的那部分给删掉，由此导致丢失尾数低位的值，所以就要进行舍入操作，以期达到减小误差的目的。在运算过程中保留右移中移出的若干高位的值，最后利用这些个位来纠正尾数，现在被广泛采用的舍入模式有四种，最近舍入，零舍入，正无穷和负无穷舍入。如图 4.18 所示，在这里 L 指的是舍入结果的最低位，其右侧为舍入位 R,S 为粘帖位（为 $A_{m-2}+A_{m-3+}\cdots\cdots A_0$ 相或的结果），本文在这里采用就近舍入法来进行设计[27-29]。如图 4.19 为程序代码。

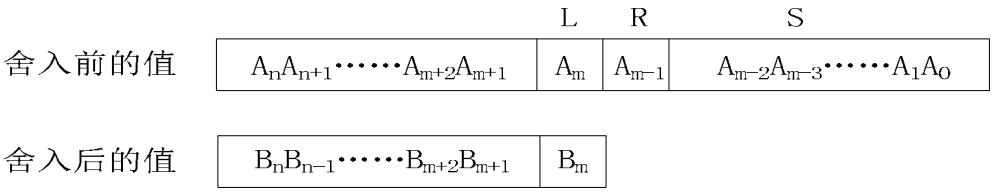



图 4.18　舍入操作

```
Module round(L,R,S)
Input L,R,S;
Begin
    if(R==0)
          begin    return 0;    end
    else if ( R& (S or L) )
        begin    return 1; end
       else return 0;
end
```

图 4.19　舍入程序

## 4.5 本章小结

本章着重对乘加融合体系结构的具体模块进行了一一设计和实现，其中包括 A，B，C 的解码，部分积形成器，部分积选择器，进位存储加法器（3:2CSA 和 4：2CSA），加数 A 移位，舍入模块等，通过采用 Verilog 硬件语言进行设计，编程实现，绘制电路图，实现设计。

# 第五章　浮点乘加融合系统仿真综合验证

## 5.1 浮点乘加融合体系结构

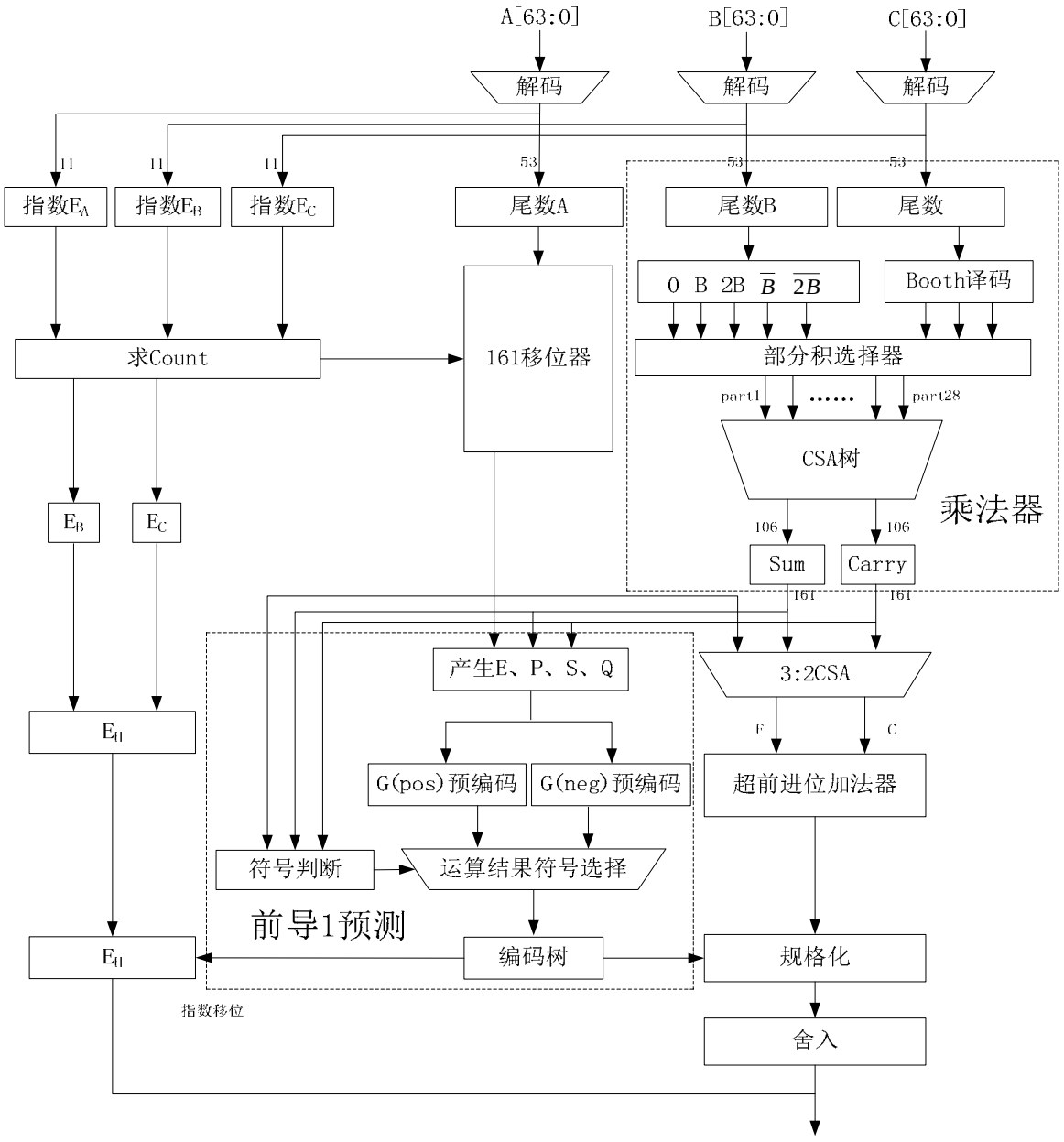



图 5.1　乘加融合 A+B×C 总体结构

在第四章中我们已经设计实现了浮点乘加融合结构的各个功能模块，其中包括解码，乘运算、对阶移位、加运算、前导 1 预测、规格化、舍入模块，对设计的各个模块进行仿真测试验证，采用 Quartus II 软件进行波形仿真[30-41]，在 FPGA 平台上进行硬件测试等，验证设计的系统结构是否正确可行。

## 5.2 浮点乘加融合--模块仿真验证

由下图所示为仿真测试流程，在这里我们依据流程图对各个模块进行仿真测试和仿真结果的分析。

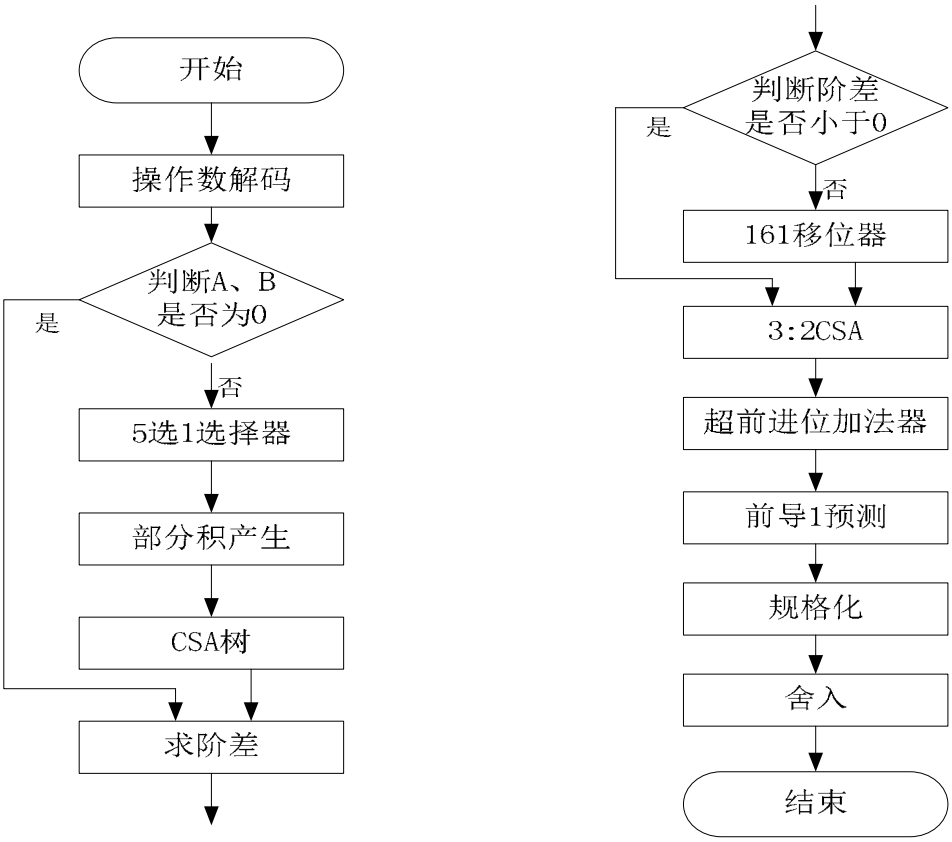



图 5.2　乘加融合体系结构仿真测试流程

### 5.2.1 操作数解码模块验证

第一步解码，下面是 Verilog 语言编写的解码程序。图 5.3 为生成的解码 RTL 级电路图和 A、B、C 的解码仿真波形。表 5.1 为仿真分析表。

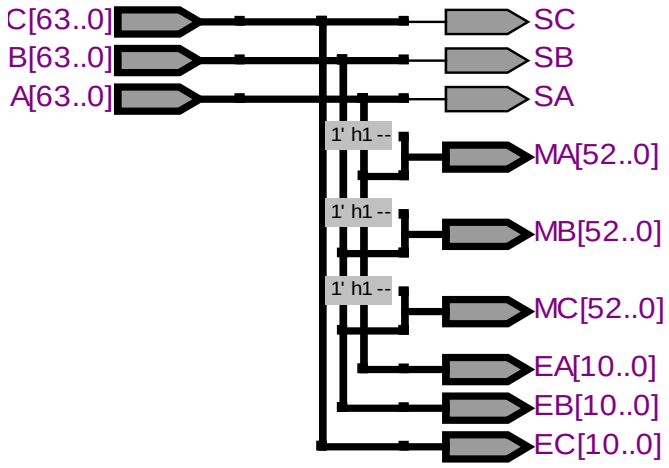



图 5.3　解码（RTL 级电路图）

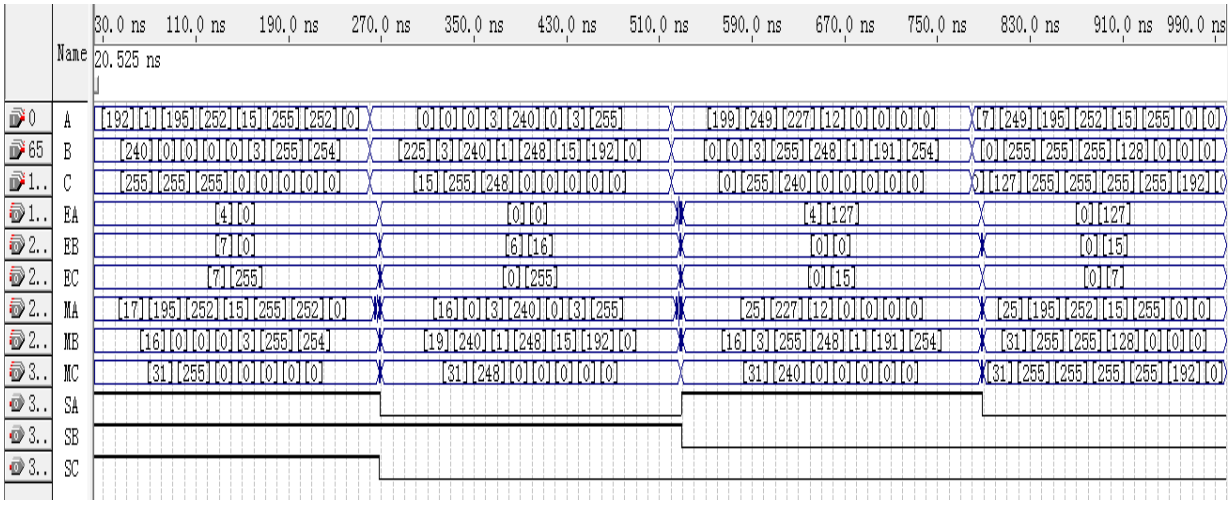



图 5.4　解码仿真波形

表 5.1　仿真结果分析

| 输入 | A | | B | | C | |
|---|---|---|---|---|---|---|
| 输出 | [192][1][195][252][15][255][252][0] | | [240][0][0][0][0][3][255][254] | | [15][255][248][0][0][0][0][0] | |
| | SA | 1 | SB | 1 | SC | 0 |
| | EA | [4][0] | EB | [7][0] | EC | [0][255] |
| | MA | [17][195][252][15][255][252][0] | MB | [16][0][0][0][3][255][254] | MC | [31][248][0][0][0][0][0] |

由上述仿真结果分析表可得到，64 位输入数据 A、B、C 分别是[192][1][195][252][15][255] [252][0]，[240][0][0][0][0][3][255][254]，[15][255][248][0][0][0][0][0]，解码后得到的（符号位）S 分别为 1、1、0，11 位 E（指数位）分别为[4][0]，[7][0]，[0][255]，53 位 M（尾数位）分别为[17][195][252][15][255][252][0]，[16][0][0][0][3][255][254]，[31][248][0][0][0][0][0]。

## 5.2.2 乘加器各模块仿真验证

**1.部分积选择器**

本文在这里用 $C_0$、$C_1$、$C_2$ 代表被乘数 C 任意相邻的三位，$d_1$、$d_2$、$d_3$、$d_4$、$d_5$ 表示 0，$[B]_{补}$，$[2B]_{补}$，$[-B]_{补}$，$[-2B]_{补}$部分积，通过对 d1~d5 设置不同的输入频率，Y 表示经过选择之后的输出部分积，其经过 Quartus II 仿真之后的波形如下图所示。且由仿真波形可得出，当 $C_0$、$C_1$、$C_2$ 设置不同的值时，其输出 Y 选择的结果也不一样。

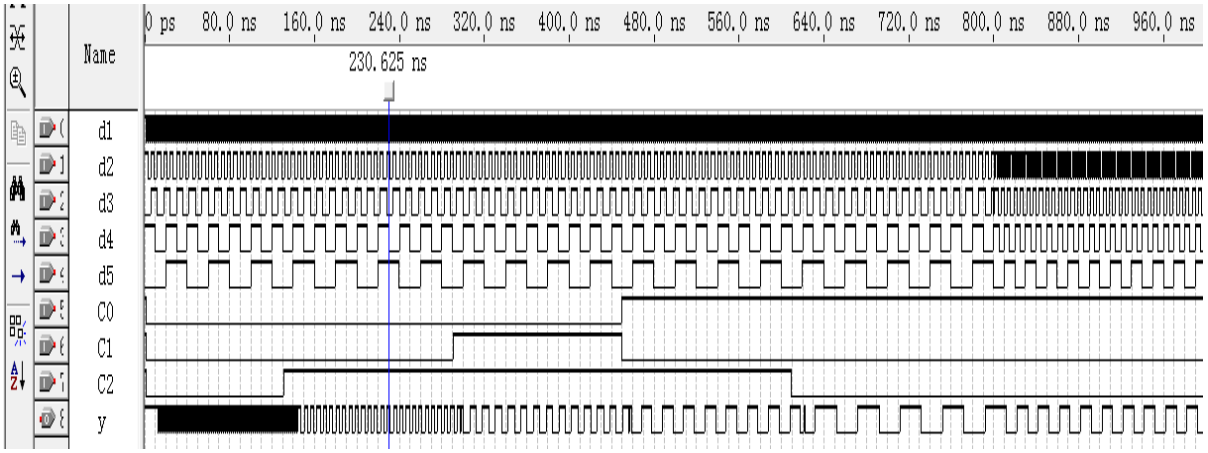



图 5.5　选择器仿真波形

**2.部分积产生设计**

部分积的形成是通过对乘数 C 的每相邻三位来判断进行的。在这里,本文以产生 6 个部分积（part1~part6）为例，如图 5.6 为仿真波形以及仿真分析表。

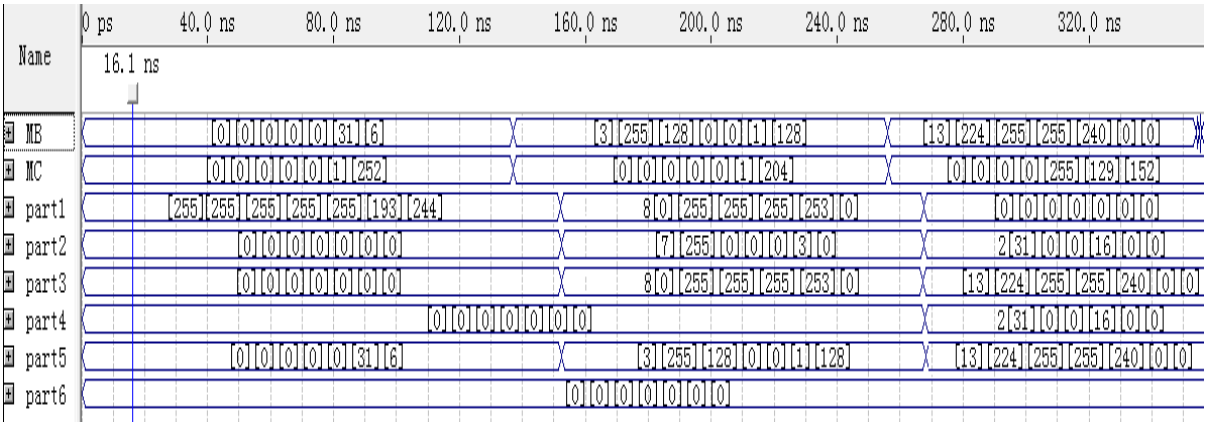



图 5.6　部分积生成波形图

表 5.2　仿真结果分析

| 输入 | B:[0][0][0][0][0][31][6] | | B : [3][255][128][0][0][1][128] |
|---|---|---|---|
| | C:[0][0][0][0][0][0][1][252] | | C : [0][0][0][0][0][1][204] |
| 输出 | Part1 | [255][255][255][255][255][193][244] | 8[0][255][255][255][253][0] |
| | Part2 | [0][0][0][0][0][0][0] | [7][255][0][0][0][3][0] |
| | Part3 | [0][0][0][0][0][0][0] | 8[0][255][255][255][253][0] |
| | Part4 | [0][0][0][0][0][0][0] | [0][0][0][0][0][0][0] |
| | Part5 | [0][0][0][0][0][31][6] | [3][255][128][0][0][1][128] |
| | Part6 | [0][0][0][0][0][0][0] | [0][0][0][0][0][0][0] |

由仿真结果分析可得到, 当 B 为[0][0][0][0][0][31][6], C 为[0][0][0][0][0][0][1] [252]时, 即 C 的最低三位为 100, 即为部分积为 $2[-B]_{补}$, 则 part1 部分积为[255][255][255][255][255][193][244]，则仿真正确。右移两位之后，C 的最低三位为 111，即部分积 Part2 为 0，本文设计的部分积生成模块仿真正确。

**3.加法器 3:2CSA 和 4:2CSA**

本文设计的 3：2CSA 和 4：2CSA 结构, Quartus Ⅱ软件仿真的结果如下图所示。然后在 FPGA 的 DE2 平台上进行仿真验证，通过在 Quartus Ⅱ软件上对输入输出信号进行引脚配置，采用 SW17、SW16、SW15、SW14、SW13 作为输入信号的控制开关，LEDR[0],LEDR[1],LEDR[2]作为输出信号，通过改变控制开关， 进行相应的 LED 亮灯显示，其仿真实物图如图 5.8 所示。

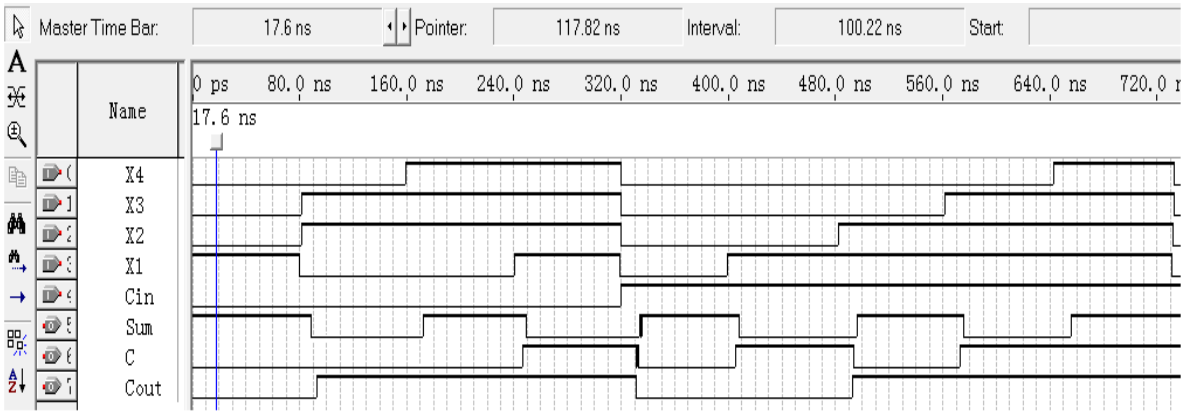



图 5.7　4:2CSA 仿真波形


图 5.8　4:2CSA DE2 仿真实物图

### 5.2.3 161 位移位器验证

当进行 B×C 乘加操作时，并行进行加数 A 的移位操作，使其与 B×C 的阶码对齐，在 161 位移位器中完成。$E_A$、$E_B$、$E_C$、$M_1$ 作为输入信号，输出结果为移位后的 $S_1$ 为 161 位，53 位尾数输入时在后面补“ 0 ”。仿真示意图如图 5.10 所示。

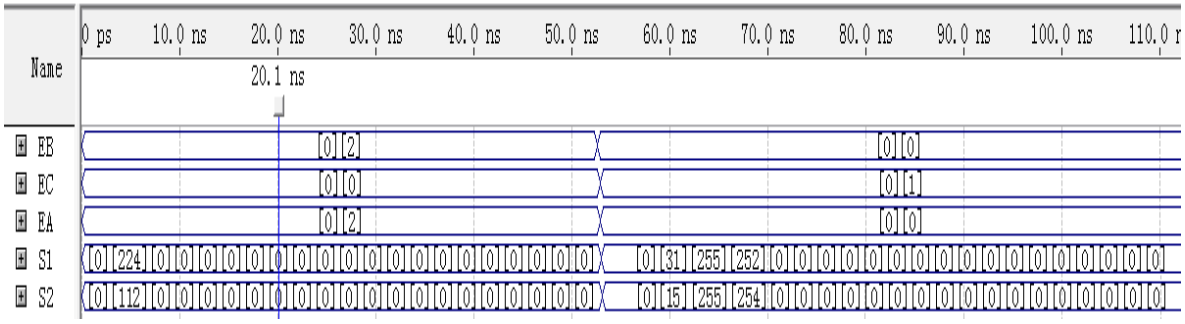



图 5.10　161 位移位器仿真图

### 5.2.4 前导 1 预测验证

前导 1 预测部分已在第三章中进行仿真验证，在这里不再赘述。

## 5.3 本章小结

本章主要是对浮点乘加的融合系统中各个关键模块的仿真验证，编写了测试程序，描述了测试程序的执行过程，以及分析了仿真结果。最后，将程序或者电路图移植到 FPGA 芯片中，对其进行仿真测试，通过 DE2 上的 LED 灯显示测试程序的结果。

# 第六章　总结与展望

## 6.1 总结

本论文在深入研究浮点乘加融合结构的基础上，通过设计浮点乘加融合系统中的三操作数前导 1 预测优化算法，使其在功耗方面以及关键路径的延时上有了明显的提高，最后在 FPGA 实验平台上进行仿真测试其可行性，仿真结果表明设计的优化算法结构能够有效运行，对计算机处理大量数据方面有较大的可行性。

论文通过深入分析现今浮点乘加融合思想与结构，通过设浮点乘加融合结构中前导 1 预测环节中的关键算法来完成研究目的，采用 Verilog 语言对各个模块进行编程设计，生成子模块，最后对各个模块进行综合仿真，并在 Altera 公司的 DE2 平台上进行仿真实现。

论文首先介绍了浮点运算的基础知识，以及对浮点乘加融合思想与结构进行了详细的介绍，为实现浮点乘加融合体系结构，把系统结构模块化，分为以下主要模块：

1.解码模块：根据 64 位浮点数的格式，对 A,B,C 三个操作数进行解码操作，提取出最高位符号位 S 与最后 52 位尾数 M 构成 53 位有效数字，11 位指数位 E 单独组成一个有效数字。

2. 乘法器的设计，乘法器的设计包括符号扩展、部分积的产生、5 选 1 选择器以及 4:2CSA 进位存储加法器。

3. 移位器，通过移位要使加数 C 与 A×B 乘法后结果的小数点对齐，从而来完成相同指数下的 C+ A×B 运算。本论文要在这里采用一个 161 位的移位器，加数 C(尾数)的有效位需要右移的位数为 Count=Exp（A×B）－Exp（C）+56，根据 Count 的值来移位量。

4. 最重要的创新点就在于设计前导 1 预测模块。在这一模块先是分析了两操作数前导 1 预测的编码规则，并深入探讨了其存在的不足，并针对这一不足，在 FPGA 平台上设计了能够直接处理三操作数的前导 1 预测算法的完整实现方案，可以有效降低关键路径延时和功耗，通过在 FPGA 硬件验证平台上对系统结构合理模块化，且采用硬件描述语言 VerilogHDL 对部分功能进行编程，优化了设计过程，最后对仿真结果进行了分析。

5. 舍入，运算结束后经过规格化，最后需要结果舍入，本文采用的是最近舍入

法。

使用 Quartus Ⅱ软件工具，采用 Verilog 语言进行编程设计，实现电路图的设计实现，生成模块，最后在 DE2 实验板对该设计进行仿真实现。最后，结合各个模块综合出整个 64 位的浮点乘加融合器，通过验证并进行分析。

## 6.2 展望

本文针对的是 64 位浮点数据，因此在这个基础之上更进一步的研究是非常必要的。在今后的工作上可重点研究一下几个方面：

1．进一步结构的过程段，均衡各个过程段之间的所产生的延时，进一步提高运算速率。

2．本文采用的是规格化的操作数，在以后的工作中设计非规格化的操作数。

3．浮点数的精度上还可以再提高，本文研究的是 64 位浮点数，可以再深入研究一下针对 128 位浮点数据的浮点乘加融合系统。

4．论文中研究的 3 操作数前导 1 预测算法虽然减少了关键路径的延时，但是在占用逻辑单元数目比较多，可以再深入研究。

# 参考文献

[1] 张峰. 128 位浮点乘加融合部件的研究与实现[D]. 国防科学技术大学,2007

[2] 毛二坤. 高性能浮点乘加部件的研究和实现[D]. 国防科学技术大学,2006

[3] 李星. 前导 1 预测算法的设计与实现[J]. 计算机科学. 2013,40(4):31~33

[4] 梅小露.浮点乘加部件中三操作数前导 1 预测算法的设计[J]. 微电子学与计算机. 2005

[5] Javuer D Bruguera, Tomas Lang.Leading-one Prediction with Concurrent Position Correction. IEEE Transactions on Computers, 1999,48(10): 1083~1097

[6] Vojin G Oklobdzija. An Algorithmic and Novel Design of a Leading Zero Detector Circuit : Comparison with Logic Synthesis. IEEE Transactions on VLSI systems, 1994,2(1):124~128

[7] 梅小露. 高性能通用处理器中浮点乘加部件的设计[D]. 中国科学院研究生院（计算技术研究所）.2005

[8] 靳战鹏 .一种 64 位浮点器的设计与实现. 计算机工程与应用. 2006

[9] 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,2005

[10] 郝志刚.64 位高性能融合乘加部件的研究与实现[D]. 国防科学技术大学,2003

[11] 周润景,图雅，张丽敏.基于 QuartusⅡ的 FPGA/CPLD 数字系统设计实例[M].北京:电子工业出版社，2007

[12] 黄智伟,王彦,陈琼,等.FPGA 系统设计与实践[M]. 北京:电子工业出版社，2005

[13] 潘松,黄继业. EDA 技术与 VHDL(第 2 版)[M]. 北京:清华大学出版社, 2006

[14] 肖朝晖. 结合 EDA 技术的“计算机组成原理”教学实践改革. 计算机教育. 2007

[15] 潘松,黄继业. EDA 技术与 VHDL(第 3 版)[M]. 北京:清华大学出版社，2009

[16] 黎峥.浮点运算单元的设计与实现[D]. 清华大学，2005

[17] RM Jessani, Michael Putrino. Comparison of Single-and Dual-Pass Multiply-Add Fused Floating-Point Units. IEEE Transaction on Computers, 1998, 47(9): 927-937

[18] 凌志强.支持并行整数乘的双通路浮点融合乘加的研究与实现[D]. 国防科学技术大学，2006

[19] Mark Ronald Santoro. Design and Clocking of VLSI Multipliers[D]. Stanford University, 1989.

[20] 数值计算指南. Sun Microsystems,Inc.www.sun.com

[21] 白中英. 计算机组成原理(第三版.网络版)[M]. 北京:科学出版社,2001

[22] 耿恒山,张军,田红丽,等.计算机组成原理[M]. 北京:机械工业出版社,2009

[23] 杨松华,冯毛官,孙万蓉,等.数字电子技术基础[M]. 西安:西安电子科技大学出版社,2000

[24] 岳伟甲.一种基于 FPGA 的 32 位快速加法器设计[J]. 四川兵工学报,2011,32(7).

[25] 赵霞,杨茜.基于进位存储加法器的数字滤波器的设计[J]. 黑龙江科技信息.2012,(12):30-30

[26] JD Bruguera, Tomas Lang. Floating-point Multiply-add Fused: Latency Reduction for Floating-Point Addtion. 17th IEEE Symposium on Source: IEEE Xplore, 2005

[27] WilliamStallings. Computer organization and architecture Performance. Higher Education Press Pearson Edueation. 2001

[28] 陈雷,高德远,樊晓桠. 一种宽位乘法器设计方案[J]. 计算机工程与应用,2003,(34): 28-30

[29] 蒋勇,罗玉平,马晏等. 基于 FPGA 的 32 位并行乘法器的设计与实现. 计算机工程,2005,31(23): 222-224

[30] Ahmet Akkas, Michael J Schulte. Dual-mode floating-point multiplier architectures with parallel operations. Journal of Systems Architecture, 2006

[31] Vassiliadis S, Schwarz E M, Sung B M. Hard—Wired Multipliers with Encoded Partial Products, IEEETrans Computer,1991,(4001): 1181—1197

[32] Bewick G. High Speed Multiplication. Electronic Research Seminar,Stanford Univ,1993

[33] 付娟. 浮点处理单元结构和算法研究[D]. 西北工业大学硕士论文,2004

[34] 孙旭光. 高性能算术单元的研究与实现[D]. 哈尔滨工业大学博士论文,2003

[35] 周昔平, 高德远, 樊晓桠.乘累加运算器的高性能解决方案[J]. 微电子学与计算机,2002, (10): 21-24

[36] 侯申,陈讯,陈跃跃.浮点乘加部件中的两种前导零预测算法. 中国计算机学会[C],2007

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

[38] 冯寅翀,张盛兵,黄嵩人等. 支持多精度小数的运算单元设计. 微电子学与计算机,2012, 29(4):151-153

[39] Yamin Li, Wanming Chu. Parallel-Array Implementations of A Non-Restoring Square Root Algorithm[J]. International Conference on ComputerDesign, 1997, (97): 690-695

[40] 刘诗斌,高德远,樊晓桠等. 嵌入式微处理运算器设计[J]. 航空电子技术, 1999,(3):18-22

[41] 张小研,邵杰. 高速浮点运算单元的 FPGA 实现. 信息化研究,2009,35(11):24-27

# 附录

## 部分积生成部分代码

```
module partmutiply(MB,MC,part1,part2,part3,part4,part5,part6);
input [53:0]MB; input [52:0]MC;
output reg [53:0]part1; output reg[53:0]part2;
output reg[53:0]part3;output reg [53:0]part4;
output reg[53:0]part5; output reg[53:0]part6;
reg [53:0]B; reg [52:0]C;
reg [53:0]Zero; reg [53:0]NB;
reg [53:0]DB; reg [53:0]DNB;
reg [53:0]p1; reg [53:0]p2;
reg [53:0]p5; reg [53:0]p6;
reg [53:0]p3; reg [53:0]p4;
integer i;
always@(MB,MC)
begin
    B=MB;
    C=MC;
  for(i=0;i<53;i=i+1)                    // add "0"
      Zero[i]=0;

  for(i=0;i<54;i=i+1)                    // get [-B]complement
       begin
         if(B[i]==0) NB[i]=1;
         else if(B[i]==1) NB[i]=0;
       end
       NB=NB+1;

  for(i=1;i<54;i=i+1)     // left-shift to get2[B]complement
         DB[i]=B[i-1];
         DB[0]=0;

  for(i=1;i<54;i=i+1)    //left-shift to get2[-B]complement
         DNB[i]=NB[i-1];                 // left-shift to get2[-B]complement
         DNB[0]=0;
```

```
  for(i=1;i<2;i=i+1)
        begin
                if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p1=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p1=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p1=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p1=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p1=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p1=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p1=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p1=Zero;end
        end
for(i=3;i<4;i=i+1)
        begin
                if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p2=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p2=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p2=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p2=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p2=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p2=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p2=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p2=Zero;end

           end
  for(i=5;i<6;i=i+1)
        begin
                if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p3=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p3=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p3=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p3=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p3=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p3=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p3=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p3=Zero;end

           end
    for(i=7;i<8;i=i+1)
        begin
                if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p4=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p4=B; end
```

```
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p4=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p4=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p4=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p4=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p4=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p4=Zero;end
          end
  for(i=9;i<10;i=i+1)
        begin
                if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p5=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p5=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p5=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p5=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p5=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p5=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p5=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p5=Zero;end
          end

  for(i=11;i<12;i=i+1)
        begin
               if ((C[i+1]==0) & (C[i]==0) &(C[i-1]==0 )) begin p6=Zero; end
                else if ((C[i+1]==0) & (C[i]==0) & (C[i-1]==1)) begin p6=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==0)) begin p6=B; end
                else if ((C[i+1]==0) & (C[i]==1) & (C[i-1]==1)) begin p6=DB; end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==0)) begin p6=DNB;end
                else if ((C[i+1]==1) & (C[i]==0) & (C[i-1]==1)) begin p6=NB;end
                else if ((C[i+1]==1) & (C[i]==1) & (C[i-1]==0)) begin p6=NB;end
                else if    ((C[i+1]==1) & (C[i]==1) & (C[i-1]==1)) begin p6=Zero;end
          end

    part1=p1;
    part2=p2;
    part3=p3;
    part4=p4;
    part5=p5;
    part6=p6;
   end
endmodule
```

# 攻读学位期间所取得的相关科研成果

[1] 罗淑贞，耿恒山，徐祥男等 .基于 HLS 协议的流媒体直播系统的研究和改进[J].郑州大学学报工学版. (2014 年 5 月)

[2] 罗淑贞，富坤，高艳，孙豪赛.基于 FPGA 的三操作数前导 1 预测算法的设计与性能分析[J].微电子学与计算机.(已确定被录用)

[3] 高艳，富坤，罗淑贞，李钦.基于 FPGA 模型机的组合逻辑控制器的设计与实现[J].实验室研究与探索.(已确定被录用)

[4] 徐祥男, 耿恒山, 罗淑贞等 . HLS 直播流媒体传输系统的冗余优化[J]. 电视技术. (2014 年 6 月)

# 致　谢

首先，我要感谢我的导师耿恒山教授，还记得刚入校时，第一次见到耿老师，就觉得耿老师是那么的亲切和蔼，进到我们的嵌入式智能控制实验室，每当有问题不明白时，耿老师总是给我耐心讲解，对待我们就像对待自己的孩子一样，孜孜不倦的教导。因为自己一直有读博的打算，还记得在选导师的时候，耿老师给我提出了那么多宝贵的建议，让我受益匪浅，虽然老师已进入花甲，但是他那认真严谨的学术态度始终是我们的向导。

感谢富坤老师，初次见到富老师就被她的热情感动了，她总是带着春风般的笑容感染着我们，这次论文的课题选择都要得益于富老师，每当研究课题时遇到困难，她总是认真指导我，并且从心灵上安慰我，让我有自信有勇气去进行研究，富老师既是我的老师也是我的朋友，很幸运能让我遇见这样的老师。

感谢我的学校，很庆幸自己是在河北工业大学读的研究生，严谨的校风，谦虚的态度教会我人生中最重要的东西，谢谢学校给我们提供了一个这么好的平台，让我在这里成长了太多太多。

感谢我的父母，我知道不论我遇到多大的困难，我都知道他们会一直站在我身后支持我，不论我走到哪里，走的多远，只要想起父母，我就觉得我不是一个人在孤军奋战，你们是我的奋斗源泉，我爱你们，感谢你们！

感谢我的朋友：高艳、李钦、徐祥男、安娜在整个论文研究期间，当有技术上和生活上的难题，他们总能给予我很大帮助，我们就像一家人一样那么有默契，很幸运在我的研究生期间能遇到你们。感谢你们！

总之，千言万语都要道一声感谢，我的朋友，老师，谢谢你们给了我太多的支持和帮助，使我度过了我人生中最有意义的学习时光，我一定不辜负你们的期望，努力为社会，为国家做出贡献。

硕士学位论文

MASTER THESIS