Single-Precision and Double-Precision Merged Floating-Point Multiplication and Addition Units on FPGA

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

By

Hao Zhang

Permission to Use

Abstract

Floating-point (FP) operations defined in IEEE 754-2008 Standard for Floating-Point Arithmetic can provide wider dynamic range and higher precision than fixed-point operations. Many scientific computations and multimedia applications adopt FP operations. Among all the FP operations, addition and multiplication are the most frequent operations.

In this thesis, the single-precision (SP) and double-precision (DP) merged FP multiplier and FP adder architectures are proposed. The proposed efficient iterative FP multiplier is designed based on the Karatsuba algorithm and implemented with the pipelined architecture. It can accomplish two parallel SP multiplication operations in one iteration with a latency of 6 clock cycles or one DP multiplication operation in two iterations with a latency of 9 clock cycles. Implemented on Xilinx Virtex-5 (xc5vlx155ff1760-3) FPGA device, the proposed multiplier runs at 348 MHz using 6 DSP48E blocks, 1117 LUTs, and 1370 FFs. Compared to previous FPGA based multiple-precision FP multiplier, the proposed designs runs at 4% faster clock frequency with reduction of 33% of DSP blocks, 17% latency for SP multiplication, and 28% latency for DP multiplication.

The proposed high performance FP adder is designed based one the two-path FP addition algorithm. With fully pipelined architecture, the proposed adder can accomplish one DP or two parallel SP addition/subtraction operations in 6 clock cycles. The proposed adder architecture is implemented on both Altera and Xilinx 65nm process FPGA devices. The proposed adder can run up to 336 MHz with 1694 FFs, 1420 LUTs on Xilinx Virtex-5 (xc5vlx155ff1760-3) FPGA device. Compared to the combination of one DP and two SP architecture built with Xilinx FP operator, the proposed adder has 11.3% faster clock frequency. On Altera Stratix-III (EP3SL340F1760C2) FPGA device, the maximum clock frequency of the proposed adder can reach 358 MHz and 1686 ALUTs and 1556 registers are occupied. The proposed adder is 11.6% faster than the combination of one DP and two SP architecture built with Altera FP megafunction.

For the reference of other researchers, the implementation results of the proposed FP multiplier and FP adder on the latest Xilinx Virtex-7 device and Altera Arria 10 device are also provided.

Acknowledgements

This project is supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Department of Electrical and Computer Engineering in University of Saskatchewan.

I wish to express my sincere thanks to my supervisor, Dr. Seok-Bum Ko, for his guidance and support during my M.Sc. program. At the beginning of my master program, Dr. Ko gave me much freedom in research topic selection. I could choose my most interested topic, computer arithmetic. During the period of my study, Dr. Ko gave me much help not only in academic research but also in daily life.

I would like to place on record my gratitude to Dr. Dongdong Chen. He helped me review the fundamentals of computer arithmetic at the beginning of my master program and throughout my program. He discussed research ideas with me every week. His experience in research really gave me great help and we have a great period of corporation.

My sincere thanks also goes to Dr. Li Chen, Dr. Francis M. Bui, Dr. Anh V. Dinh, and other professors in University of Saskatchewan who had taught me. I learned many advanced knowledges from their lectures which not only broadened my horizon but also gave me many research ideas.

Thanks to my committee members Dr. Raymond J. Spiteri, Dr. Li Chen, and Dr. Anh V. Dinh for their time and effort to review my thesis and for their suggestions to imporve the quality of this thesis.

I also thank every technical staffs in the Department of Electrical and Computer Engineering who helped me setup my research environment and helped me solve the technique issues I encountered when using the development softwares.

Finally, a big thanks to my parents, my family, and my friends for their moral support throughout the years.

This is the thesis dedicated to my beloved parents.

Contents

List of Tables

List of Figures

List of Abbreviations

ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ALUT	Adaptive Look-Up Table
ASIC	Application-Specific Integrated Circuit
ASMBL	Advanced Silicon Modular Block
CLB	Configurable Logic Blocks
CST	Carry Svae Tree
DP	Double-Precision
DSP	Digital Signal Processing
FF	Flip-Flop
FLOPS	Floating-Point Operations per Second
FMA	Fused Multiply-Add
FP	Floating-Point
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Units
GPGPU	General-Purpose Computing on Graphics Processing Units
HDL	Hardware Description Language
IP	Intellectual Property
LAB	Logic Array Block
LSB	Least Significant Bit
LUT	Look-Up Table
LZA	Leading Zero Anticipation
LZAC	Leading Zero Anticipation and Counting
MSB	Most significant Bit
NaN	Not a Number
QoS	Quality of Service
QP	Quadruple-Precision
SP	Single-Precision
SSI	Stacked Silicon Interconnect
XOR	Exclusive-OR

Chapter 1
Introduction

This chapter introduces the work proposed in this thesis. Section 1.1 presents the overview of the binary floating-point addition and multiplication operations. The motivation of this thesis work is presented in Section 1.2. Section 1.3 presents the objectives of this thesis work. Finally, the outline of the remaining chapters of this thesis is presented in Section 1.4.

1.1 Overview

Floating-Point (FP) multiplication and addition/subtraction defined in [1] are the most frequent arithmetic operations [2]. The average frequency of various FP operations relative to the total number of FP operations in the benchmark suite, which is used to evaluate the FP system performance, is shown in Figure 1.1. The FP multiplication and FP addition/subtraction occupy large portion among all FP operations. Compared to the conventional fixed-point numbers, FP numbers can provide wider dynamic range. Therefore, many applications, such as scientific computation, digital signal processing (DSP) applications, and multimedia applications, adopt FP multiplication and addition operations.

FP multiplication needs a large width multiplier to accomplish the mantissa multiplication, therefore, it is time-consuming and occupies large area. During the past few decades, many research works have been done in order to improve the performance or reduce the area of FP multiplications [3] [4] [5]. In [3], Andrew Booth proposes the Booth multiplication algorithm that greatly reduces the area of the multiplier and also improves the performance. A high performance FP multiplier designed by combining the modified Booth algorithm, the carry save adder scheme, and high speed adder is proposed in [4]. In [5], a low-power variable latency FP multiplier architecture that reduces the power consumption and average delay of

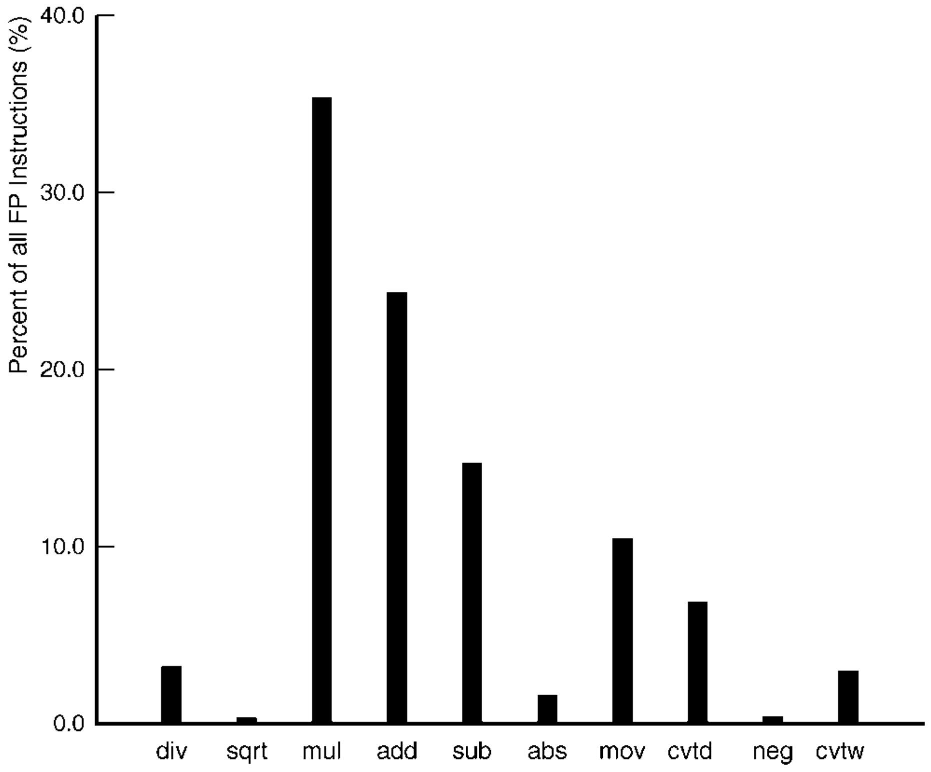


Figure 1.1: Distribution of floating-point instructions (from Fig.1 in S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," IEEE Transactions on Computers, vol.46, no.2, pp.154-161, Feb 1997)

the FP multiplier is proposed.

The FP addition operation is much more complicated than the FP multiplication. The conventional FP addition algorithm contains alignment shifter, large width mantissa adder, normalization shifter, and rounding modules and is therefore time-consuming. Many research works have been done in order to improve its performance [6–11]. In [6] [7] [8], the two-path FP addition algorithm is proposed that significantly reduces the whole latency of FP addition operation. A reduced latency FP adder that combined the rounding with the mantissa addition is proposed in [9]. In [10], a pipelined architecture of the FP adder is proposed. A delay optimized FP addition algorithm achieved by combining various optimization techniques is presented in [11].

1.2 Motivation

In multimedia applications, single-precision (SP) operations are more common. However, in recent years, double-precision (DP) and higher precision operations are getting more and more attention on scientific research [12], such as 3D and general-purpose computing on graphics processing units (GPGPU) applications. This trend is expected to continue at the upcoming big data era. On the other hand, for the graphics processing unit (GPU) in computers, when the basic display tasks are being performed, only half-precision or SP operations are required. However, when 3D gaming applications are running, DP operations are necessary. Therefore, a unified architecture for both multiplier and adder which supports multiple-precision operations is required in order to efficiently support the operations of different precisions.

Some unified architectures for FP multiplier [13] [14] [15] and FP adder [16] [17] [18] already appear in the literature. In [13], the author proposes a low-power SP and DP merged multiplier architecture. A dual-mode FP multiplier which supports both DP and quadruple-precision (QP) multiplications is proposed in [14]. In [15], a merged precision multiplier which supports SP, DP, and QP is presented. Meanwhile, in [16], the authors propose a SP and DP merged adder architecture. A dual-mode FP adder that supports both DP and QP additions is proposed in [17]. This work is further extended in [18], where a SP and DP merged adder is also implemented.

However, all these works are designed for application-specific integrated circuit (ASIC) platform that they cannot be mapped efficiently on field programmable gate array (FPGA) devices. FPGA has the advantage of reconfigurable, low power consumption [19] [20], and high performance computing [21]. As shown in Figure 1.2, the FPGA device (xc4vlx160) can operate at high speed with low energy consumption compared to CPU device (Xeon dual-core), and GPU devices (Geforce 8600GT and Tesla C1060). As shown in Table 1.1, the FPGA devices (Virtex-4 and Virtex-5) are able to achieve a much higher FP operations throughput compared to CPU device (Opteron). Moreover, current FPGA devices can provide large amount of high-speed logic resources and intellectual property (IP) cores. All of these advantages make the FPGA a good candidate for high performance applications, such

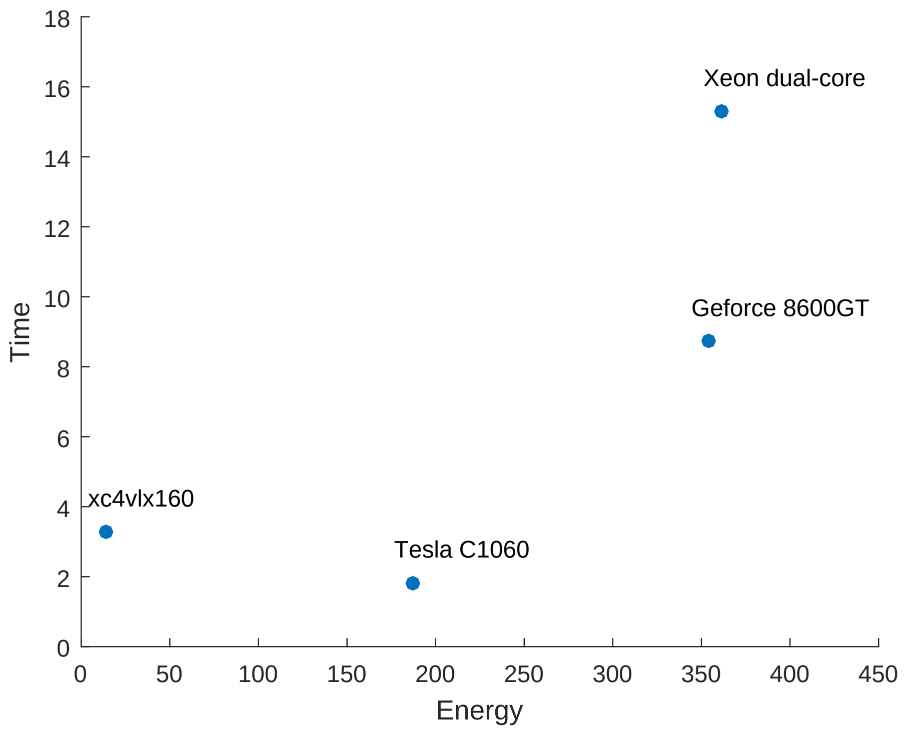


Figure 1.2: Computational time and energy consumption relationship of CPU, GPU, and FPGA devices (from A.H.T.Tse, D.Thomas, and W.Luk, "Design Exploration of Quadrature Methods in Option Pricing," IEEE Transactions on VLSI Systems, vol.20, no.5, pp.816-826, May 2012)

as scientific computation [22] and cryptography [23] [24].

In [25], the authors propose a SP and DP merged FP multiplier architecture based on the Karatsuba algorithm [26] on FPGA. On Xilinx Virtex-5 device, 9 DSP blocks are needed to implement their proposed multiplier. This amount of DSP blocks usage is large especially for some low-end FPGA devices where the number of DSP blocks is quite limited.

1.3 Objective

In this thesis, an efficient iterative SP and DP merged FP multiplier architecture for FPGA devices is proposed. This design is aimed at efficient implementation that is to reduce the resource usage as much as possible. The objectives of the multiplier design are summarized as follows:

- The proposed multiplier supports both SP and DP FP multiplication operation. The

Table 1.1: Performance comparison of double-precision operations[1][2]

	Opteron Dual-core 2.5 GHz	Virtex-4 LX200 185 MHz	Virtex-5 LX330 237 MHz
Mult/Add	10	15.9	28.0
Mult only	5	12.0	19.9
Add only	5	23.9	55.3

[1] Unit: Gflops (flops: floating-point operations per second)
[2] Data come from D. Strenski, "FPGA Floating Point Performance," HPCwire, Jan. 2007

iterative multiplication algorithm is applied so that the proposed multiplier is able to accomplish two parallel SP operations in one iteration or one DP operation in two iterations.

- The mantissa multipliers are mapped on FPGA devices using DSP blocks in order to obtain the best performance. The Karatsuba algorithm [26] is applied to further reduce the DSP block usage.

- The proposed multiplier can achieve a reduction in DSP blocks usage with satisfied performance when compared to previous work [25].

- The proposed multiplier will also be implemented on latest Xilinx Virtex-7 and Altera Arria-10 FPGA devices.

In addition, a high performance SP and DP merged FP adder is also designed and implemented on FPGA devices. The proposed adder is designed based on the two-path FP addition algorithm. The objectives of the adder design are summarized as follows:

- The proposed adder supports both one DP addition/subtraction operation or two parallel SP additions/subtractions.

- Compared to the standalone adder architecture built by combing one DP adder and two SP adders with the same data-path, the proposed adder can achieve area reduction with minor timing overhead.

- Compared to the standalone adder architecture built by combing one DP adder and two SP adders built with Xilinx FP operator [27] and Altera megafunction [28], the proposed adder is able to have speed advantage.

- The proposed adder will also be implemented on latest Xilinx Virtex-7 and Altera Arria-10 FPGA devices.

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 presents the background knowledge, including the IEEE floating-point format, the conventional floating-point multiplication and addition algorithm, and FPGA basics. Several previous works that are used as the competitors of the proposed designs are presented in Chapter 3. Chapter 4 presents the proposed floating-point multiplier. The proposed floating-point adder is presented in Chapter 5. Chapter 6 presents the implementation results of the proposed multiplier and adder, the analysis of these results, and the comparison of the proposed designs with previous works. Finally, the conclusion is drawn in Chapter 7.

Chapter 2
Background

This chapter introduces the background information and fundamental concepts in this thesis. In Section 2.1, the binary floating-point format defined in IEEE Standard 754-2008 is presented. The rounding algorithm of binary floating-point numbers is presented in Section 2.2. Section 2.3 presents the binary floating-point multiplication algorithm. In Section 2.4, the binary floating-point addition algorithm is presented. Moreover, the target FPGA architecture, including the Xilinx and Altera FPGA logic resources and DSP blocks is presented in Section 2.5. Section 2.6 summarizes the whole chapter.

2.1 Binary Floating-Point Format

The binary FP numbers defined in [1] contain three parts: 1-bit sign (S), w-bit biased exponent (E), and p-bit mantissa (M), as shown in Figure 2.1. MSB means the most significant bit and LSB means the least significant bit. Assume e is the actual exponent value, then $E = e + bias$, where $bias = 2^{w-1} - 1$. For SP numbers, $w = 8$ and $p = 23$. For DP numbers, $w = 11$ and $p = 52$.

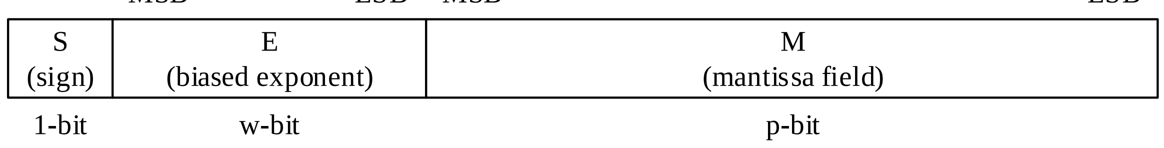


Figure 2.1: IEEE 754 floating-point format

Binary FP number in IEEE format is required to be normalized number. The significand value is always within the range $[1, 2)$. Therefore, the leading bit is hidden so that the mantissa can have 1-bit higher precision.

The IEEE FP format also reserves some special values for exceptional cases handling. The FP number T represented in IEEE format and its actual value v can be summarized as follows:

- If $E = 2^w - 1$ and $M \neq 0$, then T is not a number (NaN).
- If $E = 2^w - 1$ and $M = 0$, then T is infinity (Inf) and $v = (-1)^S \times (+\infty)$.
- If $1 \leq E \leq 2^w - 2$, then T is normal number and $v = (-1)^S \times (1.M) \times 2^{E-bias}$. Normal number has the implicit leading bit of 1.
- If $E = 0$ and $M \neq 0$, then T is subnormal number and $v = (-1)^S \times (0.M) \times 2^{1-bias}$. Subnormal number has the implicit leading bit of 0.
- If $E = 0$ and $M = 0$, then T is zero and $v = (-1)^S \times (+0)$.

2.2 Binary Floating-Point Rounding

The results of the FP operations must fit in the IEEE FP format. Sometimes there are more bits than the FP format allowed in the results. In these cases, the rounding operation must be performed in order to make the result be able to be represented by the IEEE format. Rounding is to represent the infinite precision result by choosing a representable finite precision number that is closest to the result.

In IEEE standard 754-2008 [1], five rounding schemes are proposed:

- **roundTiesToEven**: the FP number nearest to the infinite precision result should be chosen. If nearest FP numbers are equally near, the one with the even LSB should be chosen.
- **roundTiesToAway**: the FP number nearest to the infinite precision result should be chosen. If nearest FP numbers are equally near, the one with larger magnitude should be chosen.

- **roundTowardPositive**: the FP number closest to and no less than the infinite precision result should be chosen.
- **roundTowardNegative**: the FP number closest to and no greater than the infinite precision result should be chosen.
- **roundTowardZero**: the FP number closest to and no greater in magnitude than the infinite precision result should be chosen.

The roundTiesToEven scheme is the default rounding scheme.

2.3 Binary Floating-Point Multiplication

2.3.1 General FP Multiplication Algorithm

When performing the multiplication operations, the sign, exponent, and mantissa need to be processed separately.

For sign processing, the sign of the product (S_p) is the Exclusive-OR (XOR) of the signs of the two operands (S_1 and S_2):

$$S_p = S_1 \oplus S_2 \tag{2.1}$$

For exponent processing, the exponents of the two operands (E_1 and E_2) are added together. As discussed in Section 2.1, the exponent in IEEE format is biased exponent. In order to keep the exponent of the product (E_p) conform to the IEEE format, the bias should be subtracted from the summation:

$$E_p = E_1 + E_2 - bias \tag{2.2}$$

The exponent obtained here may need further adjustment when normalizing the mantissa of the product.

For mantissa processing, the mantissas of the two operands (M_1 and M_2) are prefixed with the leading bits. Then the mantissa of the product (M_p) can be obtained by performing unsigned fixed-point multiplication on the two prefixed mantissas:

$$M_p = M_1 \times M_2 \tag{2.3}$$

As discussed in Section 2.1, M_1 and M_2 are within the range $[1, 2)$. Therefore, the mantissa of the product M_p is in the range $[1, 4)$. In other words, no more than 1-bit normalization shifting is needed for M_p. If M_p is within the range $[1, 2)$, then no further operations are needed. Otherwise, if M_p is larger than 1, then 1-bit right shift is needed for M_p and at the same time, E_p should be added by 1.

The normalized mantissa is then rounded to reserve the most significant 24-bit (including the leading bit). If overflow (rounded M_p is larger than 1) occurs, further normalization and exponent adjustment is needed. Finally, S_p, E_p, and M_p (discarding the leading bit) are packed according to the IEEE FP format.

For large width mantissas, direct multiplication is not efficient. In order to improve the performance, the recursive method is proposed. Moreover, the Karatsuba algorithm [26] is proposed based on the recursive method to further improve the performance of the large multiplication. On the other hand, in order to reduce the area, the iterative method is proposed. The following three sub-sections present these three methods.

2.3.2 Recursive Multiplication

In order to compute $A \times B$, the two operands A and B are first divided into two parts (with arbitrary width) A_1, A_2, B_1, and B_2. Then the product of $A \times B$ can be obtained by combining the results of $A_1 \times B_1$, $A_1 \times B_2$, $A_2 \times B_1$, and $A_2 \times B_2$ as shown in Figure 2.2.

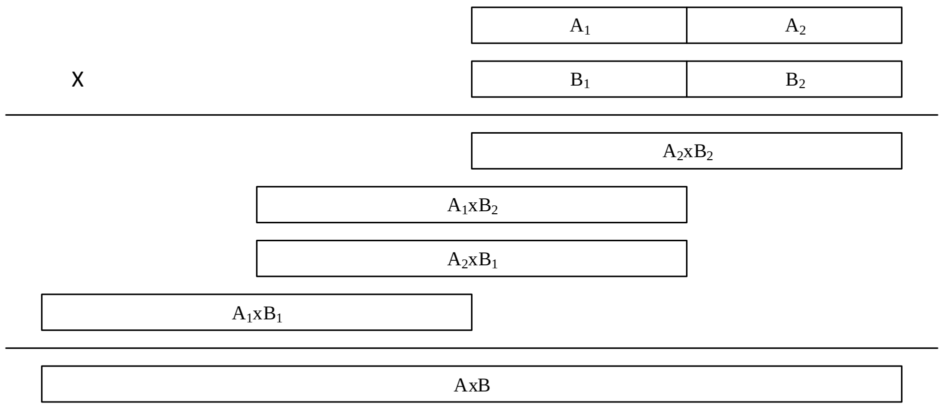


Figure 2.2: Recursive multiplication

By implementing parallel smaller width multiplications, the performance of large width multiplication can be improved.

2.3.3 Karatsuba Algorithm

Assume the two operands are A and B, they can be rewritten as equation (2.4):

$$\begin{aligned} A &= A_1 \times 2^n + A_2 \\ B &= B_1 \times 2^n + B_2 \end{aligned} \tag{2.4}$$

where A_2 and B_2 are the least significant n-bit of A and B, respectively. And A_1 and B_1 are the remaining part of A and B.

According to the Karatsuba algorithm [26], $A \times B$ can be calculated as equation (2.5):

$$\begin{aligned} A \times B &= (A_1 \times 2^n + A_2)(B_1 \times 2^n + B_2) \\ &= F_1 \times 2^{2n} + (F_1 + F_3 - F_2) \times 2^n + F_3 \end{aligned} \tag{2.5}$$

where F_1, F_2, and F_3 are shown in equation (2.6):

$$\begin{aligned} F_1 &= A_1 \times B_1 \\ F_2 &= (A_1 - A_2)(B_1 - B_2) \\ F_3 &= A_2 \times B_2 \end{aligned} \tag{2.6}$$

Therefore, only three multiplications instead of four in recursive method are needed to accomplish a large width multiplication. This algorithm is especially helpful when doing the multiplication operation on FPGA using DSP blocks. With reduced number of multiplications, the DSP blocks usage can also be reduced.

2.3.4 Iterative Multiplication

In order to compute $A \times B$, the two operands A and B are first divided into two parts (with equal width) A_1, A_2, B_1, and B_2. Because A_1 and A_2 (B_1 and B_2) have the same width, $A_1 \times B_1$ and $A_1 \times B_2$ can be finished using the same hardware. Similarly, $A_2 \times B_1$ and $A_2 \times B_2$ can be done with the same hardware. Therefore, only two multipliers are built and these two multipliers can be reused in different iterations. As shown in Figure 2.3, in the first iteration, $A_1 \times B_2$ and $A_2 \times B_2$ are computed and in the second iteration, $A_1 \times B_1$ and $A_2 \times B_1$ are computed.

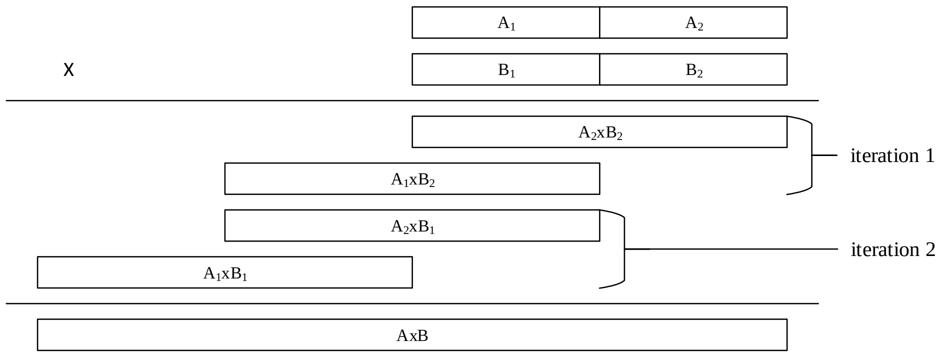


Figure 2.3: Iterative multiplication

By using the iterative method, the area of the multiplier can be significantly reduced.

In the proposed multiplier architecture, the recursive method, the Karatsuba algorithm, and the iterative method are combined with the general FP multiplication algorithm. This will be discussed in detail in Chapter 4.

2.4 Binary Floating-Point Addition

2.4.1 One-Path Algorithm

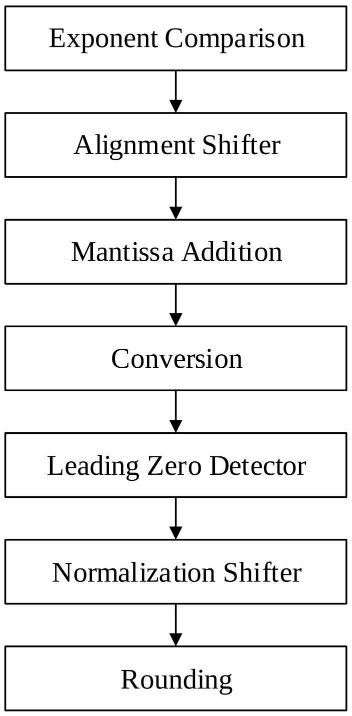
The diagram of conventional one-path FP addition algorithm is shown in Figure 2.4a. In order to compute $a \pm b$, seven major steps need to be executed:

1. Exponent comparison: Subtract the exponents $|e_a - e_b| = d$ and swap the mantissa accordingly. Denote the larger exponent as e_{base}.

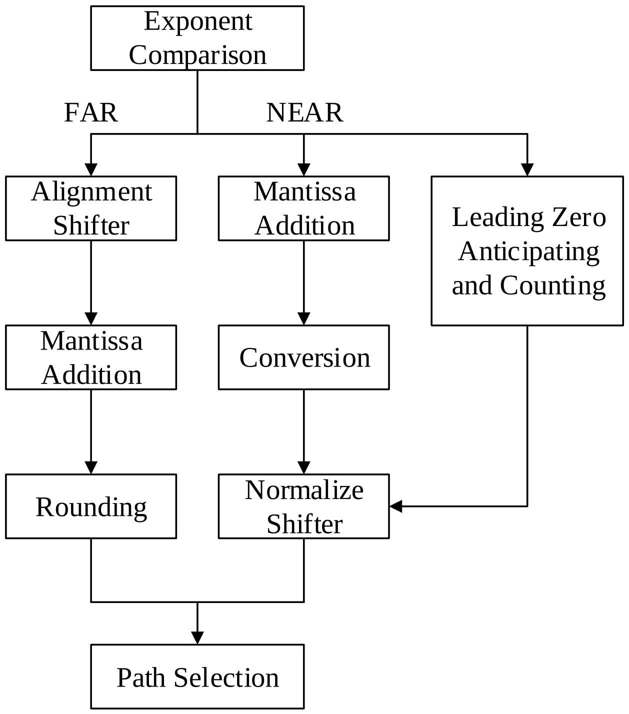
2. Alignment shifting: Right shift the mantissa of the second operand by d-bit, so that the two operands now have the same exponent e_{base}.

3. Mantissa addition: Execute addition or subtraction to the two operands according to the effective operation *eop*. *eop* can be determined by the sign of the two operands and the actual operation *op*.

4. Conversion: Convert the result if it is negative.



(a) One-path algorithm



(b) Two-path algorithm

Figure 2.4: Binary floating-point addition algorithms

5. Leading zero detection (LZD): Detecting the number n of zeros in front of the most significant one.

6. Normalization shifting: Left shift the result by n-bit and add n to e_{base}.

7. Rounding: Round the normalized result according to one of the rounding algorithm defined in [1].

2.4.2 Two-Path Algorithm

The steps in the conventional one-path algorithm are executed in sequential which makes the FP addition time-consuming. In order to improve the performance of the FP addition, in [6] [8] [29], the two-path addition algorithm was proposed. The data-flow, as shown in Figure 2.4b, is split into two path, the NEAR path and the FAR path. The result of the NEAR path is selected when d is 0 or 1 and eop is subtraction. In other cases, the result from the FAR path is selected.

This algorithm is developed based on the following analysis:

- First, when d is 0 or 1 (NEAR path), a long normalization shifter may be needed, however, a 1-bit shifter is enough for alignment. On the other hand, when d is larger than 1 (FAR path), a long alignment shifter may be needed, however, the normalization shifting is within 1-bit. Therefore, one of the two long shifters is removed from the critical path.

- Second, the negative result occurs only when d is 0. In this case, no rounding is needed, therefore, the conversion and rounding are mutually exclusive.

The two-path algorithm can significantly improve the performance of FP addition at the expense of larger area.

The proposed adder architecture is built based on the two-path FP addition algorithm. The detail of the proposed FP adder will be discussed in Chapter 5.

2.5 Target FPGA Architecture

In this thesis, the proposed multiplier architecture is implemented on Xilinx Virtex-7 and Altera Arria-10 FPGA devices. In order to compare with the previous work, the proposed multiplier is also implemented on Xilinx Virtex-5 FPGA device. Meanwhile, the proposed adder architecture is implemented on Xilinx Virtex-5 and Altera Stratix-III FPGA devices. Moreover, the implementation results of the proposed adder on Xilinx Virtex-7 and Altera Arria-10 devices are also provided.

In this section, the architectures of Xilinx Virtex-5 and Virtex-7 and Altera Strtix-III and Arria-10 are presented.

2.5.1 Xilinx Virtex-5 and Virtex-7

Configurable Logic Blocks (CLBs)

The configurable logic blocks (CLBs) are the resources for implementing logic functions on Xilinx FPGA devices. Each CLB contains two slices. The diagram of the slice on Virtex-

5 devices is shown in Figure 2.5. Each slice contains 4 look-up tables (LUTs), 4 storage elements, the carry logic, and some multiplexers [30].

LUTs are used to implement the logic functions. Virtex-5 devices adopt 6-input LUTs and each LUT has 6 independent inputs and two independent outputs. One LUT can either implement one 6-input boolean logic functions or two 5-bit boolean logic functions. One LUT is fully utilized when all its 6-input are used.

Each slice also contains 4 storage elements. They can be configured to flip-flops (FFs). The FFs can be used as input/output registers in synchronous designs or pipeline registers in pipelined designs.

Another important component of the slice is the carry logic. The dedicated carry logic can perform fast addition and subtraction in slices. The adder can be described using behavior level model in hardware description language (HDL) code and the FPGA tools will map the adder to the dedicated carry logic. The carry lookahead adder using the dedicated carry logic has the best performance over other kinds of adders on FPGA [31]. This is different from ASIC design. In ASIC design, some fast adders, for example parallel prefix tree adders, have better performance than carry lookahead adders. However, if the prefix tree adder is implemented on FPGA, they will be mapped to LUTs which is much slower than the dedicated carry logic.

The CLBs and LUTs in Virtex-7 devices [32] have similar architectures as those of Virtex-5 devices. One of the differences is that it has 8 storage elements in each LUT. Some other new technologies, for example the advanced silicon modular block (ASMBL) and stacked silicon interconnect (SSI), will not be discussed here.

DSP Blocks

In this thesis, the DSP blocks are only used in the proposed multiplier design in order to get the best performance. Because the multiplier built by DSP blocks is much faster than that built by LUTs. The DSP blocks are not used in the proposed adder design, because the dedicated carry logics are already able to provide satisfied performance.

The diagram of DSP48E blocks in Virtex-5 devices is shown in Figure 2.6. By assigning different OPMODE and ALUMODE value, DSP48E can accomplish various arithmetic op-

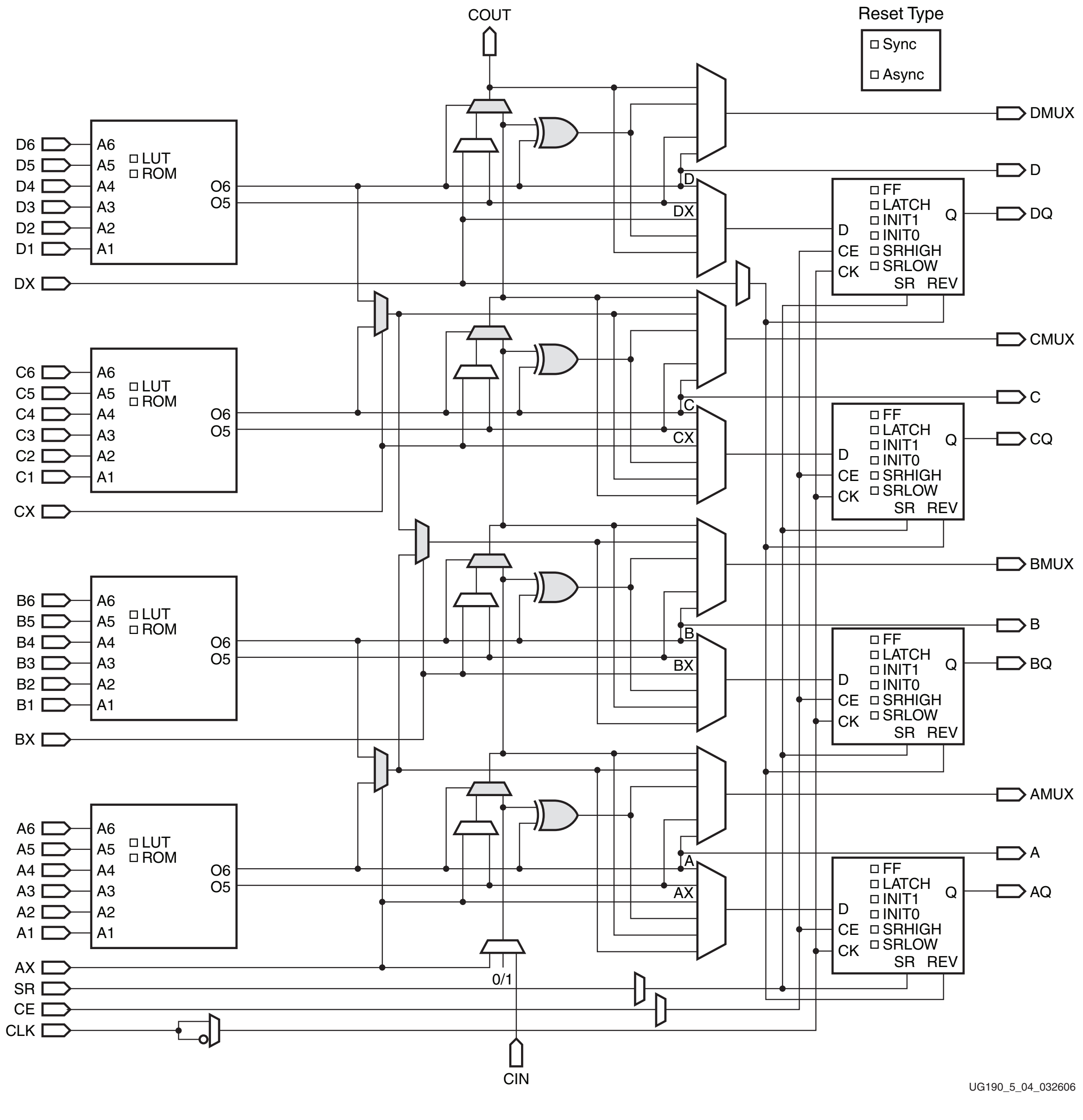


Figure 2.5: Slice diagram of Virtex-5 devices (from Virtex-5 FPGA User Guide (UG190(v5.4)), Mar. 2012, p.176)

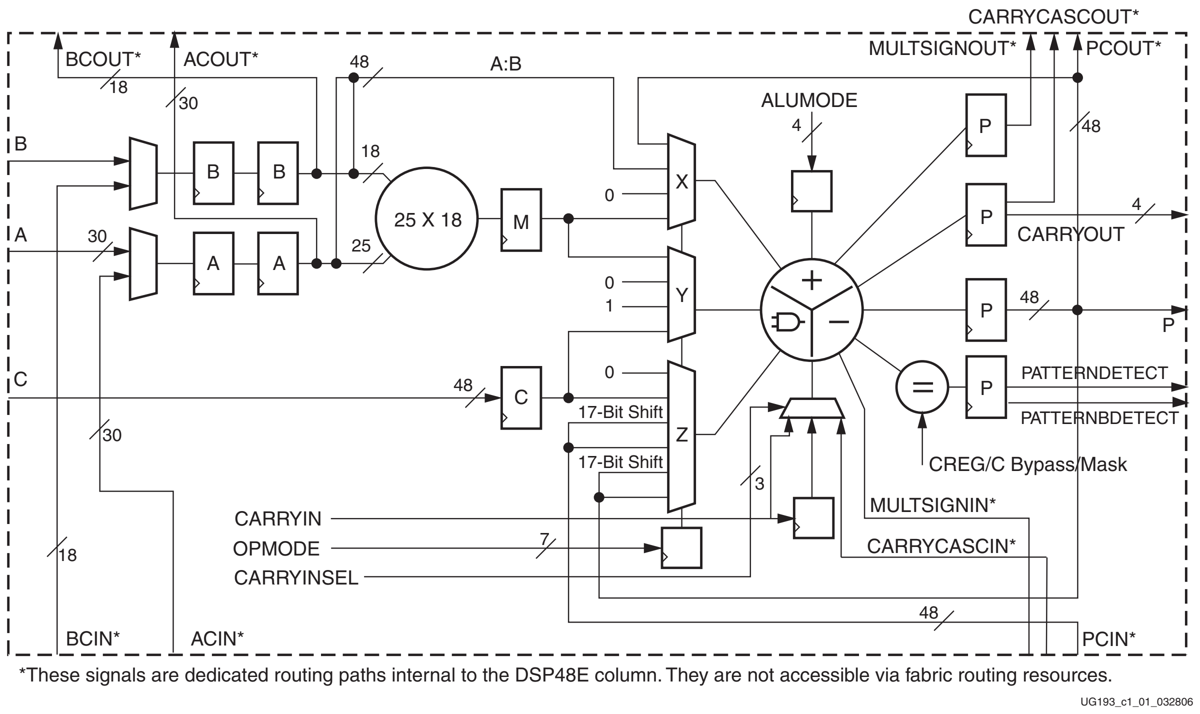


Figure 2.6: Diagram of Virtex-5 DSP48E blocks (from Virtex-5 FPGA XtremeDSP Design Considerations User Guide (UG193(v3.5)), Jan. 2012, p.14)

erations. DSP48E supports 25×18 two's complement multiplication [33]. In multiplication mode, A and B are the input ports for the two input operands. P is the output port. DSP48E also provides cascaded data input port PCIN and cascaded output port PCOUT. PCIN can receive the data from PCOUT of the previous DSP48E. Similarly, PCOUT can send the data to PCIN of the next DSP48E. The cascade connection is used to calculate the $F_1 + F_3 - F_2$ as discussed in Section 2.3.3.

In order to get the maximum speed, three stages pipeline registers are required for the multiplier based designs [33]. With the speed grade of -3, the clock frequency of DSP48E in multiplier mode can be up to 550 MHz [34].

There are two methods to use the DSP blocks. One is to use behavior level model to implicit the multiplier in HDL code. The FPGA synthesis tools will automatically detect the multiplication operation and map it to the DSP blocks. Another way is to use the instantiation template provided by Xilinx [35]. The mapping using this method is more accurate. In the proposed multiplier design, the DSP instantiation template is used in order

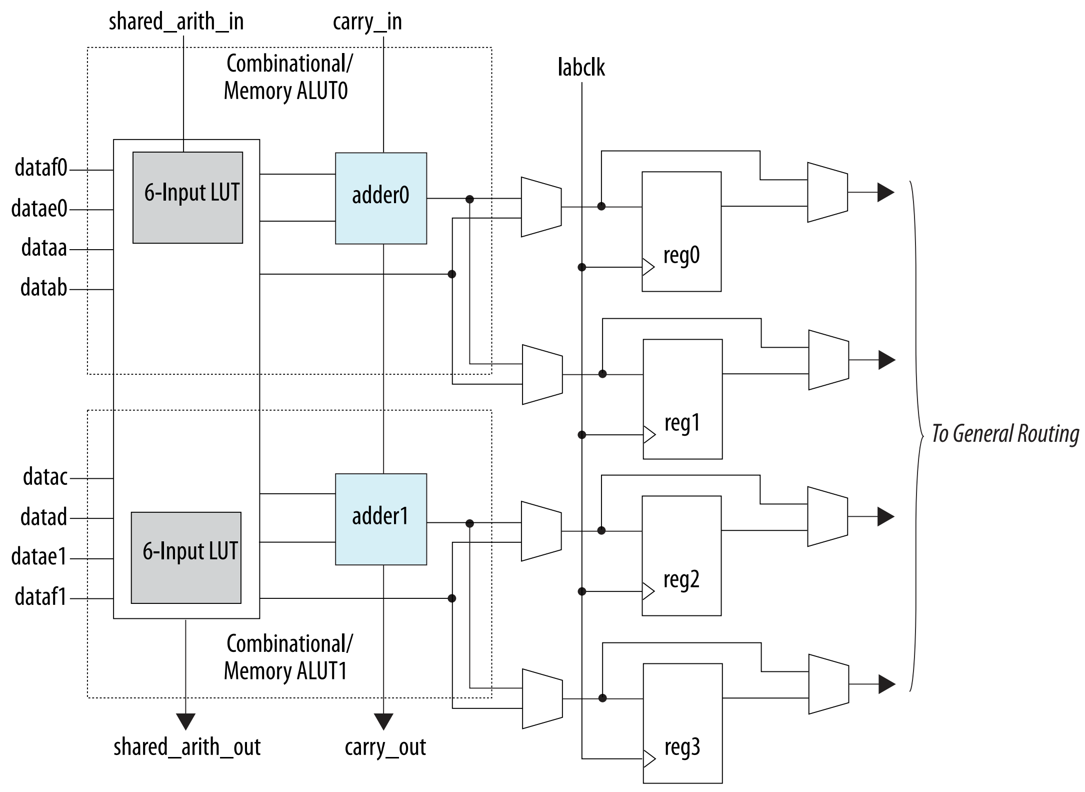


Figure 2.7: ALM diagram of Arria-10 devices (from Arria 10 Core Fabric and General Purpose I/Os Handbook, May. 2015, p.1-7)

to ensure the mapping is exactly the same as designed.

DSP48E1 in Virtex-7 devices has the same data-path in multiplier mode [36]. The new functions, for example the pre-addition, are not used in the proposed design and they will not be discussed here.

2.5.2 Altera Stratix-III and Arria-10

Logic Array Blocks (LABs)

The logic array blocks (LABs) are the logic functions generators on Altera FPGA devices. Each LAB contains 10 adaptive logic modules (ALMs). The diagram of the ALM on Arria-10 devices is shown in Figure 2.7. Each ALM contains 2 adaptive look-up tables (ALUTs), 4 registers, 2 dedicated full adders, carry chain, and shared arithmetic chain [37].

ALUT has similar functions as the LUT in Xilinx devices. The two ALUTs in one ALM have a total of 8-input. Therefore, one ALM can either implement one 6-input or 7-input function or various combinations of two functions. The ALM architecture is backward-compatible with 4-input LUT architectures in the older product series.

Similar to the storage elements in Xilinx FPGA slices, the four registers in one ALM can be used as input/output registers or pipeline registers.

The dedicated carry chain can perform fast addition and subtraction in ALMs.

The ALM has four operating modes [37]:

- Normal Mode: Allow two functions to be implemented in one ALM, or a single function of up to 6 inputs.
- Extended LUT Mode: Allow 7-input functions to be implemented in one ALM.
- Arithmetic Mode: Allow two sets of two 4-input LUTs along with two dedicated full adders. Each adder can add the output of two 4-input functions. The carry chain works in this mode.
- Shared Arithmetic Mode: Allow 3-input addition in ALM by using shared arithmetic chain.

The LABs, ALMs, and ALUTs in Stratin-III devices have similar architectures and functions with those in Arria-10 devices except that there is only two registers in each ALM.

Native Fixed-Point DSP IP Core

In Arria-10 devices, although the hardened floating-point DSP blocks are provided, only the SP operations are supported [37]. In the proposed multiplier, in order to support both SP and DP operations in a unified architecture, the native fixed-point DSP blocks [38] is adopted.

The fixed-point DSP blocks in Arria-10 devices support 27×27 unsigned multiplication [38]. The diagram of Arria-10 DSP in fixed-point 27×27 mode is shown in Figure 2.8. In this mode, $result = dataa\_x0 \times dataa\_y0$.

In order to get the maximum speed, three stages pipeline registers are required in 27×27 mode. Under normal working voltage, the DSP in the device with extended operation temperature, standard power, and speed grade of 1 can run up to 541 MHz [39].

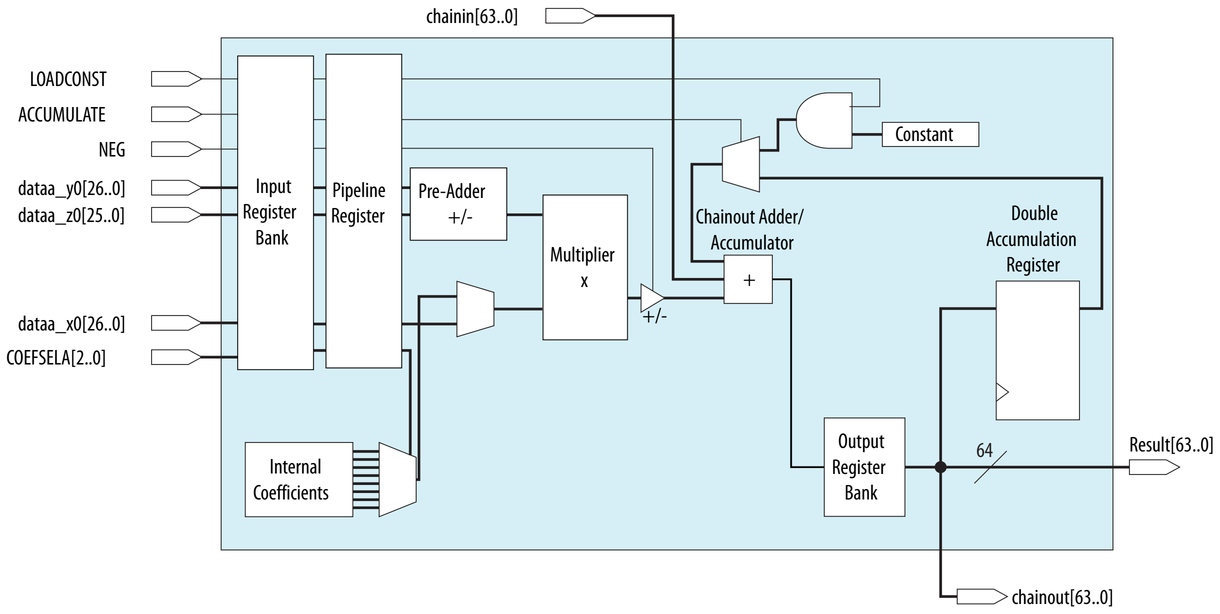


Figure 2.8: Fixed-Point 27 × 27 mode in Arria-10 DSP (from Arria 10 Core Fabric and General Purpose I/Os Handbook, May. 2015, p.3-11)

There are two methods to use the DSP blocks in Altera devices. One is the same as that of Xilinx DSP, that is to implicit the multiplier in HDL code. Another method is to use the IP Catalog and Parameter Editor provided in Quartus II software [38]. In order to ensure the mapping is exactly the same as designed, the IP Catalog tool is used in the proposed multiplier design.

Only the Arria-10 device is used to implement the proposed multiplier architecture, therefore, the DSP blocks in Stratix-III device is not discussed here.

2.6 Summary

In this chapter, the background information and fundamental concepts related to this thesis are presented.

The binary floating-point format defined in IEEE Std 754-2008 is first introduced. It contains three components: sign, exponent, and mantissa. The actual value of the number represented in IEEE format is discussed, including normal number, zero, subnormal number, NaN, and Inf.

There are five rounding schemes defined in IEEE Std 754-2008: roundTiesToEven, roundTiesToAway, roundTowardPositive, roundTowardNegative, and roundTowardZero. The roundTiesToEven is the default rounding scheme.

When implementing FP multiplication, the three components need to be processed separately. For large width multiplication, the recursive method, the Karatsuba algorithm, and the iterative method can be applied in order to efficiently implement the large multiplication.

The conventional FP addition algorithms include the one-path algorithm and the two-path algorithm. The two-path algorithm can significantly improve the performance of FP adder but at the expense of larger area.

Finally, the architecture of the LUTs resource and DSP blocks in Xilinx Virtex-5 and Virtex-7 and Altera Stratix-III and Arria-10 devices are presented.

Chapter 3
Previous Works

This chapter briefly presents two previous works [25] and [40], and the DP multiplication solution for Xilinx Virtex-5 FPGA provided in [33]. The proposed FP multiplier is to be compared with these three works in order to show its advantages. For the proposed FP adder design, to the best of our knowledge, there is no published work on such SP and DP merged adder designed on FPGA devices. The proposed FP adder is to be compared with the Xilinx FP operator [27] and Altera FP megafunction [28]. The architectures of FP solutions in Xilinx FP operator and Altera FP megafunction are encrypted and will not be discussed in this thesis.

3.1 Double-Precision Multiplication Solution on Xilinx Virtex-5 Device

In [33], the DP multiplication solution for Xilinx Virtex-5 FPGA using DSP48E blocks is provided. A 59×59 signed multiplier is built to implement the 53×53 mantissa multiplication. Because the FP mantissa multiplication is an unsigned multiplication, the extra sign bits are all set to zero.

The 59×59 signed multiplier is implemented with 10 DSP48E blocks. The diagram of the lower 5 DSP blocks is shown in Figure 3.1 and that of the upper 5 DSP blocks is shown in Figure 3.2. As shown in Figure 3.1 and Figure 3.2, the whole 59×59 multiplication is divided into 10 smaller multiplications, each of which can be handled by one DSP block. These 10 DSP blocks are cascaded using the internal data bus between each two DSP blocks. These internal data buses are only available to DSP blocks and cannot be accessed through

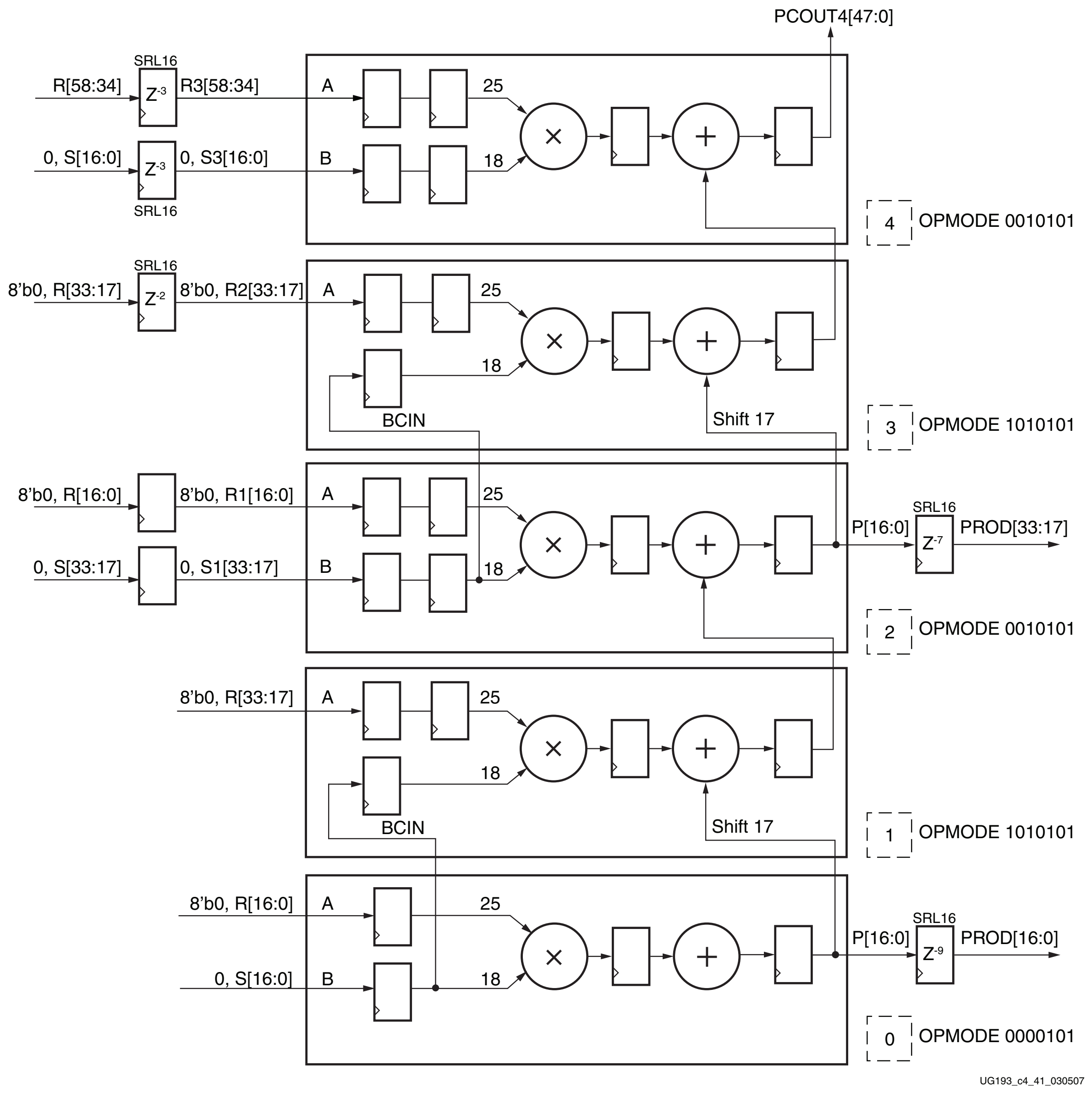


Figure 3.1: 59×59 signed multiplier implemented with 10 DSP48E blocks: Lower five DSP blocks (from Virtex-5 FPGA XtremeDSP Design Considerations (UG193(v3.5)), Jan. 2012, p.72)

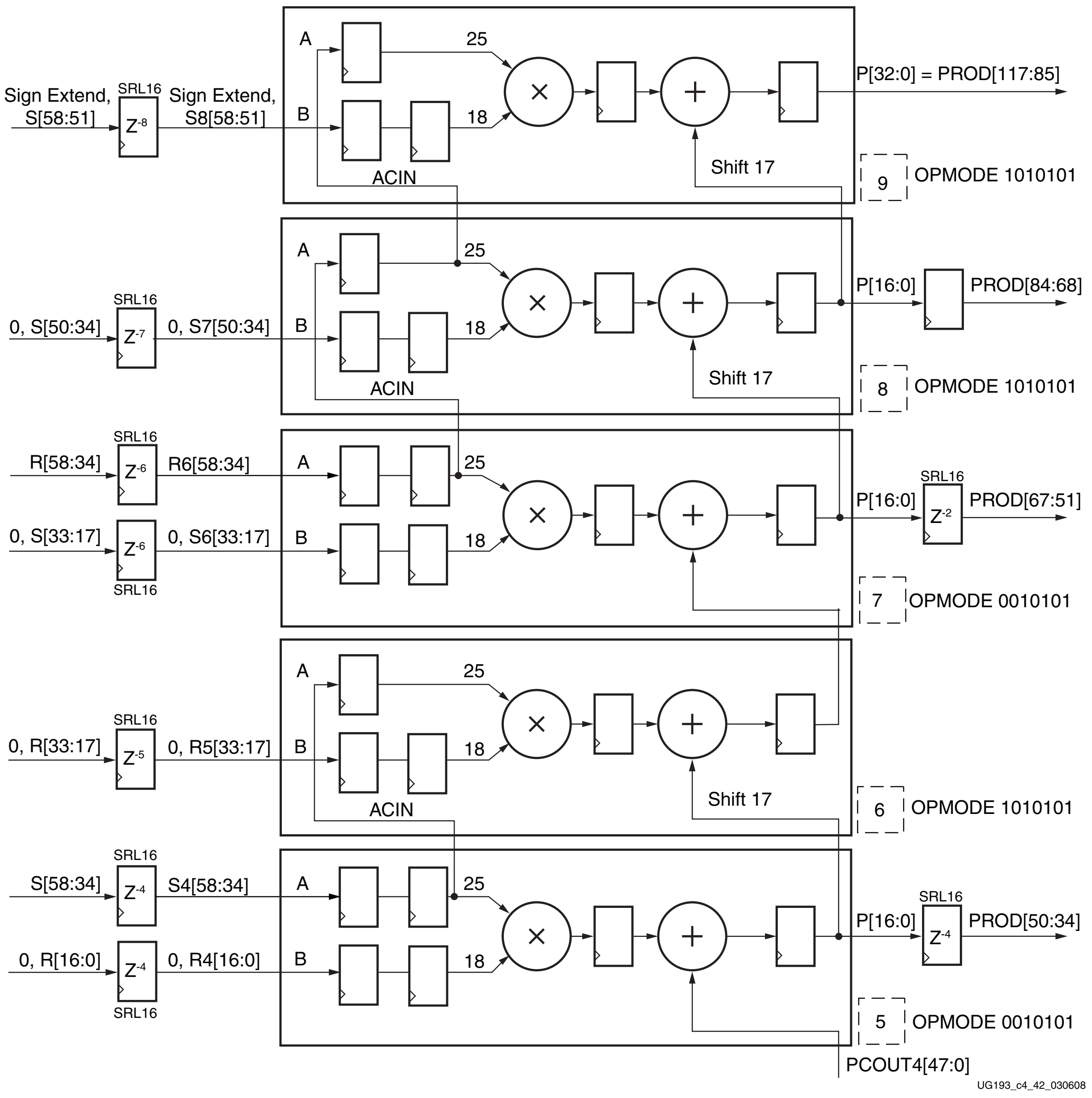


Figure 3.2: 59 × 59 signed multiplier implemented with 10 DSP48E blocks: Higher five DSP blocks (from Virtex-5 FPGA XtremeDSP Design Considerations (UG193(v3.5)), Jan. 2012, p.73)

LUT resources. In order to obtain the best performance of the DSP blocks, all the embedded pipeline registers are utilized. In addition, several external registers are used in order to maintain the data coherence of the pipeline design. At the end of the 12th clock cycles, the complete product of 59×59 multiplication can be obtained. In DP multiplication operation, this product needs to be normalized and rounded in the following steps. These steps contain another 6 pipeline stages. Therefore, a total of 18 clock cycles are needed for a DP multiplication operation.

The sign and exponent processing of this DP multiplication solution is implemented on LUT resources.

The implementation results show that this design needs a total of 10 DSP blocks, 339 LUTs, and 482 FFs. The maximum clock frequency runs at 319 MHz and the complete DP multiplication operation needs 18 clock cycles to accomplish.

This DP solution is basically mapping the DP multiplier on the FPGA devices straightforwardly. It occupies large amount of DSP blocks and is therefore not efficient. Besides, this solution only supports DP operation. If the SP operation is required, another standalone SP multiplier should be built. Unless the SP operands extend to DP numbers to perform DP multiplication operation. However, either way is not efficient.

3.2 Double-Precision Multiplication on FPGA using Tiling Technique

In [40], the authors propose a DP multiplier architecture for Xilinx FPGA devices using their proposed tiling technique.

For the multiplication of double-precision or beyond, the multiplication is first divided into several smaller multiplications to make them fit in the capability of DSP blocks. Each of the smaller multiplication is done by one DSP blocks. When adding these partial products together, the same DSP blocks are used. The internal data bus between two DSP blocks can shift the previous result by 17-bit. The partial product from the previous DSP block is sent to the next DSP block using the internal data bus and it is added to the result generated by the next DSP block. Therefore, a large amount of LUT resources can be saved.

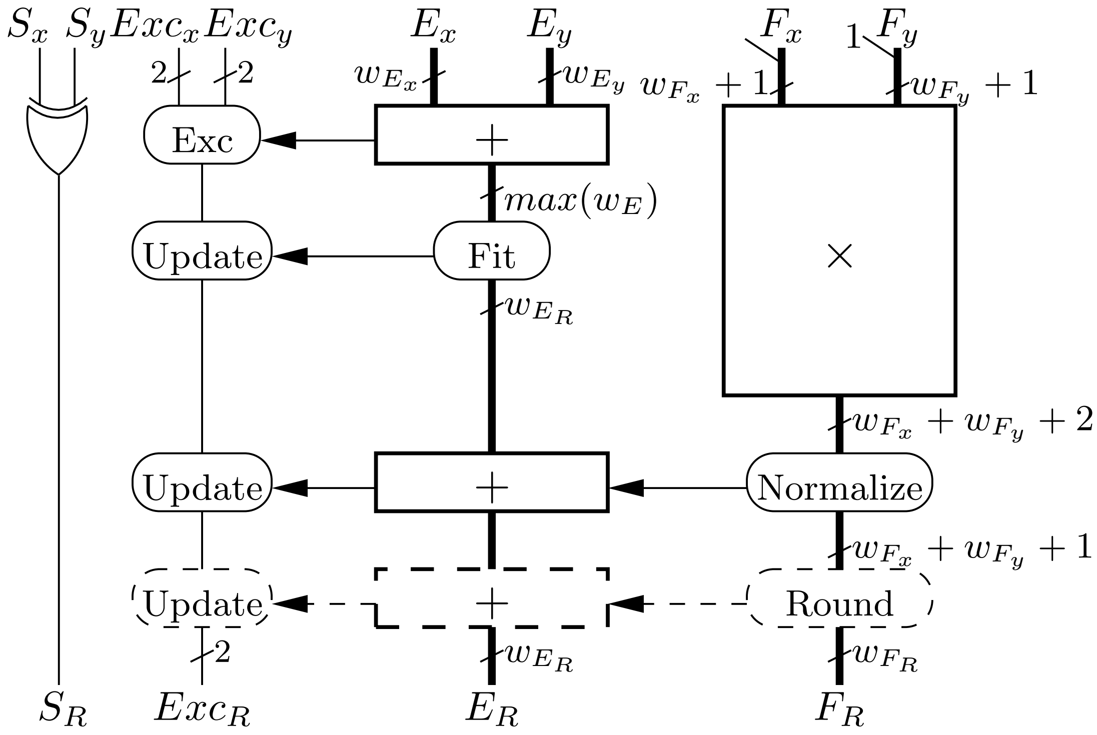


Figure 3.3: Architecture of the floating-point multiplier (from S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for Floating-Point Double-Precision and Beyond on FPGAs," SIGARCH Computer Architecture News, vol.38, no.4, pp. 73-79, Jan. 2011)

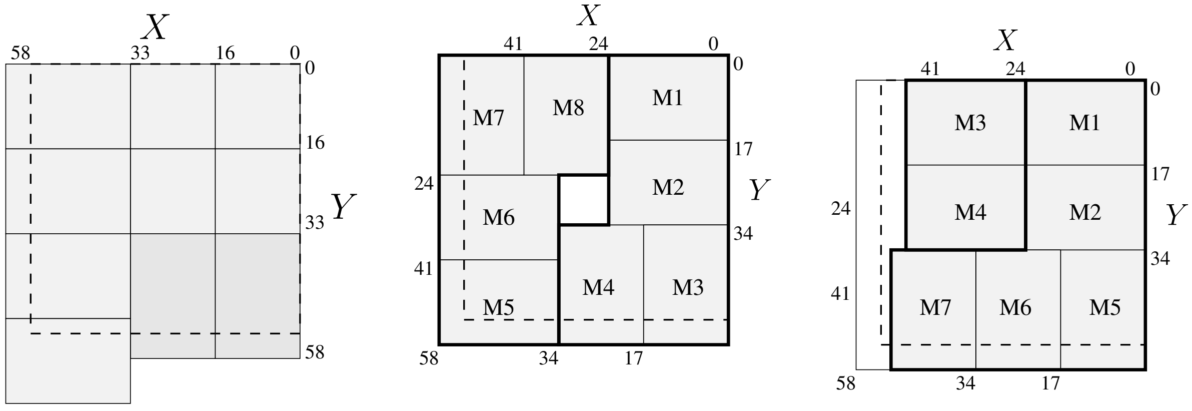


Figure 3.4: Mantissa multiplier using DSP blocks (from S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for Floating-Point Double-Precision and Beyond on FPGAs," SIGARCH Computer Architecture News, vol.38, no.4, pp. 73-79, Jan. 2011)

The diagram of the multiplier architecture presented in [40] is shown in Figure 3.3. The mantissa multiplier is realized by tiling technique and the diagram of the mantissa multiplier is shown in Figure 3.4. The first one is the tiling method used by Xilinx tools. The second one is the authors' previous work and the third one is the tiling diagram presented in [40]. The dash square in Figure 3.4 is the resource required by the mantissa multiplication. Each of the rectangle is one DSP block. The space outside the DSP means these functions have to be realized by LUT resource. The DSP blocks are arranged in this format so that the results from one DSP blocks can be sent to the next one using internal data bus. For the mantissa multiplication, 7 DSP blocks are needed.

For the whole multiplier, when the latency is set to 14 clock cycles, this multiplier can run up to 407 MHz, using 804 LUTs, 804 FFs, and 9 DSP blocks. When the latency is 13 clock cycles, the maximum clock frequency remains 407 MHz, however, 1184 LUTs, 1080 FFs, and 9 DSP blocks rare required.

In this work, the tiling technique is used in order to make fully use of the DSP blocks. The DSP blocks required by the DP multiplication operation are reduced. However, this design still only supports DP operations and is therefore not efficient to implement SP operations.

3.3 Double-Precision Multiplier on FPGA with Dual Single-Precision Support

In [25], the authors propose a SP and DP merged FP multiplier for FPGA device. Their proposed multiplier supports one DP operation or two parallel SP operations.

The diagram of their proposed architecture is shown in Figure 3.5. One 66×66 multiplier is implemented to realize the mantissa multiplication. With two partition Karatsuba algorithm [26], the 66-bit multiplier can be implemented using two 33-bit multipliers ($m00$ and $m11$ in Figure 3.5) and one 34-bit multiplier ($m10$ in Figure 3.5). As a result, the multiplier supports both DP multiplication and SP multiplications. In SP mode, the first SP operation is done by $m00$, and the second SP operation is done by $m11$. In DP mode, all these three multipliers are used to calculate the DP mantissa result. The results from these multipliers are combined using additional adders and subtractors built using LUT resources to generate

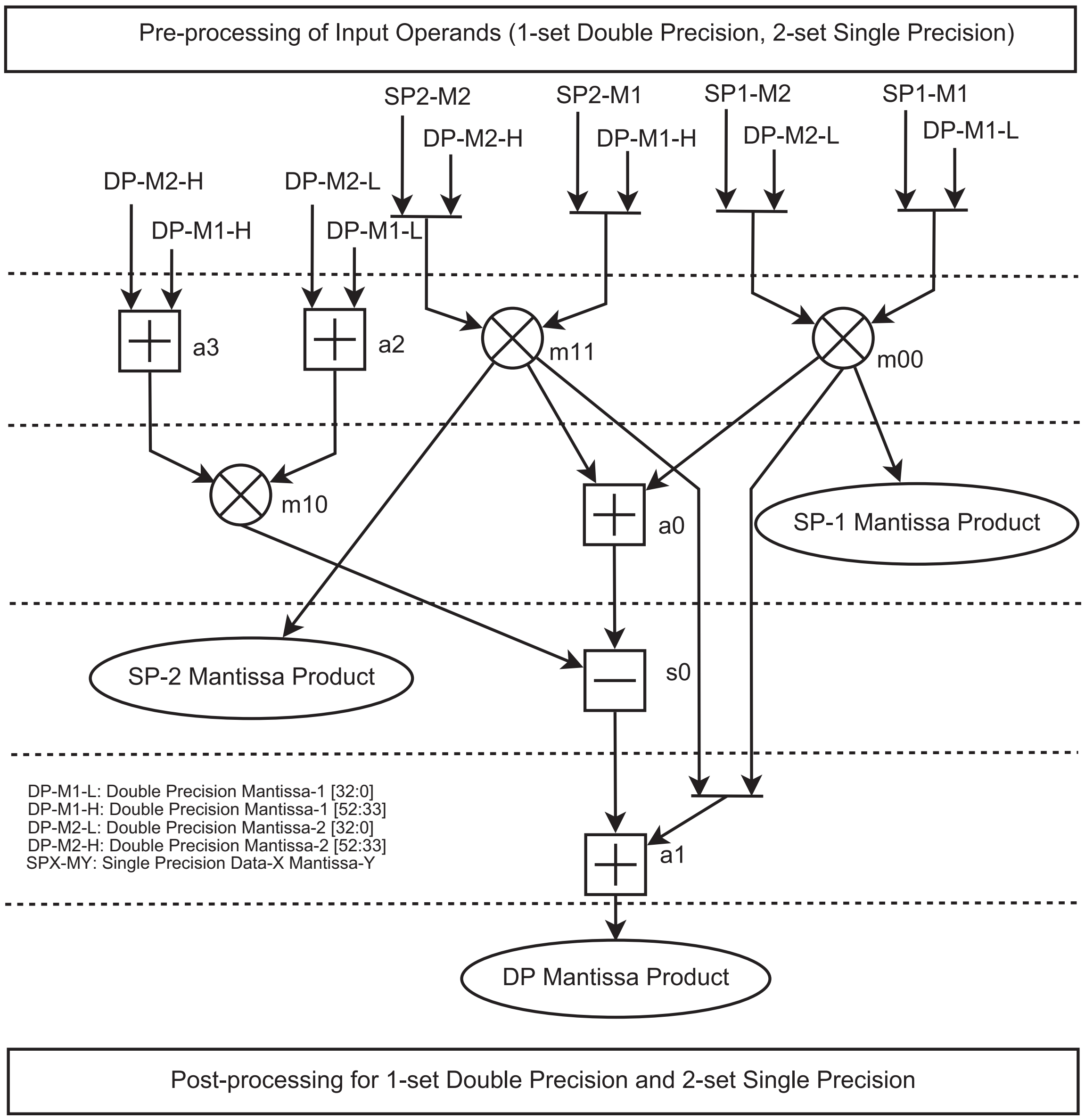

Pre-processing of Input Operands (1-set Double Precision, 2-set Single Precision)
SP2-M2
DP-M2-H
SP2-M1
DP-M1-H
SP1-M2
DP-M2-L
SP1-M1
DP-M1-L
DP-M2-H
DP-M1-H
DP-M2-L
DP-M1-L
a3
a2
m11
m00
m10
a0
SP-1 Mantissa Product
SP-2 Mantissa Product
s0
DP-M1-L: Double Precision Mantissa-1 [32:0]
DP-M1-H: Double Precision Mantissa-1 [52:33]
DP-M2-L: Double Precision Mantissa-2 [32:0]
DP-M2-H: Double Precision Mantissa-2 [52:33]
SPX-MY: Single Precision Data-X Mantissa-Y
a1
DP Mantissa Product
Post-processing for 1-set Double Precision and 2-set Single Precision

Figure 3.5: DP and dual SP floating-point multiplier architecture (from M. K. Jaiswa and R. C. C. Cheung, "Area-efficient architecture for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support," Microelectronics Journal, vol.44, no.5, pp. 421-430, 2013)

the DP mantissa result.

In order to use DSP blocks, the two 33-bit multipliers and the 34-bit multiplier are again implemented using two partition Karatsuba algorithm. The diagram of the 34-bit multiplier is shown in Figure 3.6. The 33/34-bit multiplier needs two 17-bit multiplier and one 18-bit multiplier. Each of the 17 or 18-bit multiplier is done by one DSP block. A total of three DSP blocks are required for one 33 or 34-bit multiplier. Therefore, the whole multiplier needs 9 DSP blocks.

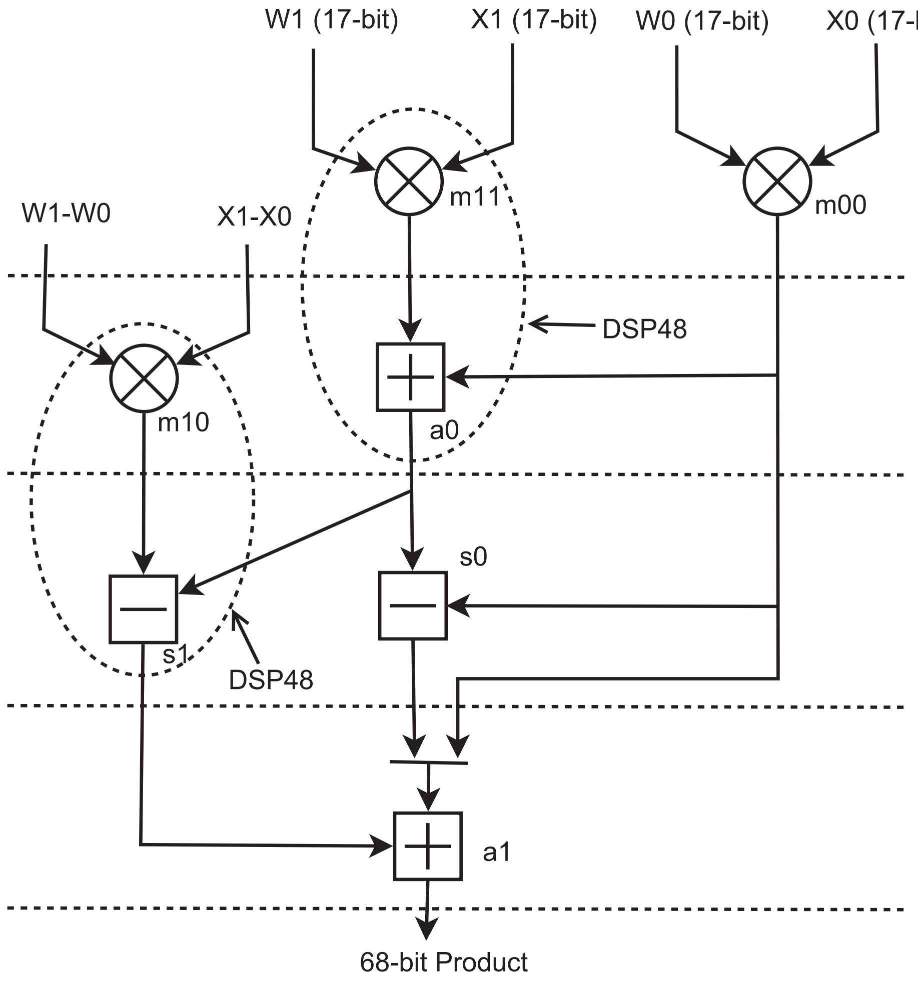


Figure 3.6: Architecture of one 34-bit multiplier (from M. K. Jaiswa and R. C. C. Cheung, "Area-efficient architecture for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support," Microelectronics Journal, vol.44, no.5, pp. 421-430, 2013)

Considering both Figure 3.5 and Figure 3.6, the SP operation needs 7 clock cycles to accomplish and the DP operation needs 12 clock cycles to accomplish. The implementation results show that the clock frequency of this multiplier can run up to 336 MHz, and 9 DSP blocks, 1168 LUTs, and 1373 FFs are required.

This design supports both SP and DP multiplication operations. The number of DSP blocks for the DP multiplication are reduced due to the use of Karatsuba algorithm. However, the mantissa multiplier in this design is a 66-bit multiplier that is larger than the requirement of DP mantissa multiplication. On the other hand, this design uses a fully parallel architecture for both SP and DP operations. If the iterative method is applied to the DP operation, the area of the multiplier can be further reduced.

3.4 Summary

In this chapter, the two previous works and one DP multiplication solution provided by Xilinx Corporation are briefly presented. The proposed multiplier in this thesis is to be compared with these three works in order to show the advantage of the proposed multiplier architecture.

Chapter 4
The Proposed Floating-Point Multiplier

This chapter presents the proposed FP multiplier architecture and its implementation on Xilinx and Altera FPGA devices. In Section 4.1, the architecture of the proposed multiplier and its implementation on Xilinx Virtex-5 device are presented. The mapping solutions of the proposed multiplier architecture on the latest Xilinx Virtex-7 and Altera Arria-10 devices are presented in Section 4.2. Finally, the whole contents of this chapter are summarized in Section 4.3.

4.1 The Proposed FP Multiplier Architecture

The data-path of the proposed FP multiplier for Xilinx FPGA is shown in Figure 4.1. The bold lines represent the pipeline registers and the dash lines separate pipeline stages.

The data-path consists of 6 pipeline stages for SP multiplication and 8 pipeline stages for DP multiplication. For SP operation, the *multiplicand processing*, *multiplier processing*, *exponent processing*, *sign processing*, and the two 27×27 multipliers occupy the first five pipeline stages. The sixth pipeline stage consists of *sp round and normalize* module and *sp post-processing* module. For DP operation, the first five pipeline stages are the same as those of SP multiplication operation. The sixth pipeline stage is composed of *dp carry save tree* module. Then the *dp compound adder* occupies the seventh pipeline stages. The eighth pipeline stage contains *dp rounding and normalize* module and *dp post-processing* module.

In the proposed architecture, the 64-bit input operands contain one-set of DP number (DP mode) or two-set of SP numbers (SP mode). In SP mode, the first inputs of the two SP multiplication operations are included in *multiplicand* and the second inputs are stored in *multiplier*. All major components of the proposed architecture are discussed below in detail.

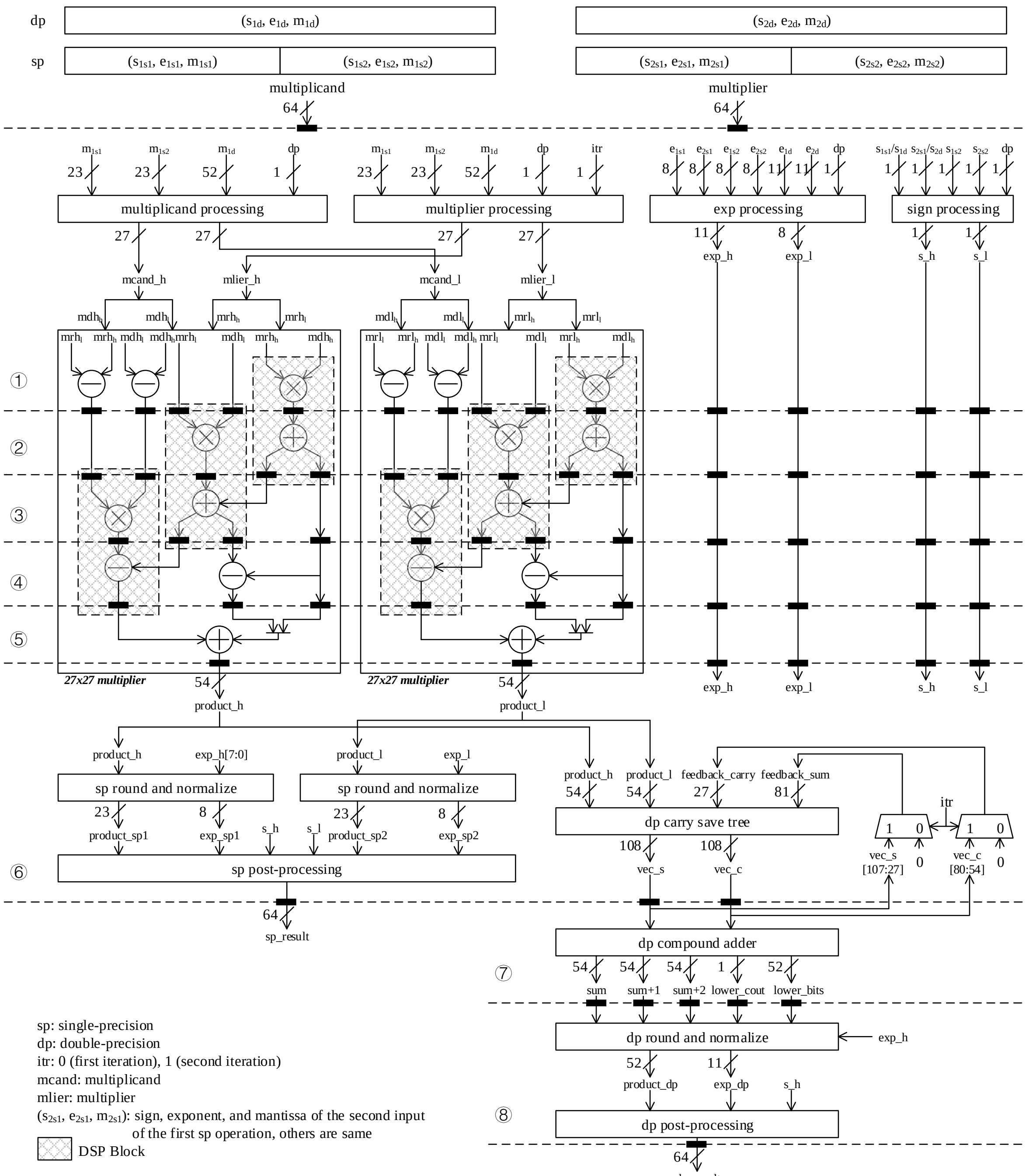
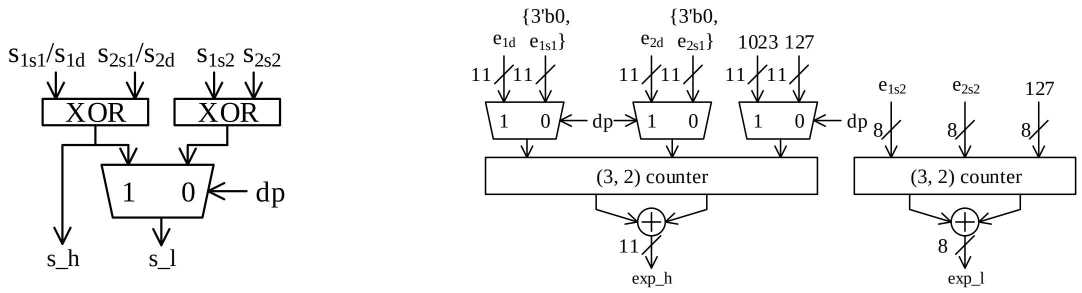


Figure 4.1: Data-Path of the proposed single-precision and double-precision merged floating-point multiplier for Xilinx Virtex-5 device



(a) Sign processing **(b)** Exponent processing

Figure 4.2: Sign and exponent processing in the proposed multiplier

4.1.1 Sign and Exponent Processing

As discussed in Section 2.3, the sign of the product is the XOR of the signs of the two operands and the exponent of the product is the sum of the exponents of the two operands with a bias adjustment.

In the proposed architecture, the sign and exponent of the product are obtained with the circuits shown in Figure 4.2. For sign processing, as shown in Figure 4.2a, two XOR circuits are built. The signs of the operands of DP operation and first SP operation are in the same bit position. Therefore, they share the first XOR gate. The second XOR gate is dedicated for the second SP operation. The *s_h* contains the sign of the product of DP operation (DP mode) or the sign of the product of the first SP operation (SP mode). In SP mode, the *s_l* contains the sign of the product of the second SP operation. In DP mode, the *s_l* is set to the same as *s_h*.

For exponent processing, as shown in Figure 4.2b, two addition circuits are built: the 11-bit addition logic shared by the DP operation and the first SP operation; the 8-bit addition logic dedicated to the second SP operation. As shown in equation (2.2), the exponent processing is a 3-input addition operation. In order to improve the performance, one level of (3,2) counters are used to reduce the operands from three to two. Then the two operands are added using carry lookahead adder. The (3,2) counter has the same architecture as a full adder. The difference is that instead of propagating the output carry of the previous adder to the next adder, all the output carry are reserved and form a new data vector. The result

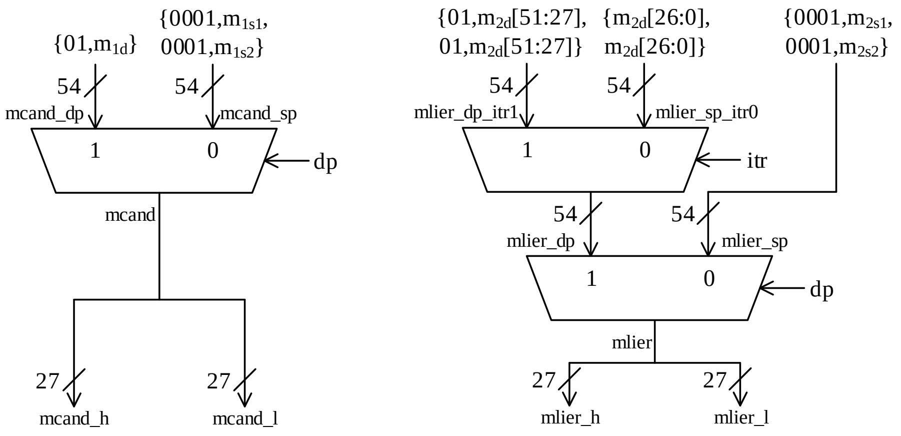


Figure 4.3: Multiplicand and multiplier processing in the proposed multiplier

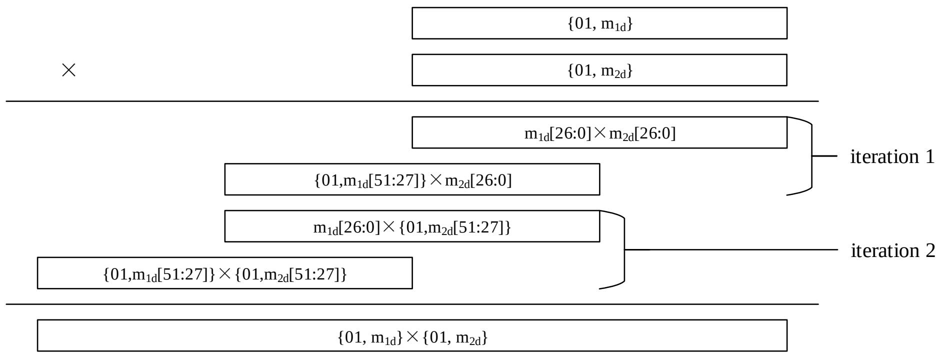

{01, m1d}
×
{01, m2d}
m1d[26:0]×m2d[26:0]
{01,m1d[51:27]}×m2d[26:0]
iteration 1
m1d[26:0]×{01,m2d[51:27]}
{01,m1d[51:27]}×{01,m2d[51:27]}
iteration 2
{01, m1d}×{01, m2d}

Figure 4.4: Mantissa multiplication in DP mode

of (3,2) counter is the redundant carry-save format sum vector and carry vector.

4.1.2 Multiplicand and Multiplier Processing

In parallel with the sign and exponent processing, the mantissas of multiplicand and multiplier are processed to generate the operands of the 27×27 mantissa multipliers.

The diagram of *multiplicand processing* and *multiplier processing* module is shown in Figure 4.3. In SP mode, the 23-bit mantissa m_{1s1} and m_{1s2} of the multiplicand and m_{2s1} and m_{2s2} of multiplier are prefixed with the leading bit and extended with 3-bit zeros to form the 27-bit *mcand_h*, *mcand_l*, *mlier_h*, and *mlier_l*, respectively.

In DP mode, the mantissa multiplication operation is shown in Figure 4.4. For the multiplicand, the 52-bit mantissa m_{1d} is prefixed with the leading bit and extended with 1-bit zero to form the 54-bit *mcand_dp*. Then *mcand_dp* is divided into two parts with equal length 27-bit and assign to *mcand_h* and *mcand_l*, respectively. For the multiplier, the 52-bit mantissa m_{2d} is also extended with $2'b01$. Then the 54-bit format is divided into the 27-bit $\{2'b01, m_{2d}[51:27]\}$ and 27-bit $m_{2d}[26:0]$. In the first iteration ($itr = 0$), both *mlier_h* and *mlier_l* are set to $m_{2d}[26:0]$ and in the second iteration ($itr = 1$), both *mlier_h* and *mlier_l* are set to $\{2'b01, m_{2d}[51:27]\}$.

4.1.3 Mantissa Multiplier

The two 27×27 mantissa multipliers are designed and implemented using DSP blocks embedded in Xilinx Virtex-5 devices in order to efficiently implement the proposed multiplier and to obtain the best performance.

DSP48E blocks in Virtex-5 device support 25×18 two's complement multiplication. The required 27×27 multiplication cannot be achieved by using only one DSP blocks. In order to accomplish this multiplication, the Karatsuba algorithm [26] is applied. For the left 27×27 multiplier, *mcand_h* is divided into higher 14-bit mdh_h and lower 13-bit mdh_l. Similarly, *mlier_h* is divided into 14-bit mrh_h and 13-bit mrh_l. According to the Karatsuba algorithm [26], $mcand\_h \times mlier\_h$ can be calculated as equation (4.1):

$$\begin{aligned} mcand\_h \times mlier\_h &= (mdh_h 2^{13} + mdh_l)(mrh_h 2^{13} + mrh_l) \\ &= M_1 2^{26} + (M_1 + M_3 - M_2) 2^{13} + M_3 \end{aligned} \tag{4.1}$$

where M_1, M_2, and M_3 are shown in equation (4.2):

$$\begin{aligned} M_1 &= mdh_h \times mrh_h \\ M_2 &= (mdh_h - mdh_l)(mrh_h - mrh_l) \\ M_3 &= mdh_l \times mrh_l \end{aligned} \tag{4.2}$$

Each of M_1, M_2, and M_3 is generated by one DSP block. Therefore, a total of three DSP blocks are needed for one 27×27 multiplication.

The mapping diagram of the left 27×27 multiplier on Xilinx Virtex-5 device is shown in Figure 4.5. The bold vertical lines represent pipeline registers. In Figure 4.5, A, B, P, PCIN,

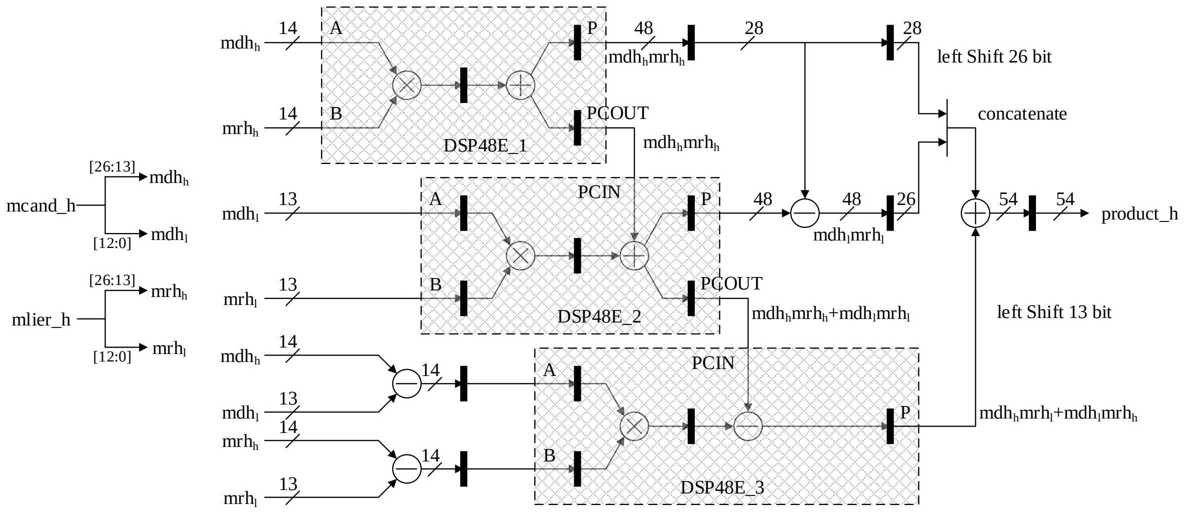


Figure 4.5: Mapping diagram of the left 27×27 mantissa multiplier on Xilinx Virtex-5 FPGA

and PCOUT are DSP48E ports that discussed in Section 2.5. For the three DSP48E blocks, DSP48E_1 is configured to A×B mode, DSP48E_2 is configured to PCIN+A×B mode, and DSP48E_3 is configured to PCIN-A×B mode. These three DSP blocks are used to generate M_1, M_1+M_3, and $M_1+M_3-M_2$, respectively. With the subtractor built with peripheral LUT logics, M_3 can be generated from $M_1 + M_3$. Finally, according to equation (4.1), the product of *mcand_h* × *mlier_h* can be obtained. The proposed multiplier architecture contains two such 27 × 27 multipliers, therefore, a total of 6 DSP blocks are required for the proposed design.

4.1.4 Single-Precision Rounding and Normalization

For SP operation, after the mantissa multiplication, rounding and normalization should be performed to the two products generated by the two 27 × 27 mantissa multipliers.

For SP multiplication, the mantissa plus the leading bit has a length of 24-bit. Therefore, the actual product of the mantissa multiplication cannot be larger than 48-bit. As a result, the most significant 6-bit of the two products generated by the 27 × 27 multipliers can be discarded. The remaining 48-bit has the format as shown in Figure 4.6. In order to make the product fit in the IEEE FP format, the rounding operation is required.

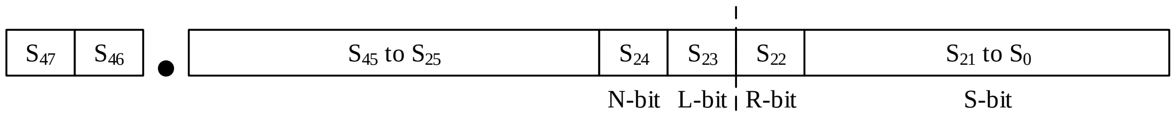


Figure 4.6: Result format of single-precision mantissa multiplication

In Figure 4.6, S_{47}, S_{46}, S_{24} (N-bit), S_{23} (L-bit), S_{22} (R-bit), and $S_{21:0}$ are the bits needed to perform rounding operation. S-bit is sticky bit that can be obtained by ORing $S_{21:0}$. S_{24} and S_{23} are the possible LSB of the rounded product, depending on S_{47}. If $S_{47} = 0$, then S_{23} is the LSB, otherwise, if $S_{47} = 1$, thenS_{24} is the LSB. In the proposed design, roundTiesToEven is performed. The rounding rule can be found in Table 4.1.

When $Round = 1$, the rounding decision needs to add 1 to the product. Here, the 1 is always added to L-bit. It is reasonable when $S_{47} = 0$, because in that case, L-bit is the LSB of the rounded product. However, it is still reasonable when $S_{47} = 1$, although the LSB of the rounded result is N-bit. When rounding is required in this case, L-bit is always 1. Therefore, add 1 to the L-bit and a carry will be generated and propagated to N-bit. It is equivalent to add 1 to N-bit. In hardware, always add 1 to the L-bit can significantly simplify the circuit.

The rounded product can then be normalized according to the value of S_{47}. When $S_{47} = 1$, a 1-bit right shift is needed and the exponent should be added 1. When $S_{47} = 0$, usually no further operation is needed. However, there is one case need to be careful. When $S_{47} = 0$ and $Round = 1$, after plus 1 to the L-bit, $(S + 1)_{47}$ may equal to 1. In this case, normalization is still needed. Therefore, when $S_{47} = 0$ and $Round = 1$, $(S + 1)_{47}$ need to be detected to determine if normalization is required.

Each of the normalized mantissa product is then packed with its corresponding exponent and sign to form the IEEE format SP number. In SP mode, the 64-bit result contains two SP numbers. The SP operation needs 6 clock cycles to accomplish.

4.1.5 Double-Precision Carry Save Tree (CST)

For DP mode, in each iteration, two partial products are generated after the two 27×27 mantissa multipliers. These four partial products need to be accumulated by the carry save

Table 4.1: Rounding rule for SP operation

N	L	R	S	Round	
				$S_{47} = 0$	$S_{47} = 1$
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

tree (CST) to two data vectors before being sent to the DP adder.

The iterative multiplication scheme is realized by this CST module. As shown in Figure 4.1, in the first iteration ($itr = 0$), $feedback\_sum$ and $feedback\_carry$ are set to all zeros. The two products $product\_h$ and $product\_l$ and the two feedback vectors are accumulated by the DP CST. In the second iteration ($itr = 1$), the two vectors vec_s and vec_c generated in the first iteration are feedbacked. These two vectors together with the two products generated in the second iteration are combined by the DP CST and generate the two inputs to the DP compound adder.

In DP mode, the vec_s and vec_c generated by the CST is 108-bit in length. Feedback

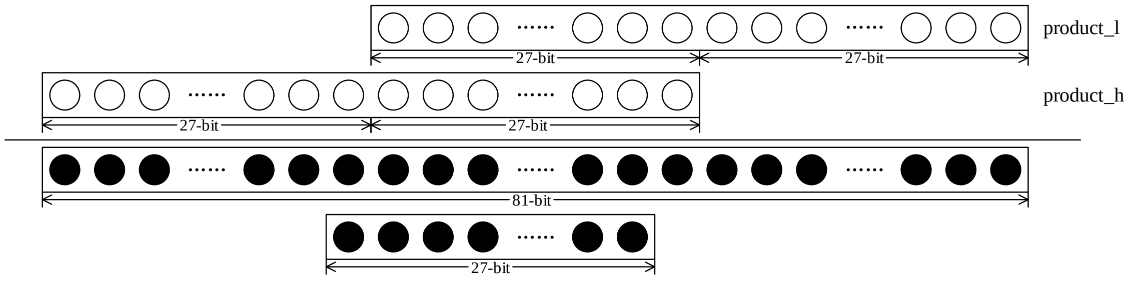


Figure 4.7: Bitwise analysis of partial products and feedback vectors

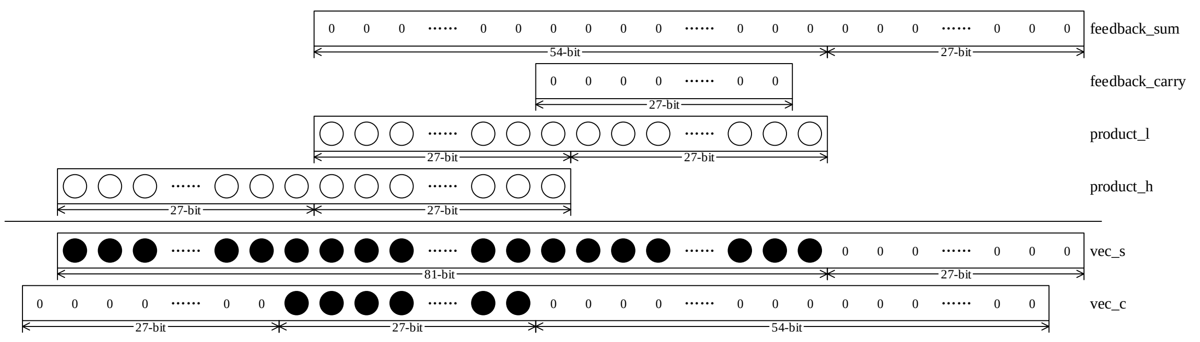
these two large vectors needs a large amount of multiplexers and a quite complicated routing. In order to efficiently realize the feedback operation, a bitwise analysis of the partial products and the feedback vectors is performed.

In the first iteration, the two feedback vectors are all zeros. Therefore, the two vectors need to be feedbacked are determined by the accumulation of the two partial products. As shown in Figure 4.7, the black dots are the bits generated by accumulating the two partial products and they need to be feedbacked to participate the second iteration operation. All other bit positions are zero and do not need to be sent back. Therefore, the required length for $feedback\_sum$ and $feedback\_carry$ are only 81-bit and 27-bit, respectively, which significantly reduces the area consumption and routing complexity.

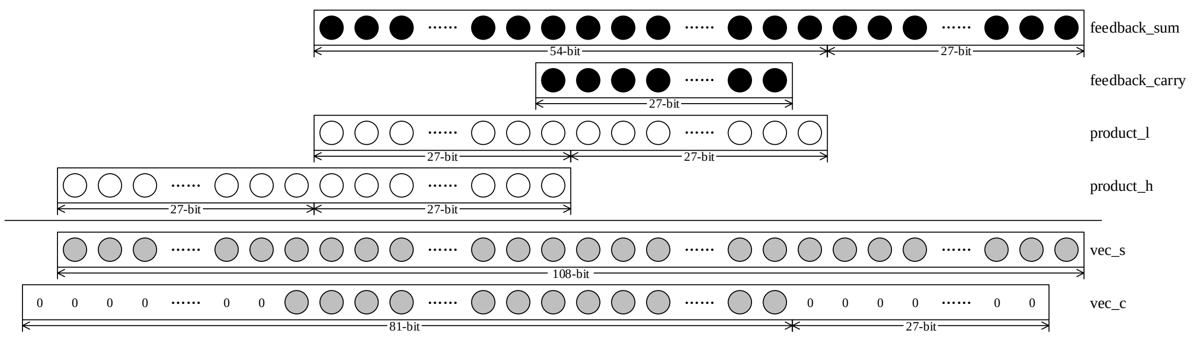
The operations of the optimized DP CST is shown in Figure 4.8. In the first iteration, the two feedback vectors are set to all zeros. Two 108-bit vectors *vec_s* and *vec_c* are generated by accumulating these two feedback vectors and the two partial products *product_h* and *product_l*. *vec_s*[107 : 27] and *vec_c*[80 : 54] are feedbacked and assigned to $feedback\_sum$ and $feedback\_carry$ in the second iteration, respectively. In the second iteration, $feedback\_sum$ and $feedback\_carry$ and the two newly generated partial products are combined to produce two new *vec_s* and *vec_c*. Then the *vec_s* and *vec_c* are sent to *dp compound adder* module to perform the final addition.

4.1.6 Double-Precision Compound Adder

DP addition and rounding are time-consuming due to the large operand length. In the proposed multiplier, in order to improve the performance, the compound adder that combines



(a) First iteration ($itr = 0$)



(b) Second iteration ($itr = 1$)

Figure 4.8: Double-precision carry save tree operations in two iterations

addition and rounding operations is adopted. The compound adder can calculate sum, $sum + 1$, and $sum + 2$ in parallel. Therefore, the following rounding operation is simplified to select one out of these three sums instead of the time-consuming plus one operation.

The DP mantissa multiplication is actually a 53×53 multiplication. Therefore, the product cannot be exceeded 106-bit. As a result, the most significant two bits of vec_s and vec_c can be discarded. Then each of the remaining 106-bit is divided into two parts: the higher 54-bit and the lower 52-bit. The higher 54-bit are used to generate the mantissa of the product and the lower 52-bit is only used for rounding. The addition of these two parts is implemented separately. The lower order adder will generate an output carry $lower\_cout$. It is only used for rounding and will not be propagated to the higher order adder.

In the proposed multiplier design, all the sum, $sum+1$, and $sum+2$ of the higher 54-bit are required. In the rounding stage, when the rounding is not required and $lower\_cout = 0$, the sum is selected. When rounding needs to plus one and $lower\_cout = 0$, or rounding is

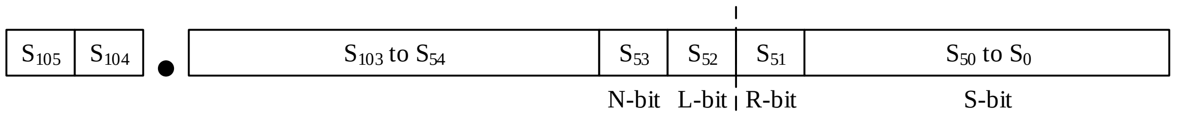


Figure 4.9: Result format of double-precision mantissa multiplication

not required and *lower_cout* $= 1$, $sum + 1$ is selected. When rounding needs to plus one and *lower_cout* $= 1$, $sum + 2$ is selected. These three sums can be generated in parallel.

For the lower order addition, as shown in Figure 4.8, the least significant 27-bit of *vec_c* are all zeros. Therefore, for the least significant 28-bit of *vec_s*, no addition operation is required. As a result, the adder size can be reduced to only 24-bit.

The generated *sum*, $sum + 1$, $sum + 2$, *lower_cout*, and lower order 52-bit *lower_bits* are sent to the *dp round and normalize* module to perform the rounding selection and normalization operation.

4.1.7 Double-Precision Rounding and Normalization

The *sum* concatenated with the *lower_bits* has the format as shown in Figure 4.9. S_{53} (N-bit) and S_{52} (L-bit) are the possible LSB of the rounded result depending on the the value of S_{105}. If $S_{105} = 0$, then S_{52} is the LSB. Otherwise, S_{53} is the LSB. S_{51} (R-bit) is the MSB of *lower_bits*. The remaining bits of *lower_bits*, $S_{50:0}$, are ORed together to generate the sticky-bit (S-bit).

The rounding selection rule is shown in Table 4.2. When *lower_cout* $= 0$, *sum* or $sum + 1$ is selected as the rounded result. The $sum + 1$ is generated by adding 1 to the L-bit of the *sum*. When $S_{105} = 0$, L-bit is the LSB and $sum + 1$ can be generated by adding 1 to L-bit. When $S_{105} = 1$, N-bit is the LSB. However, $sum + 1$ can still be generated by adding 1 to L-bit. Because, as shown in Table 4.2, when $sum + 1$ is selected for $S_{105} = 1$, L-bit is always 1. Therefore, by adding 1 to L-bit, a carry will be generated and propagate the N-bit.

When *lower_cout* $= 1$, $sum + 1$ or $sum + 2$ is selected as the rounded result. In order to generate the rounding rule, the *lower_cout* can be first added to L-bit. Then the rounding rule can be generated according to the new $NLRS$-bit. At this time, when rounding is

Table 4.2: Rounding selection rule for DP operation

N	L	R	S	Round			
				$lower\_cout = 0$		$lower\_cout = 1$	
				$S_{105} = 0$	$S_{105} = 1$	$S_{105} = 0$	$S_{105} = 1$
0	0	0	0	sum	sum	$sum + 1$	$sum + 1$
0	0	0	1	sum	sum	$sum + 1$	$sum + 2$
0	0	1	0	sum	sum	$sum + 2$	$sum + 2$
0	0	1	1	$sum + 1$	sum	$sum + 2$	$sum + 2$
0	1	0	0	sum	sum	$sum + 1$	$sum + 1$
0	1	0	1	sum	$sum + 1$	$sum + 1$	$sum + 1$
0	1	1	0	$sum + 1$	$sum + 1$	$sum + 1$	$sum + 1$
0	1	1	1	$sum + 1$	$sum + 1$	$sum + 2$	$sum + 1$
1	0	0	0	sum	sum	$sum + 1$	$sum + 2$
1	0	0	1	sum	sum	$sum + 1$	$sum + 2$
1	0	1	0	sum	sum	$sum + 2$	$sum + 2$
1	0	1	1	$sum + 1$	sum	$sum + 2$	$sum + 2$
1	1	0	0	sum	$sum + 1$	$sum + 1$	$sum + 1$
1	1	0	1	sum	$sum + 1$	$sum + 1$	$sum + 1$
1	1	1	0	$sum + 1$	$sum + 1$	$sum + 1$	$sum + 1$
1	1	1	1	$sum + 1$	$sum + 1$	$sum + 2$	$sum + 1$

required, $sum + 2$ is selected, otherwise, $sum + 1$ is selected.

The selected rounded result is then normalized according to S_{105}. If $S_{105} = 1$, 1-bit right shift is needed and the exponent should be adjusted by adding 1. If $S_{105} = 0$, the same as we discussed in SP rounding, the $(S+1)_{105}$ or $(S+2)_{105}$ also needs to be considered to determine if normalization is needed.

The normalized mantissa product is then packed with sign and exponent to form the IEEE format DP number. In DP mode, the 64-bit result contains one DP number. The DP data-path contains 8 pipeline stages. However, in order to realize the iterative multiplication

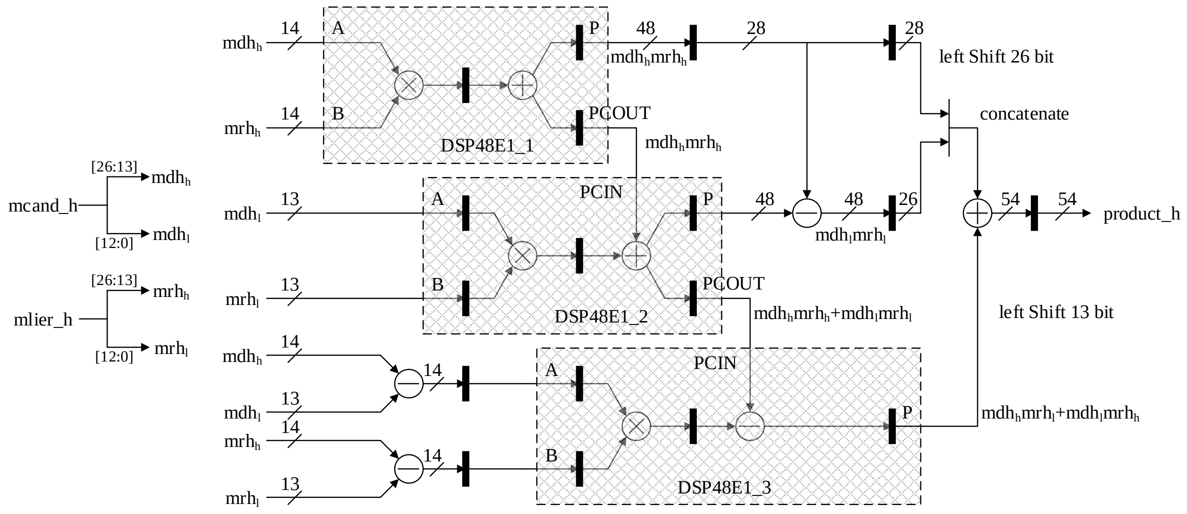


Figure 4.10: Mapping diagram of the left 27 × 27 mantissa multiplier on Xilinx Virtex-7 FPGA

method, the DP CST is used twice. Therefore, the complete DP operation needs 9 clock cycles.

4.2 Mapping Solutions on Xilinx Virtex-7 and Altera Arria-10 Devices

The two 27 × 27 mantissa multipliers in the proposed multiplier architecture are mapped on FPGA using DSP blocks. The DSP blocks among different FPGA devices have different architectures or support different operation modes. Therefore, the mapping solutions of the mantissa multiplier on various FPGA devices may be different. In this section, the mapping solutions of the proposed multiplier on the latest Xilinx Virtex-7 and Altera Arria-10 devices are provided.

4.2.1 Xilinx Virtex-7 Device

In Xilinx Virtex-7 device, the embedded DSP48E1 has similar architecture as DSP48E in Virtex-5 devices. They all support 25 × 18 two's complement multiplication. Therefore, the mapping strategy on Virtex-7 device is the same as that on Virtex-5 device.

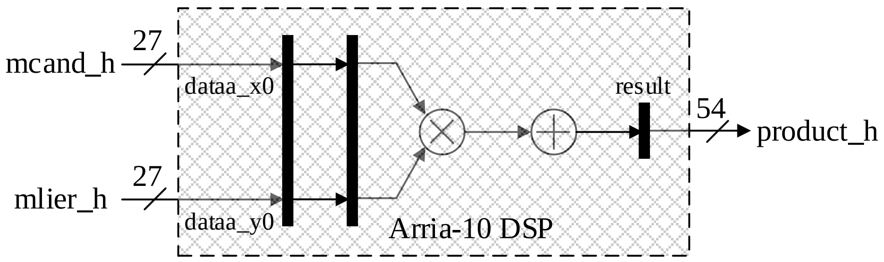


Figure 4.11: Mapping diagram of the left 27 × 27 mantissa multiplier on Altera Arria-10 FPGA

The mapping diagram of the left 27 × 27 multiplier on Xilinx Virtex-7 device is shown in Figure 4.10. The bold vertical lines represent pipeline registers. The mapping diagram here is the same as that of Virtex-5 device. One 27 × 27 mantissa multiplier needs three DSP blocks and a total of 6 DSP48E1 blocks are occupied by the whole multiplier design.

The design pipeline stages on Virtex-7 device is also the same as that on Virtex-5 device. Two parallel SP operations can be accomplished in one iteration with a latency of 6 clock cycles and one DP operation can be accomplished in two iterations with a latency of 9 clock cycles.

4.2.2 Altera Arria-10 Device

In Arria-10 devices, although the hardened floating-point DSP blocks are provided, only the SP operations are supported. In the proposed multiplier, in order to support both SP and DP in a unified architecture, the proposed architecture is mapped on Arria-10 FPGA using the native fixed-point DSP blocks. The fixed-point DSP blocks in Arria-10 support 27 × 27 unsigned multiplication. Therefore, only one DSP block is needed to implement one 27 × 27 mantissa multiplier in the proposed architecture.

The mapping diagram of the left 27 × 27 multiplier on Arria-10 device is shown in Figure 4.11. The bold vertical lines represent the registers inside the Arria-10 DSP blocks. The first and last stages registers are data input and data output registers. The middle one is one stage of pipeline registers. According to [38], in order to obtain the best performance, all three stages registers are required by the DSP blocks. Therefore, each 27 × 27 multiplier needs 2 clock cycles to finish its operation.

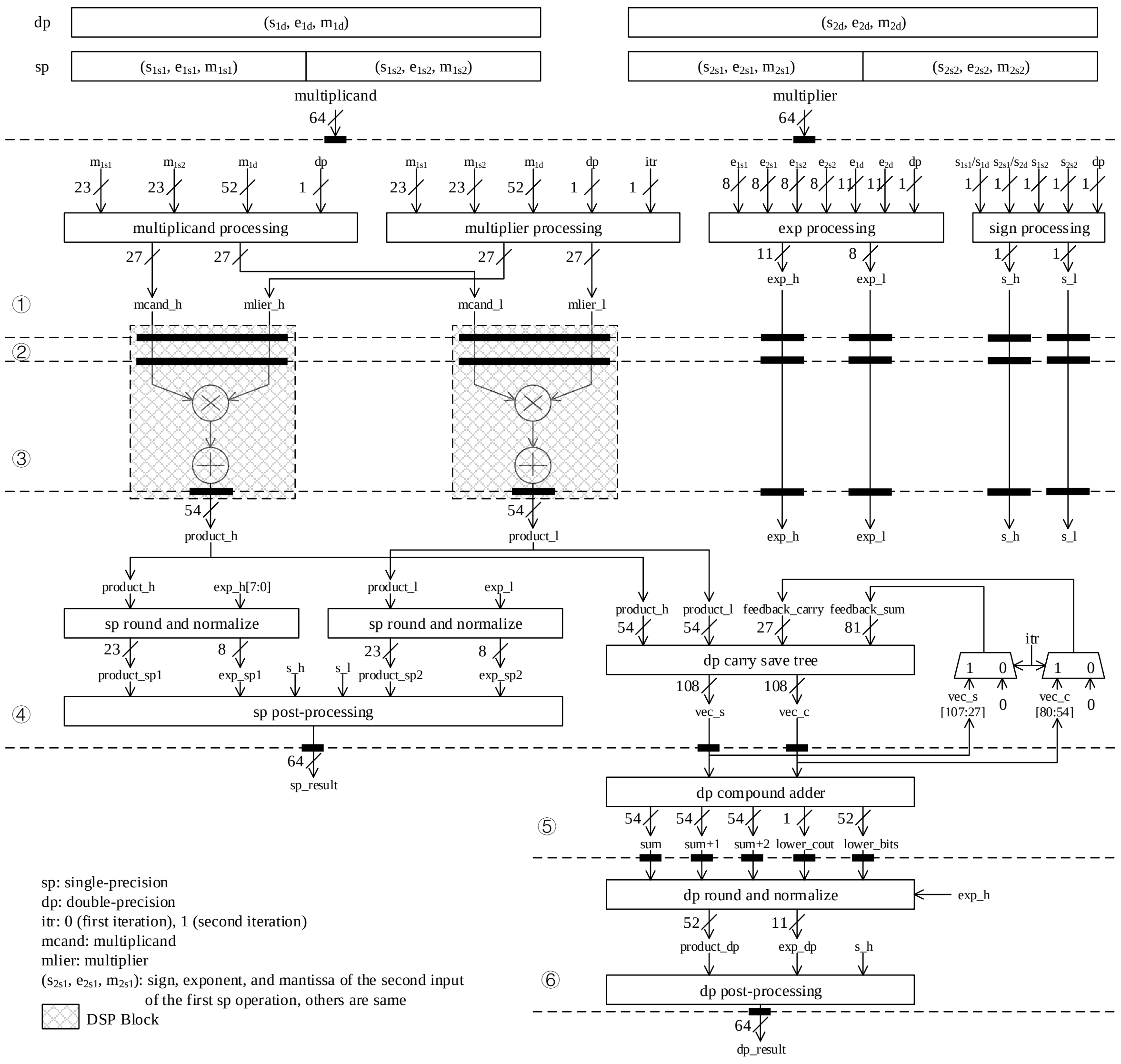


Figure 4.12: Data-Path of the proposed single-precision and double-precision merged floating-point multiplier on Altera Arria-10 device

Because of the pipeline stages of the mantissa multiplier are changed, the latency of SP and DP operations are also changed. The full data-path of the proposed multiplier on Altera Arria-10 device is shown in Figure 4.12. The bold lines represent the pipeline registers and the long dash lines separate pipeline stages.

As shown in Figure 4.12, on Altera Arria-10 device, two parallel SP multiplications can be accomplished in 4 clock cycles and one DP operation can be finished in two iterations with a latency of 7 clock cycles.

4.3 Summary

In this chapter, the architecture of the proposed FP multiplier is presented. The design of each component is discussed in detail. The iterative multiplication method is used in order to reduce the area of the proposed design. On Xilinx FPGA device, the Karatsuba algorithm is also applied in order to further reduce the DSP block usage.

In order to efficiently implement the proposed multiplier on FPGA device, the DSP blocks are used to map the mantissa multipliers. The DSP blocks in different FPGA devices may have different architectures or support different operation modes. Therefore, different mapping strategies may be applied to different FPGA devices. In this chapter, in addition to Virtex-5 device, the mapping solutions for Virtex-7 and Arria-10 devices are also provided.

On both Virtex-5 and Virtex-7 devices, two parallel SP operations can be accomplished in one iteration with a latency of 6 clock cycles and one DP operation can be accomplished in two iterations with a latency of 9 clock cycles. Whereas, on Arria-10 device, two parallel SP operations can be accomplished in one iteration with a latency of 4 clock cycles and one DP operation can be accomplished in two iterations with a latency of 7 clock cycles.

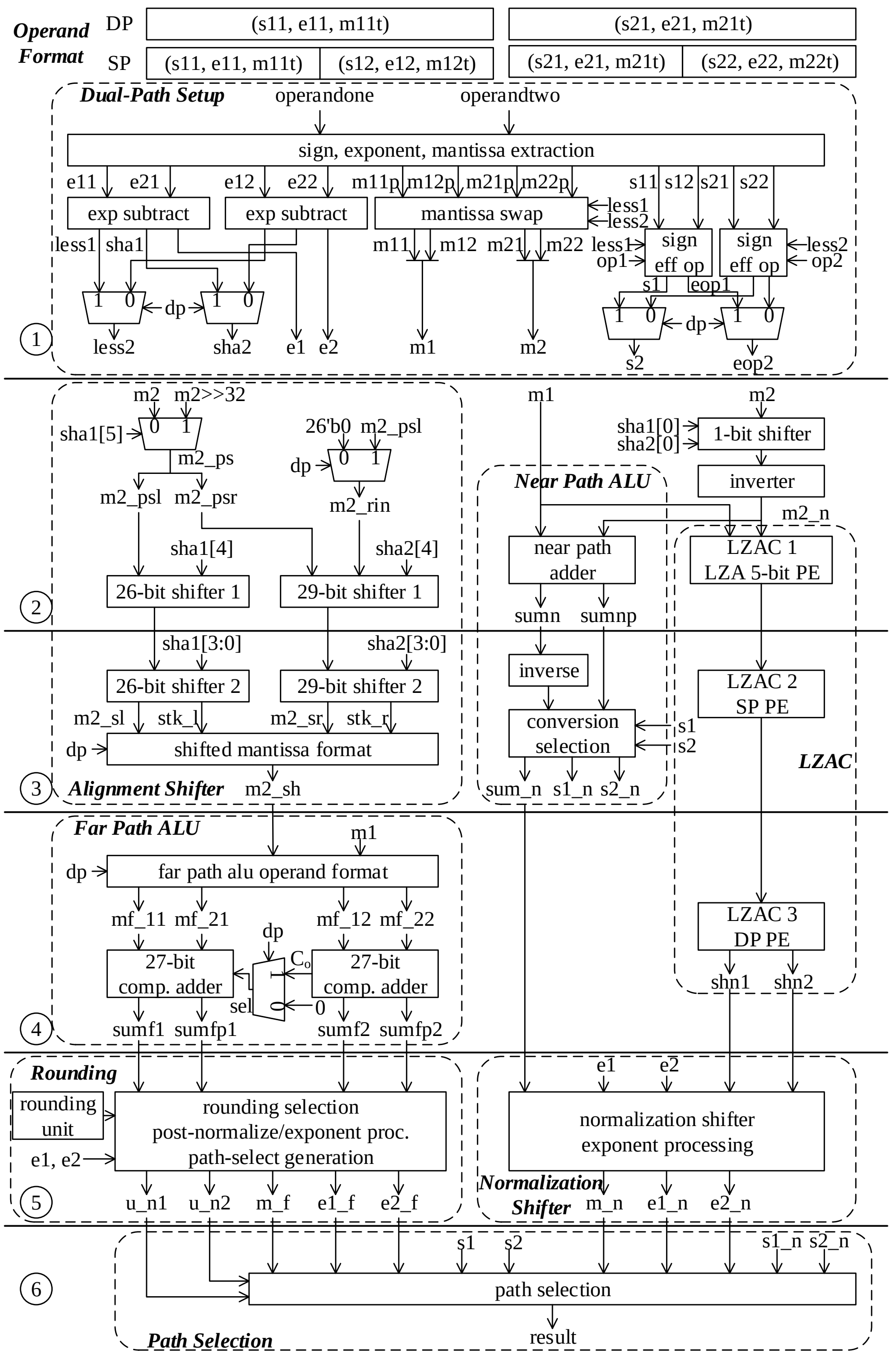
Chapter 5

The Proposed Floating-Point Adder

This chapter presents the proposed FP adder architecture. The design of pipeline stages is presented. The design of each components, Dual-Path Setup (Section 5.1), Alignment Shifter (Section 5.2), FAR Path ALU and Rounding (Section 5.3), NEAR Path ALU (Section 5.4), Leading Zero Anticipation and Counting (Section 5.5), Normalization Shifter (Section 5.6), and Path Selection (Section 5.7), to support both SP and DP operations are presented in detail in each section. Finally, the whole design is summarized in Section 5.8.

The proposed adder is designed based on the two-path FP addition algorithm. The data-path of the proposed adder is shown in Figure 5.1. The whole data-path is divided into 6 pipeline stages. As shown in Figure 5.1, the first pipeline stage contains the *Dual-Path Setup* module. After that the data-path is divided into two paths: NEAR path and FAR path. For FAR path, the *Alignment Shifter* occupies the second and third pipeline stages. Following by *Far Path ALU* which occupies the fourth pipeline stages. The fifth pipeline stage is occupied by *Rounding* module. Meanwhile, for NEAR path, the *LZAC* module occupies three pipeline stages. In parallel, the *Near Path ALU* module takes up two pipeline stages. The *Normalization Shifter* module is in the fifth pipeline stage. After the fifth pipeline stage, the two paths re-convergence by the *Path Selection* module which occupies the sixth pipeline stage.

In the proposed design, the 64-bit input operands contain one-set of DP number (DP mode) or two-set of SP numbers (SP mode). In SP mode, the first inputs of the two SP operations are included in *operandone* and the second inputs are stored in *operandtwo*. All major components of the proposed architecture are discussed below in detail.

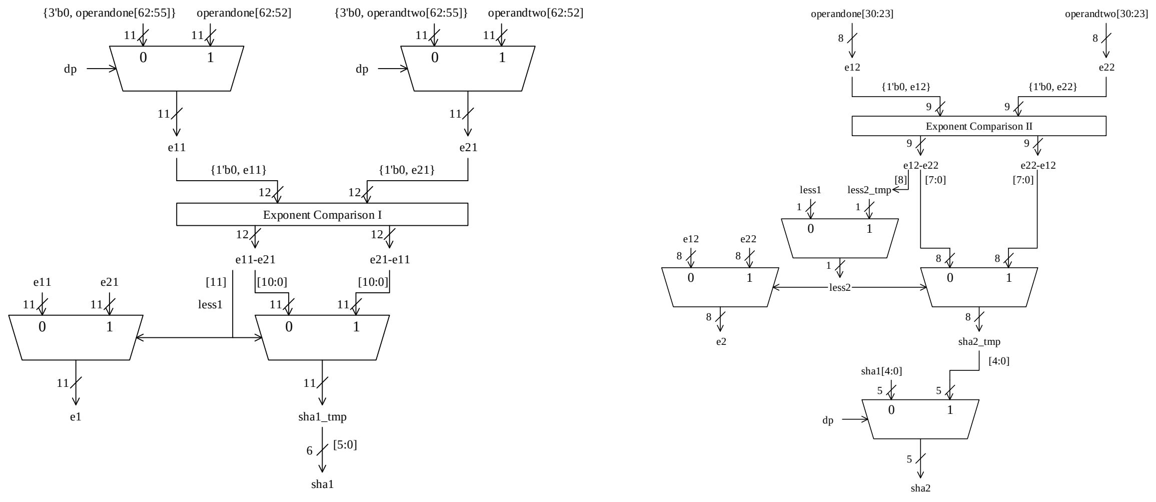


e: exponent m: mantissa s: sign op: operation eop: effective operation
less: first operand is smaller than the second one
sha: alignment shifting shn: normalization shifting
u_n: use_nearpath _n: near path _f: far path sumxp: sum plus one

Figure 5.1: Data-Path of the proposed single-precision and double-precision merged floating-point adder

5.1 Dual-Path Setup

The *Dual-Path Setup* module is to prepare the data for the FAR path and NEAR path for further processing. In this module, the sign, exponent, and mantissa of the two operands are first extracted. The exponents of the two operands are compared and the mantissas are swapped accordingly. Besides, the effective operation and the sign of the result are generated. The effective operation is the operation (addition/subtraction) performed on the two operands. For example, if A is positive, B is negative, and the actual operation is addition, then the effective operation of $A + B$ is subtraction.



(a) Exp subtract I **(b)** Exp subtract II

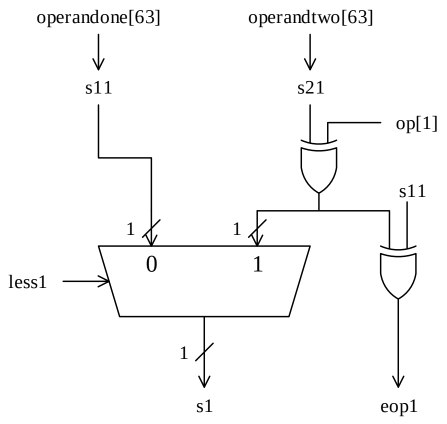
Figure 5.2: Exp subtract blocks in Dual-Path Setup module

m11t	0	0	0	0	0	1	m12t
m21t	0	0	0	0	0	1	m22t

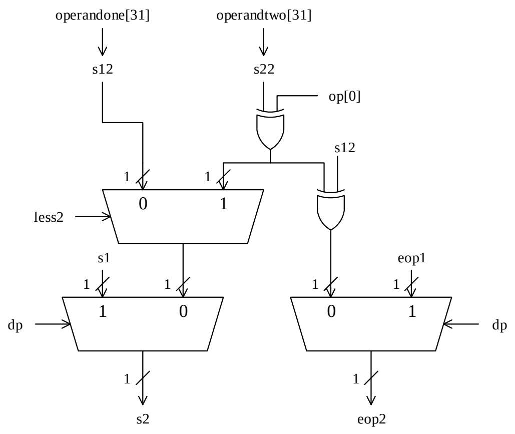
Figure 5.3: Combine two single-precision mantissa

In order to reduce the area, the DP operation and the first SP operation share the same hardware to compare their exponents, as shown in Figure 5.2a. This hardware generates a

6-bit shifting amount $sha1$ for the corresponding mantissa. Similarly, the second exponent comparison circuit, as shown in Figure 5.2b, generates a 5-bit shift amount $sha2$ for the second single-precision operation. In DP mode, $sha2$ is set to $sha1$[4:0]. Moreover, the mantissa of the two SP operations are combined into a 52-bit format which is the same width of the double-precision mantissa, as shown in Figure. 5.3. Then each of the two 52-bit mantissa is divided into two 26-bit $m11p$, $m12p$ ($m21p$, $m22p$). After swapping, $m1$ always contains the mantissa of the larger number.



(a) Sign and eff_op I



(b) Sign and eff_op II

Figure 5.4: Sign and effective operation generation logic in Dual-Path Setup module

The diagram of sign generation and effective operation generation logics is shown in Figure 5.4. For each operation, assume the sign of the two operands are s_1 and s_2, the actual operation is *op* (*op* $= 0$ for addition and *op* $= 1$ for subtraction), the result of the exponents comparison is $less$ ($less = 0$ represents the first exponent is larger than the second one, and $less = 1$ otherwise), the sign of the result s and the effective operation *eop* can be generated according to equation (5.1)(5.2). The DP operation shares these logics with the first SP operation.

$$s = (s_1 \cdot \overline{less}) + (s_2 \oplus op) \cdot less \tag{5.1}$$

$$eop = s_1 \oplus s_2 \oplus op \tag{5.2}$$

5.2 Alignment Shifter

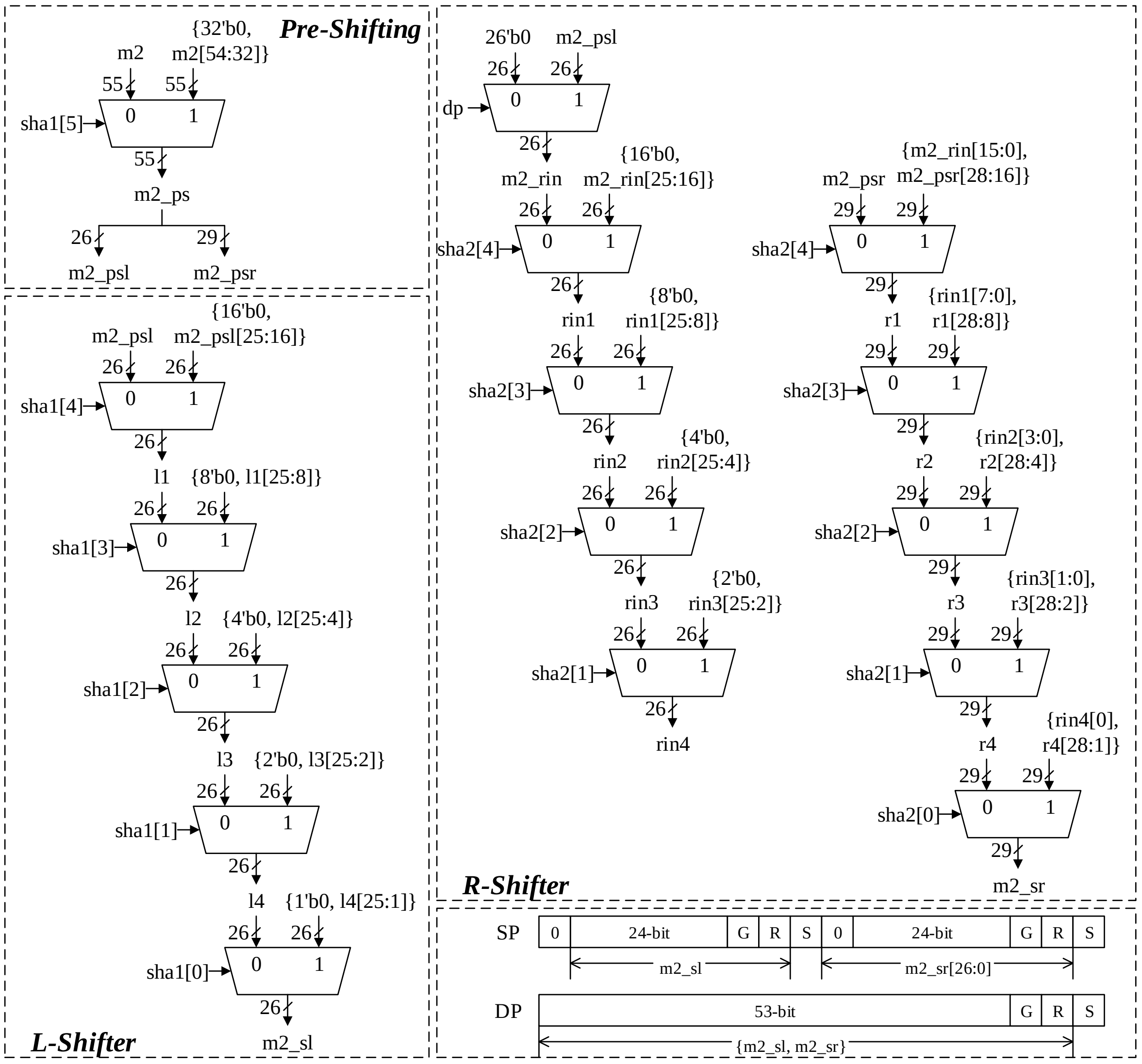


Figure 5.5: Alignment shifter of the proposed FP adder

The *Alignment Shifter* module is used to shift the mantissa of the operand that has the smaller exponent to make the two operands have the equal exponent.

The $m2$ from the *Dual-Path Setup* module is prefixed with the hidden 1 and extended 2-bit zeros to the right of the LSB (in order to reserve the Guard bit (G-bit) and Round bit (R-bit) for the future rounding) to form the 55-bit input signal for the alignment shifter. For SP operation, as shown in Figure. 5.3, the first mantissa is already been extended by zeros, therefore, there is no need to extend more bits for the first SP mantissa.

The diagram of the alignment shifter is shown in Figure 5.5. For DP operation, a 55-bit dynamic shifter is required. The 55-bit shifter is divided into one 26-bit shifter (L-Shifter) and one 29-bit shifter (R-Shifter) to support dual single-precision operations. In the pre-shifting stage, the shifting amount $sha1[5]$ is first considered because the possible shifting amount 32-bit is larger than maximum shifting amount of L-Shifter and R-Shifter. Obviously, only in the DP mode, this 32-bit shifting can occur. The 55-bit pre-shifted signal $m2\_ps$ is then divided into the 26-bit $m2\_psl$ which is the input of the L-Shifter and the 29-bit $m2\_psr$ which is the input of the R-Shifter. The $m2\_psl$ is also sent to the R-Shifter because in DP mode, the data bits shifted out from the L-Shifter should be sent to the R-Shifter.

The L-Shifter and R-Shifter are both dynamic shifter. In each stage, one bit of the shift amount is considered. In order to obtain better performance, pipeline registers are inserted between the first and second stages of the L-Shifter and R-Shifter (register $l1$, $rin1$, and $r1$).

The sticky bit (STK-bit) is calculated in parallel with the shifter by ORing the shifted out bits. A large width OR operation can be efficiently implemented in FPGA by using the dedicated carry logic in the slices. Therefore, the performance of the shifter is not affected. Finally, the output signal is formed as shown in the bottom-right of Figure. 5.5, where S represents the STK-bit.

In this module, the SP and DP operation share the shifting logic and the DP operation shares the STK-bit generation logic with the second SP operation. Extra STK-bit generation is needed for the first SP operation.

5.3 FAR Path ALU and Rounding

In the FAR path arithmetic logic unit (ALU) module, the compound adder that combines the addition and the rounding is implemented. Two 27-bit compound adders that calculate both the sum and $sum + 1$ are used to support two SP operations. In order to support DP operation, the carry out of the lower order adder is used to select sum or $sum + 1$ of the higher order adder. The rounding bits (G-bit, R-bit, and STK-bit) are calculated separately.

In the proposed design, the roundTiesToEven as shown in Section 2.2 is implemented. For

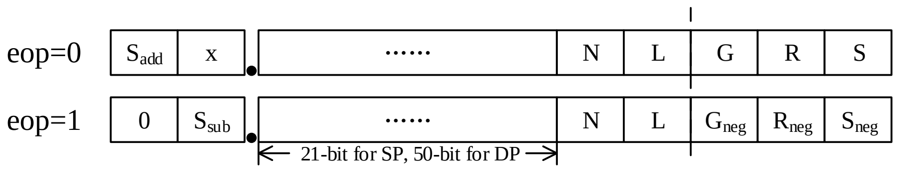


Figure 5.6: Result format of FAR path ALU

the effective addition and effective subtraction, the rounding decision is generated separately as described in [41].

The result *sum* of the FAR path ALU has the format as shown in Figure 5.6. L is the LSB, N is the bit next to the LSB, G is guard bit, R is round bit, and S is sticky bit. G_{neg}, R_{neg}, and S_{neg} are the two's complement of G, R, and S used in subtraction. These bits are used to determine if rounding is needed. The S_{add} and S_{sub} are used to determine if normalization is needed. For the two operands of the FAR path ALU, the larger mantissa is within the range $[1, 2)$ and the smaller mantissa is within the range $(0, 2)$. When the *eop* is addition, the result is in the range $[1, 4)$. Therefore, if $S_{add} = 1$, a 1-bit right shift is needed to normalize the result. On the other hand, when the *eop* is subtraction, the result is located within $(-1, 2)$. The negative result is not considered because in that situation the NEAR path result is selected. For the range of $(0, 2)$, if $S_{sub} = 1$, a 1-bit left shift is needed to normalize the result.

For effective addition, rounding operation can be implemented according to Table 5.1. When generating the $sum + 1$ in the ALU, we always add 1 to L-bit. When $S_{add} = 1$, a 1 needs to be added to N-bit, because after normalize, N-bit is the actual LSB. However, by observing Table 5.1, when rounding is needed for $S_{add} = 1$, L-bit is always 1. If the 1 is added to the L-bit, there is a carry generated to the N-bit. Therefore, it is reasonable to always add 1 to the L-bit if rounding is needed.

For effective subtraction, rounding operation can be implemented according to Table 5.2. Here $sum + 1$ is still generated by adding 1 to L-bit. When $S_{sub} = 1$, G_{neg} will become the actual LSB. Different from the N-bit in addition, we cannot add 1 to G_{neg} through the carry bit. The solution here is to flip the G_{neg}-bit which is equivalent to add 1 to it. When the original $G_{neg} = 0$, add 1 to it will not generate a carry, therefore, there is no need to add 1

Table 5.1: Rounding rule for effective addition

N	L	G	$R+S$	Round	
				$S_{add}=0$	$S_{add}=1$
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	0	1
1	1	1	0	1	1
1	1	1	1	1	1

to L-bit. On the other hand, when the original $G_{neg} = 1$, a 1 needs to be added to L-bit. When $S_{sub} = 0$, if rounding is needed, the 1 is added to L-bit. The flip G_{neg} operation does not affect the result because the G_{neg}-bit is not within the bit position range of the result.

Then according to the rounding decision, the sum and $sum+1$ are selected. The selected result then needs to be normalized by no more than 1-bit right or left shifting. The normalization operation should be determined by the rounded result instead of S_{add} or S_{sub} that is used in rounding determination. Because the rounding is determined by analyzing the bits in sum. It is possible that when eop is addition, the sum is not overflow but the $sum + 1$ is overflow. Similarly, when eop is subtraction, sum is underflow but the $sum + 1$ might be

Table 5.2: Rounding rule for effective subtraction

L	G_{neg}	R_{neg}	S_{neg}	Round		
				$S_{sub}=1$	$S_{sub}=0$	operation
0	0	0	0	0	0	-
0	0	0	1	0	0	-
0	0	1	0	0	0	-
0	0	1	1	0	1	flip G_{neg}
0	1	0	0	0	0	-
0	1	0	1	1	0	add 1 to L
0	1	1	0	1	1	add 1 to L, flip G_{neg}
0	1	1	1	1	1	add 1 to L, flip G_{neg}
1	0	0	0	0	0	-
1	0	0	1	0	0	-
1	0	1	0	0	0	-
1	0	1	1	0	1	flip G_{neg}
1	1	0	0	1	0	add 1 to L
1	1	0	1	1	0	add 1 to L
1	1	1	0	1	1	add 1 to L, flip G_{neg}
1	1	1	1	1	1	add 1 to L, flip G_{neg}

normalized number.

There are two cases that the FAR path result should be discarded:

1. The result is negative. This can only occur when the exponents of the two operands are equal and the effective operation is subtraction. The NEAR path has the circuit to deal with negative result.

2. The result is too small that needs more than 1-bit left shift to normalize. This can occur when the exponents difference is no more than one and the effective operation is subtraction. This can be handled by the NEAR path normalization shifter.

If these two cases are detected, the signal u_n1 or u_n2 is asserted that indicates the NEAR path result should be selected in the *Path Selection* module.

In this module, the SP and DP operation share the mantissa adder. The DP operation shares the rounding logic with the second SP operation and shares the normalization detection and u_n1 detection logic with the first SP operation. Extra rounding logic is needed for the first SP operation and extra normalization and u_n2 detection logic is needed for the second SP operation.

5.4 NEAR Path ALU

The NEAR path is responsible for the cases when the exponent difference is 0 or 1 and the effective operation is subtraction. Therefore, only the subtraction is performed in the NEAR path ALU. The NEAR path ALU contains two 27-bit adders. In SP mode, they run independently. In DP mode, the carry out from the lower adder is used to select the result of the higher adder to form the double-precision result. In order to perform subtraction, the second input operand B is first inverted and then added with the first input operands A with the input carry setting to 1.

Not only $A + \overline{B} + 1$, but also $A + \overline{B}$ are calculated. If $A + \overline{B} + 1$ is positive, then it is selected as the result. Otherwise, it should be complemented and at this time, the $\overline{A + \overline{B}}$ is the correct result according to equation (5.3).

$$\begin{aligned} -(A - B) &= -(A + \overline{B} + 1) \\ &= -(A + \overline{B}) - 1 \\ &= (\overline{A + \overline{B}} + 1) - 1 \\ &= \overline{A + \overline{B}} \end{aligned} \tag{5.3}$$

Note that only when the FAR path result is negative or the FAR path result needs more than 1-bit left shift to normalize, the NEAR path result is selected. In these two cases, no rounding is needed. Therefore, there is no need to perform rounding in NEAR path.

In this module, the SP and DP operation share the mantissa subtracter. The DP operation shares the sign detection logic with the first SP operation. Extra sign detection logic is needed for the second SP operation.

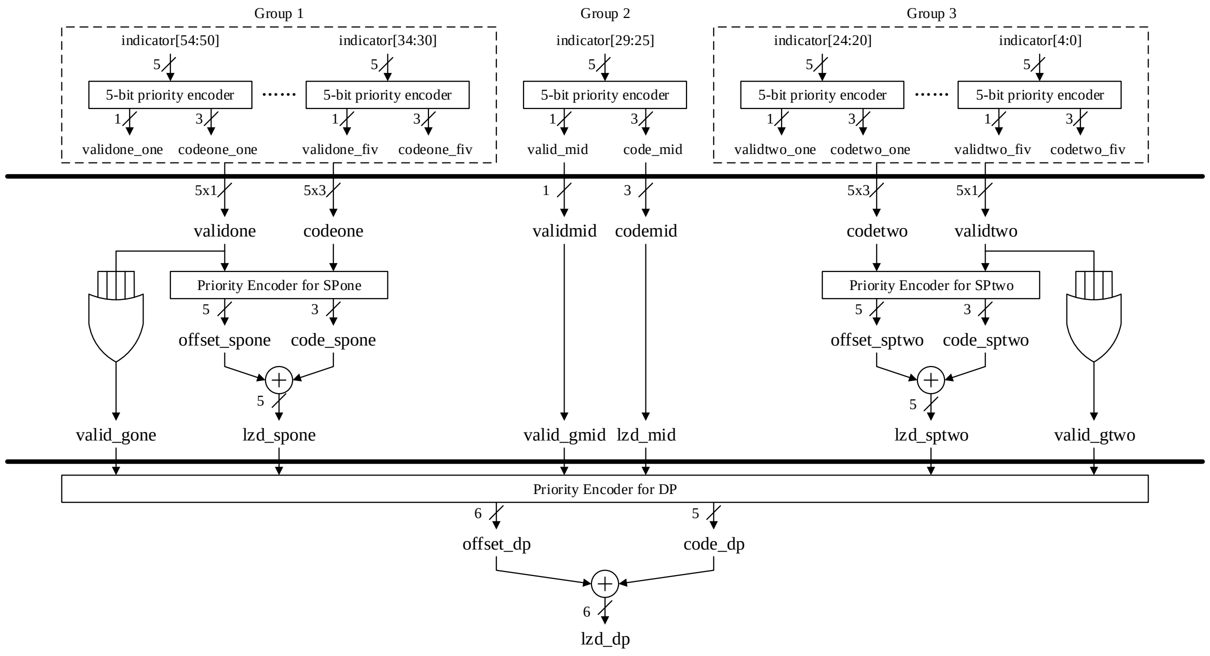


Figure 5.7: Leading zero anticipation and counting (LZAC) unit

5.5 Leading Zero Anticipation and Counting (LZAC)

When performing subtraction, leading zeros may occur when the result is positive and leading ones may occur when the result is negative. In the proposed design, LZAC is used to predict the position of the leading bit directly from the two operands and then calculate the number of zeros or ones to the left of the leading bit. This number is used later by the normalization shifter.

In the proposed design, both positive and negative results may be generated in the NEAR path, according to [42], for k-bit operands A and B, the indicator generated by leading zero anticipation (LZA) unit should be:

$$\begin{aligned} f_{k-1} =& \overline{T}_0 T_1 \\ f_i =& T_{i+1}(G_i\overline{Z}_{i-1} + Z_i\overline{G}_{i-1}) + \\ & \overline{T}_{i+1}(Z_i\overline{Z}_{i-1} + G_i\overline{G}_{i-1}) \quad i < k-1 \end{aligned} \tag{5.4}$$

where $T = A \oplus B$, $G = AB$, and $Z = \overline{A}\ \overline{B}$.

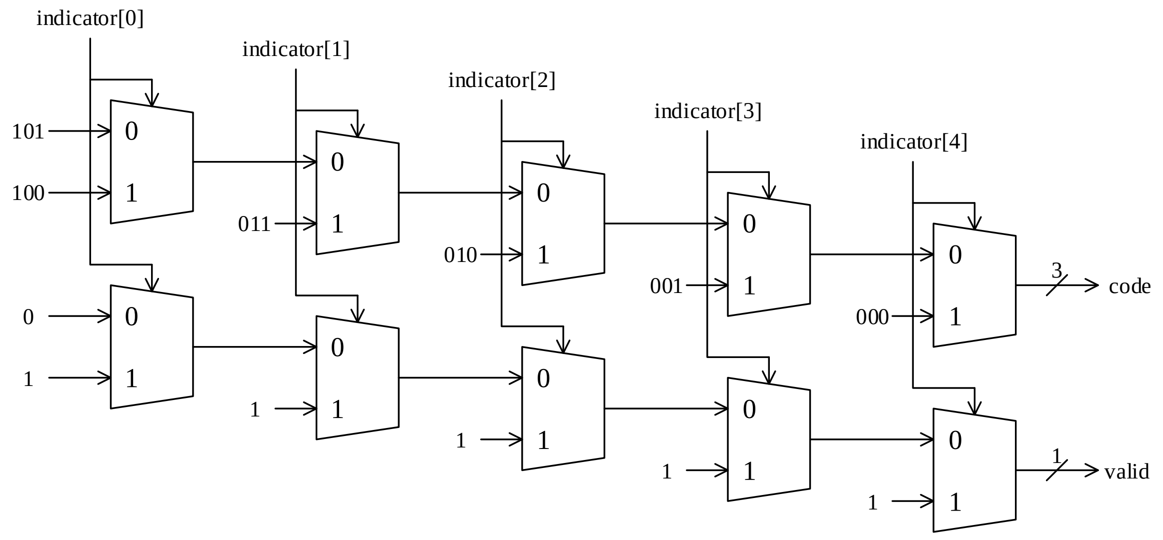


Figure 5.8: 5-bit priority encoder in the leading zero anticipation and counting (LZAC) unit

There are many methods to count the leading digit number [42]. In FPGA, a good way to realize the counting is to use the priority encoder [35]. The leading zero counting unit implemented in the proposed adder is shown in Figure 5.7. In order to fully utilize the LUT in Virtex-5 device [30], the 5-bit priority encoder is adopted. The indicator is divided into 5-bit groups. Each group uses one priority encoder to generate the valid signal and code signal. The diagram of the 5-bit priority encoder is shown in Figure 5.8. The truth table of the 5-bit priority encoder is shown in Table 5.3.

There are totally 11 such priority encoders and divided into three groups as shown in Figure. 5.7. The valid signal and code signal generated by Group 1 and Group 3 are used by another level of priority encoders to generate the number of leading bit for the two SP operations. Group 2 only contains one 5-bit priority encoder and is only used in DP mode. The diagram of the SP priority encoder is shown in Figure 5.9. The truth table of the SP priority encoder is shown in Table 5.4. The number of leading bit for the single-precision operation is obtained by adding the offset and the code together.

In the next stage, the five valid values in one group are ORed together to generate the group valid signal *valid_gone*, *valid_gmid*, and *valid_gtwo*. The group valid signals, the single-precision counting *lzd_spone* and *lzd_sptwo*, and the code signal *lzd_mid* generated

Table 5.3: 5-bit priority encoder truth table

indicator					valid	code
1	x	x	x	x	1	000
0	1	x	x	x	1	001
0	0	1	x	x	1	010
0	0	0	1	x	1	011
0	0	0	0	1	1	100
0	0	0	0	0	0	101

Table 5.4: SP priority encoder truth table

valid_one to valid_fiv					offset	code
1	x	x	x	x	0	code_one
0	1	x	x	x	5	code_two
0	0	1	x	x	10	code_thr
0	0	0	1	x	15	code_for
0	0	0	0	1	20	code_fiv
0	0	0	0	0	25	0

by the Group 2 are finally used to generate the number of leading bit for double-precision operation. The diagram of the DP priority encoder is shown in Figure 5.10. The truth table of DP priority encoder is shown in Table 5.5. By adding the offset and the code, the number of leading bit for the double-precision operation is obtained.

In order to unify the SP and DP operation, two unified shifting amount $shn1$ and $shn2$ are sent out by this module. In SP mode, the 6-bit normalization shifting amount $shn1 = \{1'b0, lzd\_spone\}$ and the 5-bit shifting amount $shn2 = lzd\_sptwo$. In DP mode, $shn1 = lzd\_dp$ and $shn2 = lzd\_dp[4:0]$.

In this module, there is no area overhead compared to the LZAC in DP adder to count the leading bits. An extra level of multiplexers are needed to generate the unified shifting

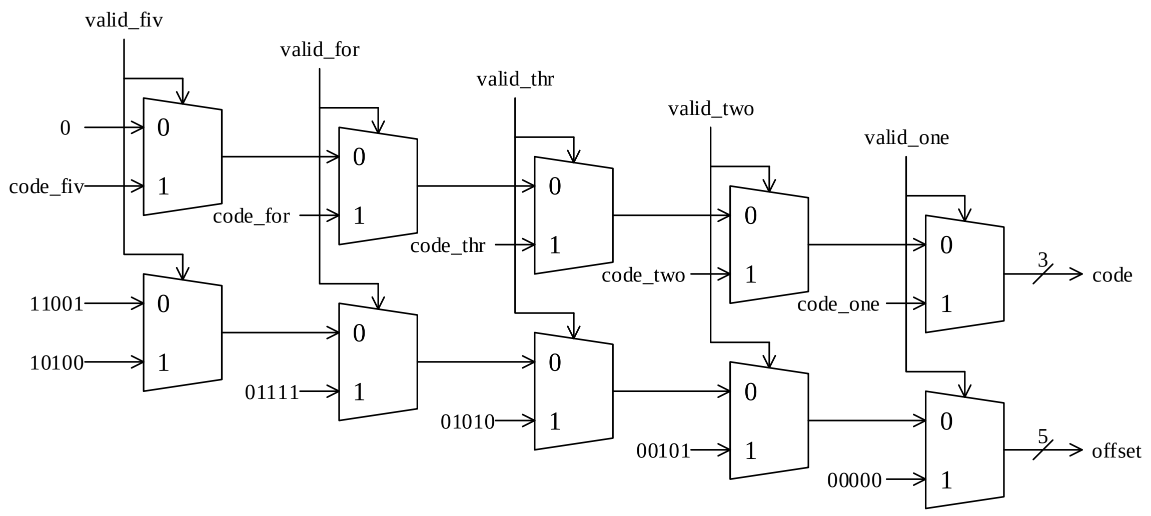


Figure 5.9: SP priority encoder in the leading zero anticipation and counting (LZAC) unit

Table 5.5: DP priority encoder truth table

valid_gone	valid_gmid	valid_gtwo	offset	code
1	x	x	0	lzd_spone
0	1	x	25	code_mid
0	0	1	30	lzd_sptwo
0	0	0	54	0

amount and these multiplexers also bring extra delay.

5.6 Normalization Shifter

The normalization shifter receives the mantissa summation from the NEAR path ALU and the two shift amounts from LZAC and left shifts the mantissa in order to normalize it. The normalization shifter contains two 27-bit shifters. The architecture of the normalization shifter as shown in Figure 5.11 is similar to the alignment shifter as shown in Figure 5.5. The difference is that the shifting direction is left.

The LZAC used in the previous step may have 1-bit error. In order to correct the 1-bit

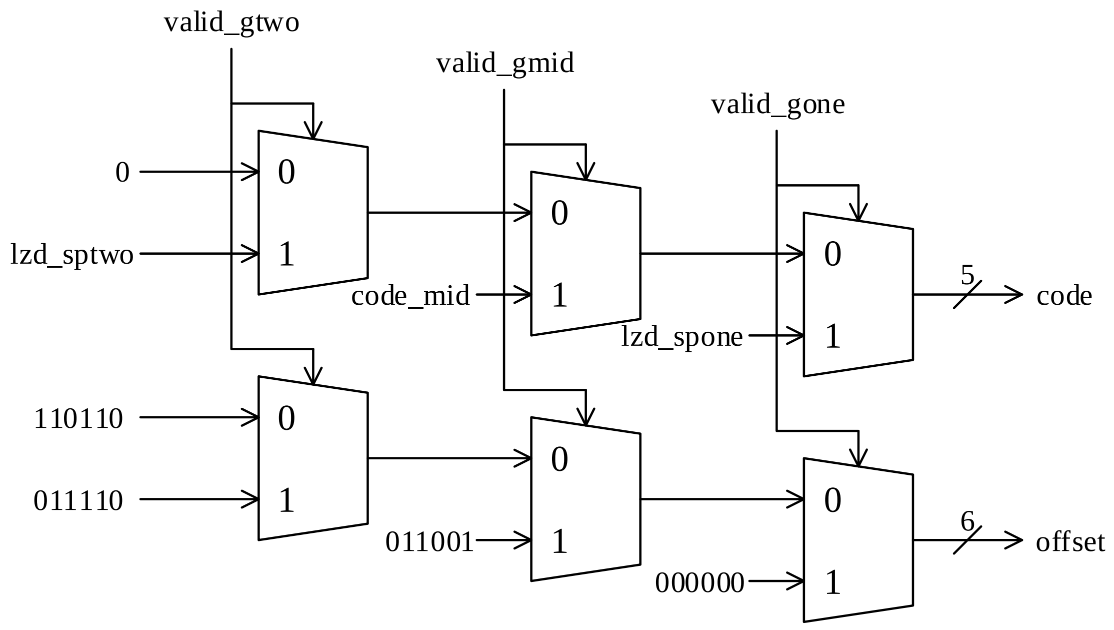


Figure 5.10: DP priority encoder in the leading zero anticipation and counting (LZAC) unit

error, one more shifting stage is added. In that stage, the mantissa from the previous stage is detected. If the MSB is 1, it is already normalized. Otherwise, another 1-bit shifting is needed. The exponent needs to be adjusted accordingly.

The SP and DP operation share the shifting logic. The DP operation shares the final stage normalization detection logic and the exponent adjustment logic with the first SP operation. Extra final stage normalization detection logic and the exponent adjustment logic are needed for the second SP operation.

5.7 Path Selection

In the last stage, the outputs from the FAR path and the NEAR path are selected by the signal u_n1 and u_n2. If u_n1 or u_n2 is 1, the outputs from NEAR path is selected. Otherwise, the FAR path outputs are selected.

Compared to DP only adder, extra logic is needed to select the exponent and sign for the second SP operation.

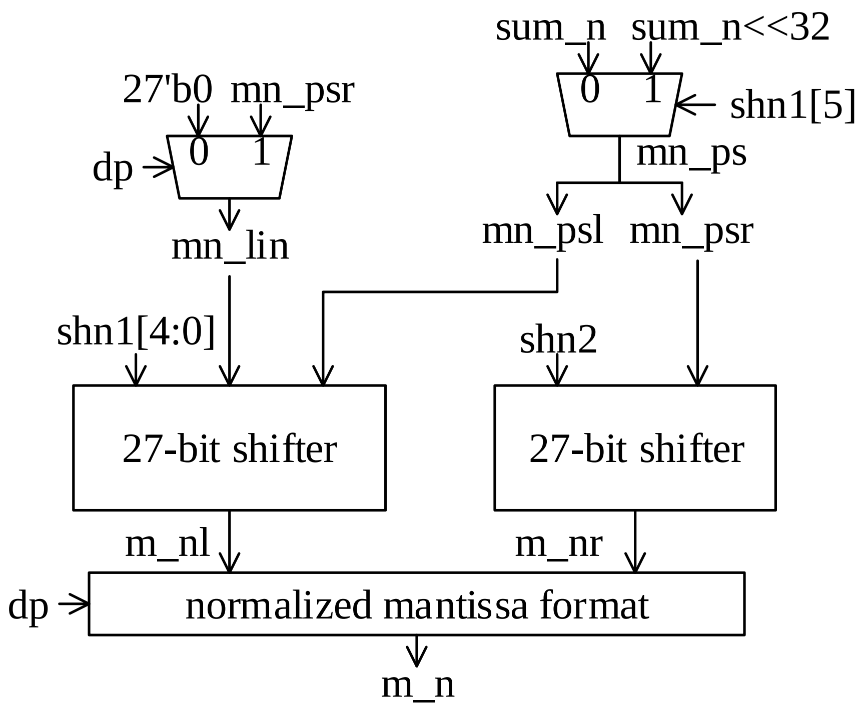


Figure 5.11: Normalization shifter of the proposed FP adder

Finally the selected sign, exponent, and mantissa are packed according to the IEEE FP format [1]. In DP mode, the result is one 64-bit DP number. In SP mode, the result is two 32-bit SP numbers.

The proposed FP adder architecture is mapped on Xilinx Virtex-5 and Virtex-7 and Altera Stratix-III and Arria-10 devices. Only LUT resources are used in the proposed architecture, therefore, different from the proposed multiplier, the same architecture is mapped on all these four devices.

5.8 Summary

In this chapter, the proposed FP adder architecture is presented. The proposed FP adder is designed based on the two-path FP addition algorithm. Both SP and DP addition/subtraction operations are supported and one DP addition or two parallel SP addition operations can be accomplished in 6 clock cycles. With fully pipelined architecture, after the latency of the first operation, new results can be generated every clock cycle.

The design of each modules is presented in detail. Each component of the proposed adder is optimized for mapping on FPGA devices in order to fully utilize FPGA logic resources. The SP and DP operations are designed to share hardware between each other in order to reduce area.

Chapter 6
Result and Analysis

This chapter presents the implementation results of the proposed multiplier and adder architectures.

In Section 6.1, the implementation results of the proposed FP multiplier on Xilinx Virtex-5 FPGA are presented. The comparison of the speed and resource usage of the proposed architecture with those of the previous designs are presented and analyzed. Moreover, the implementation results of the proposed multiplier architecture on Xilinx Virtex-7 and Altera Arria-10 devices are also provided.

In Section 6.2, the implementation results of the proposed FP adder on Xilinx Virtex-5 and Altera Stratix-III devices are presented. The speed and resource usage of the proposed adder are compared with those of the DP+2SP architecture built with: 1) the same data-path as the proposed architecture; 2) the Xilinx FP operator; 3) the Altera FP megafunction. The implementation results of the proposed adder on Xilinx Virtex-7 and Altera Arria-10 devices are also provided.

6.1 The Proposed FP Multiplier

The Verilog HDL model of the proposed multiplier architecture is created in order to verify and analyze the performance of the proposed multiplier design. For the 27×27 mantissa multipliers, instead of inferring them in Verilog code, the DSP instantiation template (for Xilinx) [43] [44] and the Quartus II DSP IP Catalog and Parameter Editor (for Altera) [38] are applied in order to ensure the mapping of the mantissa multipliers is exactly the same as discussed in Chapter 4. The functionality of the proposed design is verified by using a SystemVerilog testbench with random testing vectors.

Table 6.1: Delay and area of each pipeline stage of the proposed multiplier on Virtex-5 device

Pipeline Stage	Delay (ns)	Area	
		LUTs	Ratio
1	2.568	474 + 6 DSP	42.4%[1]
2	2.568		
3	2.568		
4	2.568		
5	2.568		
6	2.768	291	26.1%
7	2.87	128	11.4%
8	2.878	224	20.1%
Total	2.878	1117 + 6 DSP	100%

[1] LUT ratio only

6.1.1 Results on Xilinx Virtex-5 Device

Each Pipeline Stage

In order to compare with previous works, the proposed multiplier architecture is implemented on Xilinx Virtex-5 FPGA device.

Each pipeline stage of the proposed multiplier architecture for Xilinx FPGA is compiled using Xilinx ISE 14.7 on Virtex-5 (xc5vlx155ff1760-3) device. The delay and area of each pipeline stage on Virtex-5 device is shown in Table 6.1. Note that the first 5 pipeline stages contain DSP blocks, as shown in Figure 4.1, that cannot be divided and synthesized in each pipeline stage. Therefore, the first 5 pipeline stages are synthesized and reported in Table 6.1 as a whole. Each of the eight pipeline stages has similar delay value, therefore the pipeline stages of the proposed multiplier are balanced.

As shown in Table 6.1, the maximum delay falls in the last pipeline stage where the time-consuming DP rounding module is implemented. In the proposed architecture, although the compound adder is applied and the rounding operation is merely a selection among *sum*, *sum* + 1, and *sum* + 2 as shown in Section 4.1.6, the whole *dp round and normalization* module is still time-consuming because of the long OR operation to generate the sticky-bit, the complicated logic operation to generate the rounding decision, and the normalization

Table 6.2: Implementation results of the proposed multiplier on Virtex-5 device

Design	DSP	LUTs	FFs	Frequency
Proposed Multiplier	6	1117	945	348 MHz

shifting and exponent correction. In addition, the 7th pipeline stage takes long time because of the three 54-bit long adders. Although the adder is implemented using the dedicated carry logic in the slice, the 54-bit long carry propagation still needs long time.

In terms of area, the first five pipeline stages occupy more than 40% of the total LUTs. This is because in addition to the operands processing logics, the 54-bit adder, the 48-bit subtractor, and the 13-bit and 26-bit shifters incorporated with the DSP blocks, as shown in Figure 4.5, occupy large area. Besides, the 6th and 8th pipeline stages occupy large area because of the SP and DP rounding and normalization operations. For rounding, a complicated logic is required to generate the rounding decision which takes up large amount of LUTs. For the normalization, although only 1-bit shifting is required, the SP rounded mantissa (25-bit) and DP rounded mantissa (54-bit) have large length and therefore the shifting needs large amount of multiplexers.

The Whole Design

The whole proposed multiplier architecture is synthesized, placed and routed on Xilinx Virtex-5 (xc5vlx155ff1760-3) FPGA device using Xilinx ISE 14.7. The post place and route implementation results are shown in Table 6.2. The clock frequency can run up to 348 MHz, and the proposed multiplier requires 6 DSP48E blocks, 1117 LUTs, and 945 Flip-Flops (FFs). One DP operation can be accomplished in two iterations with a latency of 9 clock cycles and two parallel SP operations can be accomplished in one iteration with a latency of 6 clock cycles.

Comparison with Previous Works

The comparison of the proposed design with previous works is shown in Table 6.3. In order to better illustrate the comparison, the comparison of normalized area and timing (the ratios

Table 6.3: Area and timing comparison of the proposed multiplier with previous works

Design	Latency	ratio	DSP48E	ratio	LUTs	ratio	FFs	ratio	Frequency (MHz)	ratio
Xilinx [33] in 2012	18	2[1]	10	1.67	339	0.3	482	0.51	319	0.92
Banescu [40] in 2011	14	1.55[1]	9	1.5	804	0.72	804	0.85	407	1.17
	13	1.44[1]	9	1.5	1184	1.05	1080	1.14	407	1.17
Jaiswal [25] in 2013	7 (SP) 12 (DP)	1.16 1.33	9	1.5	1168	1.05	1373	1.45	336	0.96
Proposed in 2015	6 (SP) 9 (DP)	1	6	1	1117	1	945	1	348	1

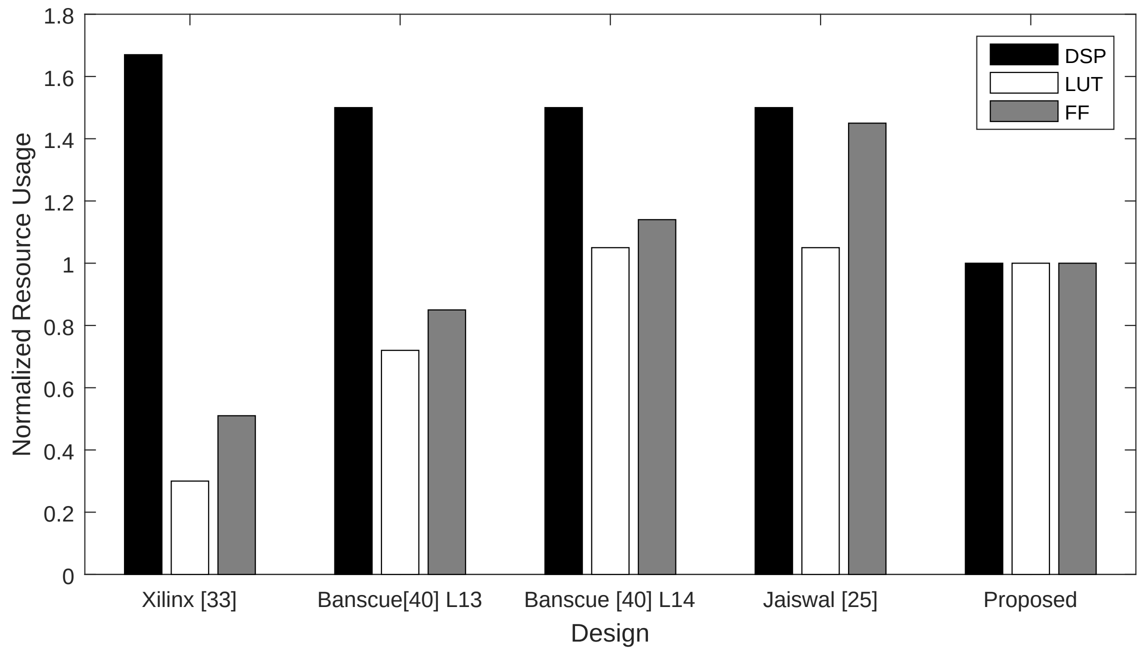
[1] Compared to the DP operation of the proposed design

shown in Table 6.3) of the proposed design with previous works is also graphically shown in Figure 6.1. The architectures design in [33] and [40] only support DP multiplication. In [25], the authors present the SP and DP merged multiplier architecture for both normalized number and subnormal number. In order to make fair comparison, only the architecture for normalized numbers is considered in this thesis.

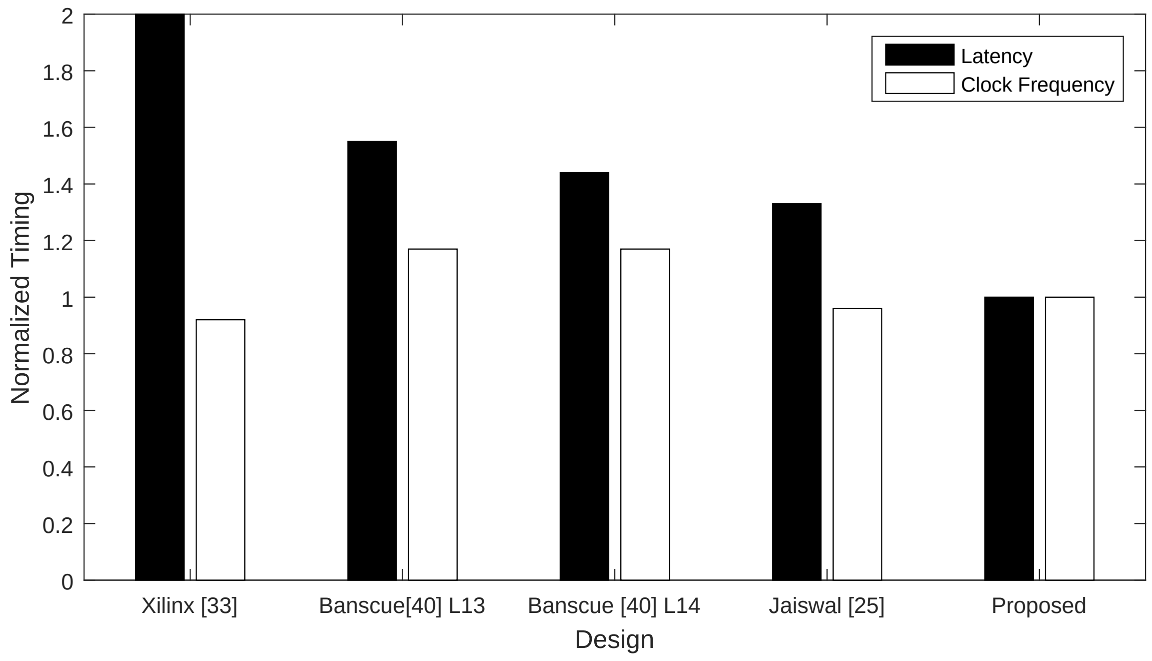
Compared to [33], the proposed multiplier needs 40% less DSP blocks, and has 50% less latency and 9% faster clock frequency for DP operation, however, it occupies more LUTs and registers. In [33], five stages of DSP blocks (instead of three in the proposed design) are cascaded, therefore, more pipeline stages are needed that leads to a large latency. The proposed design needs more LUTs resource than [33], because it needs extra logics to support both SP and DP operations. Moreover, all major computations are done with DSP blocks in [33], however, in the proposed architecture, the SP and DP incrementation as well as the DP partial product reduction and final stage addition are done with LUTs resources.

Compared to [40], the proposed design needs 3 less DSP blocks. Although the proposed design has slower clock frequency, it needs 5 (or 4) less clock cycles to accomplish one DP operation. Besides, when the design in [40] is implemented with the latency of 13, it needs 5% more LUTs and 14% more Flip-Flops than the proposed design.

Compared to [25], in terms of area, the proposed multiplier requires 33% less DSP blocks usage as well as 4.3% less LUTs and 31.2% less Flip-Flops. In terms of speed, the proposed design has 3.6% faster clock frequency and 1 less clock cycle for SP operations and 3 less clock



(a) Area comparison



(b) Timing comparison

Figure 6.1: Normalized area and timing comparison of the proposed multiplier with previous works (Banescu [40] L13 represents the design in [40] with a latency of 13, and L14 represents the design in [40] with a latency of 14)

Table 6.4: Implementation results of the proposed multiplier on Virtex-7 device

	DSP	LUTs	FFs	Frequency
Proposed Multiplier	6	952	973	427.35 MHz

cycles for DP operations. The design in [25] actually uses two levels of Karatsuba algorithm to achieve one DP operation. Therefore, more stages of DSP blocks are concatenated together that leads to more pipeline stages and DSP blocks usage.

6.1.2 Results on Xilinx Virtex-7 Device

In order to show the implementation results of the proposed multiplier on latest FPGA devices, the proposed architecture is also implemented on Xilinx Virtex-7 device and Altera Arria-10 device.

On Xilinx Virtex-7 (xc7v585tffg1761-3) device, the proposed multiplier is synthesized, placed and routed using Xilinx Vivado 2015.2. The post place and route results are shown in Table 6.4. On Virtex-7 device, the proposed multiplier can run up to 427.35 MHz, occupies 6 DSP48E1, 952 LUTs, and 973 Flip-Flops. On DP operation can be accomplished in two iterations with a latency of 9 clock cycles and two parallel SP operations can be accomplished in one iteration with a latency of 6 clock cycles.

The performance improvement on Virtex-7 device compared to Virtex-5 device comes from two reasons. On one hand, Virtex-7 is fabricated with 28 nm technology while Virtex-5 is with 65 nm technology. The logic circuits can run much faster in 28 nm technology than in 65 nm technology. On the other hand, the Vivado tool has better synthesis strategy compared to ISE tool so that the mapped logic is more compact and the routing delay can be reduced.

6.1.3 Results on Altera Arria-10 Device

In order to implement the proposed multiplier on Altera Arria-10 device, the architecture of the mantissa multipliers are modified as shown in Figure 4.11. The modified architecture is synthesized, placed and routed on Altera Arria-10 (10AX090S1F45I1SG) device using Altera

Table 6.5: Implementation results of the proposed multiplier on Arria-10 device

	DSP	ALUTs	Registers	Frequency
Proposed Multiplier	2	789	730	438.8 MHz

Quartus II 15.0. The post place and route implementation results are shown in Table 6.5. On Arria-10 device, the proposed multiplier can run up to 438.8 MHz, requiring 2 DSP blocks, 789 ALUTs, and 730 registers. One DP operation can be accomplished in two iterations with a latency of 7 clock cycles and two parallel SP operations can be accomplished in one iteration with a latency of 4 clock cycles.

Altera Arria-10 device is fabricated with 20 nm technology where the logic circuit can run very fast. In addition, the DSP block in Arria-10 device supports 27×27 unsigned multiplication, therefore, one 27×27 mantissa multiplier can be achieved by only one DSP blocks. As a result, the LUTs incorporated with the DSP blocks as in Xilinx architecture can be saved.

6.2 The Proposed FP Adder

The Verilog HDL model of the proposed adder architecture is created in order to verify and analyze the performance of the proposed adder design. The functionality of the proposed adder is verified by using a SystemVerilog testbench with random testing vectors.

6.2.1 Results on Xilinx Virtex-5 Device

Each Pipeline Stage

Each pipeline stage of the proposed adder architecture is compiled using Xilinx ISE 14.7 on Xilinx Virtex-5 (xc5vlx155ff1760-3). The area and delay of each pipeline stage is shown in Table 6.6. Each of the 6 pipeline stages has similar delay value, therefore the pipeline stages of the proposed adder are balanced.

As shown in Table 6.6, the maximum delay falls in the fifth pipeline stage where the time-consuming rounding and normalization are implemented. The rounding is time-consuming

Table 6.6: Delay and area of each pipeline stage of the proposed adder on Virtex-5 device

Pipeline	Delay	Area	
Stage	(ns)	LUTs	ratio
1	2.86	133	7.87%
2	2.91	310	18.32%
3	2.906	415	24.51%
4	2.947	162	9.6%
5	2.973	602	35.5%
6	2.361	72	4.2%
Total	2.973	1694	100%

because it needs a complicated logic to generate the rounding decision. The normalization takes long time because the near path normalization shifter contains 7 levels of shifting operation (6 level for normalization shifting and 1 more level to correct the LZAC error).

In terms of area, the third and fifth pipeline stages occupy large area because the major part of alignment shifter and normalization shifter are located in these two pipeline stages. The alignment shifter has 4-level shifting operation in the third pipeline stage and in each level the shifting operand is 55-bit long. The normalization shifter has 7-level shifting operations and on each level the shifting operand is 54-bit long. Therefore large amount of multiplexers are required for these two shifters that leads to larger area.

The Whole Design

The whole proposed adder architecture is synthesized, placed and routed on Xilinx Virtex-5 (xc5vlx155ff1760-3) FPGA device using Xilinx ISE 14.7. The post place and route implementation results are shown in Table 6.7. The clock frequency can run up to 336.36 MHz, and the proposed multiplier requires 1694 LUTs, and 1420 Flip-Flops. Both SP and DP operation needs 6 clock cycles to accomplish. However, with the fully pipelined architecture, after the latency of the first operation, new results can be obtained every clock cycle.

Table 6.7: Implementation results of the proposed adder on Virtex-5 device

Design	LUTs	FFs	Frequency
Proposed Adder	1694	1420	336.36 MHz

Table 6.8: Area and timing comparison of the proposed adder with DP+2SP architectures on Virtex-5 device

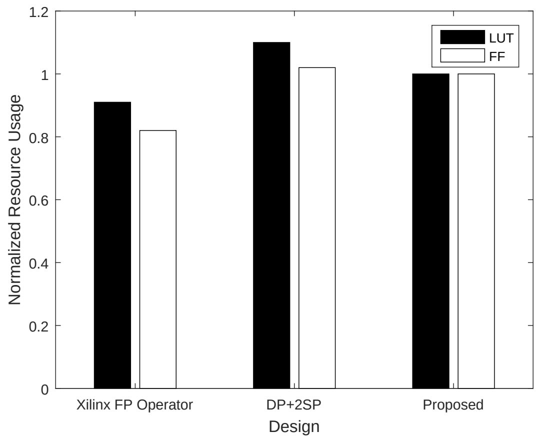
Design	Latency	LUTs	Ratio	FFs	Ratio	Frequency (MHz)	Ratio
Xilinx FP Operator	6	1541	0.91	1158	0.82	302.21	0.89
DP+2SP	6	1868	1.1	1441	1.02	346.74	1.03
Proposed Adder	6	1694	1	1420	1	336.36	1

Comparison

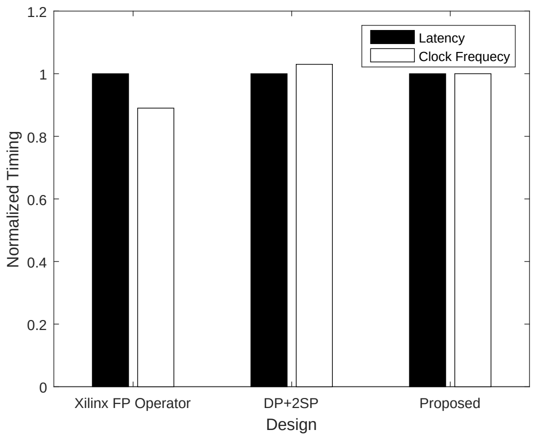
To the best of our knowledge, there is no published work on such SP and DP merged adder designed on FPGA. On the other hand, researchers can build an architecture that has the same function as the proposed design by combining two SP adders and on DP adder (DP+2SP). In order to show the advantage of the proposed adder, the DP+2SP architecture is built with 1) the same data-path as the proposed design; 2) Xilinx FP operator. Xilinx FP operator is the FP arithmetic solutions provided by Xilinx on their FPGA devices.

The comparison of the proposed design with the DP+2SP architectures are shown in Table 6.8. The comparison of normalized area and timing of the proposed adder with the DP+2SP architectures are also graphically shown in Figure 6.2. When using Xilinx FP operator, both the SP and DP adders are configured to high speed mode and only the LUTs are used. Moreover, the latency is set to 6.

Compared to the DP+2SP architecture with the same data-path, the proposed merged design has 9.3% less LUTs and 1.5% less Flip-Flops with only 2.9% timing overhead. The area is smaller because in the proposed adder design, the DP and SP adders are merged together that they can share some hardware with each other. In terms of delay, the critical path of the DP+2SP architecture falls in the data-path of the DP adder. However, in the



(a) Area comparison



(b) Timing comparison

Figure 6.2: Normalized area and timing comparison of the proposed adder with DP+2SP architectures on Virtex-5 device

proposed adder architecture, in order to also support SP operation, extra multiplexers are added to select between SP and DP operations. Therefore, the delay of the proposed adder is larger than the DP+2SP architecture with the same data-path.

Compared to the DP+2SP architecture built with Xilinx FP operator, the proposed adder has 11.3% faster clock frequency. However, it needs 9.92% more LUTs and 22% more Flip-Flops.

6.2.2 Results on Altera Stratix-III Device

Each Pipeline Stage

Each pipeline stage of the proposed adder architecture is compiled using Altera Quartus II 13.0 on Altera Stratix-III (EP3SL340F1760C2). The area and delay of each pipeline stage is shown in Table 6.9. Each of the 6 pipeline stages has similar delay value, therefore the pipeline stages of the proposed adder are balanced.

As shown in Table 6.9, the maximum delay falls in the fifth pipeline stage where the time-consuming rounding and normalization are implemented. The rounding is time-consuming because it needs a complicated logic to generate the rounding decision. The normalization takes long time because the near path normalization shifter contains 7 levels of shifting

Table 6.9: Delay and area of each pipeline stage of the proposed adder on Stratix-III device

Pipeline	Delay	Area	
Stage	(ns)	ALUTs	ratio
1	2.65	143	8.52%
2	2.69	299	17.72%
3	2.54	428	25.36%
4	2.73	166	9.85%
5	2.78	585	34.68%
6	1.86	65	3.87%
Total	2.78	1686	100%

Table 6.10: Implementation results of the proposed adder on Stratix-III device

Design	ALUTs	Registers	Frequency
Proposed Adder	1686	1556	358.42 MHz

operation (6 level for normalization shifting and 1 more level to correct the LZAC error).

In terms of area, the third and fifth pipeline stages occupy large area because the major part of alignment shifter and normalization shifter are located in these two pipeline stages. The alignment shifter has 4-level shifting operation in the third pipeline stage and in each level the shifting operand is 55-bit long. The normalization shifter has 7-level shifting operations and on each level the shifting operand is 54-bit long. Therefore large amount of multiplexers are required for these two shifters that leads to larger area.

The Whole Design

The whole proposed adder architecture is synthesized, placed and routed on Altera Stratix-III (EP3SL340F1760C2) device using Altera Quartus II 13.0. The post place and route implementation results are shown in Table 6.10. On Stratix-III device, the proposed adder can run up to 358.42 MHz, occupying 1686 ALUTs and 1556 registers. Both SP and DP

Table 6.11: Area and timing comparison of the proposed adder with DP+2SP architectures on Stratix-III device

Design	Latency	ALUTs	Ratio	Registers	Ratio	Frequency (MHz)	Ratio
Altera Megafunction	6	2416	1.43	1520	0.97	321.03	0.89
DP+2SP	6	1746	1.04	1760	1.13	362.71	1.01
Proposed Adder	6	1686	1	1556	1	358.42	1

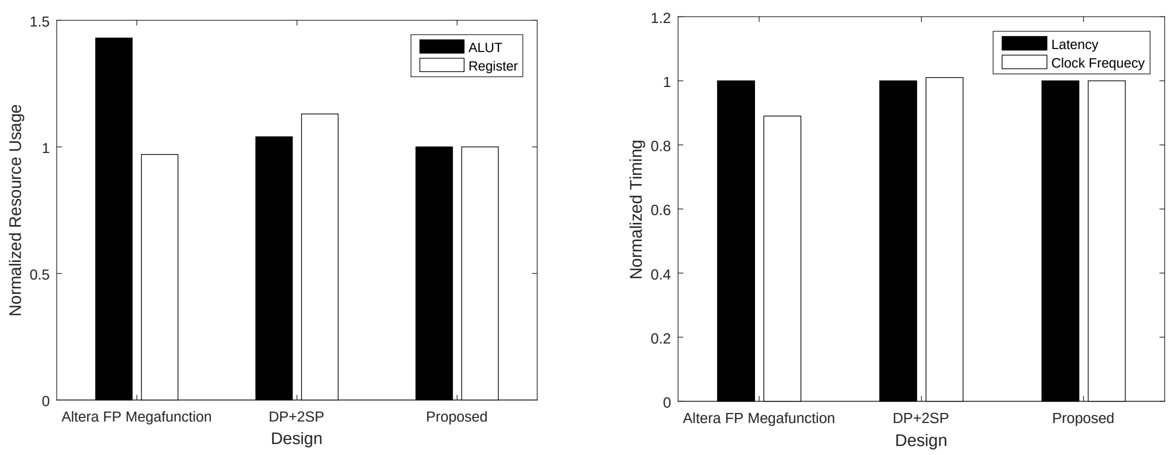
operations need 6 clock cycles to accomplish. However, with the fully pipelined architecture, after the latency of the first operation, new results can be obtained every clock cycle.

Comparison

Altera provides the FP megafunction as their FP arithmetic solutions on their FPGA devices. The comparison here is to build the DP+2SP architecture with 1) the same data-path as the proposed design; 2) Altera FP megafunction. The comparison of the proposed design with the DP+2SP architectures are shown in Table 6.11. The comparison of normalized area and timing of the proposed adder with the DP+2SP architectures are also graphically shown in Figure 6.2. When using Altera FP megafunction, both the SP and DP adders are set to speed optimized and no DSP are used. In addition, the latency is set to 6.

Compared to the DP+2SP architecture with the same data-path, the proposed merged design has 3.4% less LUTs and 11.5% less Flip-Flops with only 1.2% timing overhead. The area is smaller because in the proposed adder design, the DP and SP adders are merged together that they can share some hardware with each other. In terms of delay, the critical path of the DP+2SP architecture falls in the data-path of the DP adder. However, in the proposed adder architecture, in order to also support SP operation, extra multiplexers are added to select between SP and DP operations. Therefore, the delay of the proposed adder is larger than the DP+2SP architecture with the same data-path.

Compared to the DP+2SP architecture built with Altera megafunction, the proposed adder has 11.6% faster clock frequency. In terms of area, the proposed adder has 30.2% less



(a) Area comparison

(b) Timing comparison

Figure 6.3: Normalized area and timing comparison of the proposed adder with DP+2SP architectures on Stratix-III device

Table 6.12: Implementation results of the proposed adder on Virtex-7 device

	LUTs	FFs	Frequency
Proposed Adder	1859	1509	421.2 MHz

LUTs and 2.3% more Flip-Flops.

6.2.3 Results on Xilinx Virtex-7 Device

In order to show the implementation results of the proposed adder on the latest FPGA devices, the proposed architecture is also implemented on Xilinx Virtex-7 and Altera Arria-10 devices.

On Xilinx Virtex-7 (xc7v585tffg1761-3) device, the proposed adder is synthesized, placed and routed using Xiilinx Vivado 2015.2. The post place and route results are shown in Table 6.12. On Virtex-7 devices, the proposed adder can run up to 421.2 MHz, occupying 1859 LUTs and 1509 Flip-Flops. Both SP and DP operations need 6 clock cycles to accomplish.

Similar to the proposed multiplier implemented on Virtex-7 device, the performance improvement compared to Virtex-5 device is due to two factors: the 28 nm technology that Virtex-7 device is fabricated with can provide faster speed for logic circuits; the smarter synthesis strategy that the Vivado tool uses can help reduce the routing delay.

Table 6.13: Implementation results of the proposed adder on Arria-10 device

	ALUTs	Registers	Frequency
Proposed Adder	1396	1487	439.56 MHz

6.2.4 Results on Altera Arria-10 Device

The proposed adder architecture is also implemented on the Altera Arria-10 device. The proposed adder architecture only use LUT resources, therefore there is no need to modify the architecture when implementing on Arria-10 devices.

The proposed adder architecture is synthesized, placed and routed on Altera Arria-10 (10AX090S1F45I1SG) device using Altera Quartus II 15.0. The post place and route implementation results are shown in Table 6.13. On Arria-10 device, the proposed adder can run up to 439/56 MHz and requires 1396 ALUTs and 1487 registers. Both SP and DP operations need 6 clock cycles to accomplish.

The improvement of performance compared to Stratix-III device can be explained with two reasons. First the Altera Arria-10 device is fabricated with 20 nm technology that can provide fast speed for logic circuits. The second reason is that the new Altera Quartus II 15.0 software uses smarter synthesis strategy that can better optimize the logic circuit to reduce the routing delay and resource usage.

Chapter 7
Conclusion and Future Work

7.1 Conclusion

In this thesis, the design and implementation of an efficient single-precision (SP) and double-precision (DP) merged floating-point (FP) multiplier and a high performance SP and DP merged FP adder is presented. The proposed FP multiplier and FP adder are designed in order to efficiently support the multiplication and addition operations of multiple-precision with unified architectures, respectively.

In the proposed FP multiplier, the iterative multiplication method is applied in order to reduce the area of the proposed architecture. The proposed multiplier can accomplish two parallel SP multiplications in one iteration or one DP multiplication in two iterations. On Xilinx Virtex-5 and Virtex-7 devices, the complete SP operation needs 6 clock cycles to finish and the DP operation needs 9 clock cycles to accomplish. Whereas on Altera Arria-10 device, the SP multiplication requires 4 clock cycles and the DP multiplication needs 7 clock cycles. DSP blocks are used when mapping the proposed architecture on FPGA devices in order to reduce the look-up table (LUT) usage and improve the performance. Besides, when mapping the proposed multiplier architecture on Xilinx FPGA, the Karatsuba algorithm is applied in order to further reduce the DSP block usage. On Virtex-5 device, compared to the previous work of SP and DP merged FP multiplier on FPGA, the proposed multiplier has a 33% reduction of DSP block usage with 4% faster clock frequency.

In the proposed FP adder, the two-path FP addition algorithm is applied in order to obtain the best performance. The proposed adder supports both SP and DP addition/subtraction operations. With fully pipelined architecture, the proposed adder can accomplish on DP operation or two parallel SP operations with a latency of 6 clock cycles. Each component of the proposed adder is optimized for mapping on FPGA devices. The proposed adder is implemented on both 65 nm process Xilinx Virtex-5 and Altera Stratix-III devices. On both devices, compared to the combination of one DP and two SP adders (DP+2SP) built with the same data-path, the proposed adder can save area with limited timing overhead. On Virtex-5 device, the proposed adder runs 11.3% faster than the DP+2SP architecture built with Xilinx FP operator. Meanwhile, the proposed adder has 11.6% faster clock frequency than the DP+2SP architecture built with Altera FP megafunction on Stratix-III device.

The mapping solutions and implementation results of the proposed multiplier and adder on the latest Xilinx Virtex-7 and Altera Arria-10 FPGA devices are also provided. These two series of devices are fabricated with better semiconductor technology (28 nm technology for Virtex-7 and 20 nm technology for Arria-10). In addition, the two latest FPGA development tools, Xilinx Vivado 2015.2 and Altera Quartus II 15.0, use smarter synthesis strategies to implement the logic design. Therefore, on these two devices, there is a significant improvement on the performance of the proposed multiplier and adder.

By using the proposed multiplier architecture, more DSP blocks can be reserved for other functions or operations. Therefore, the proposed multiplier architecture is especially suitable for the low-end FPGA devices where the number of DSP blocks is quite limited. As the proposed adder is designed for high performance operation, therefore it is also suitable for FP applications where the performance is the first consideration.

7.2 Future Work

There are several extensions of this work that can be investigated in the future, including the support for subnormal numbers, the support for quadruple-precision (QP) numbers, and the multiple-precision fused multiply-add (FMA) unit design.

The FP multiplier and FP adder proposed in this thesis only support normalized numbers. The support for subnormal numbers can be added. Subnormal numbers can ensure that the subtraction of two close FP numbers can never be underflow and always has a representable value. Otherwise even though the values of these two numbers are not equal, the result of subtraction will be zero. Subnormal support is important in audio processing applications.

In audio applications, subnormal numbers represent very quiet signal. If subnormal operation is not supported, they will be cut to zero and the quality of the signal will be affected.

The design of unified FP multiplier and unified FP adder that also support QP operations can be investigated. In the upcoming big data era, more and more applications, such as financial calculation and aerospace applications, require QP operations. However, the software data processing solutions running on CPUs are not fast enough to process the big data dataset which is measured with petabyte or even exabyte. FPGA is a good candidate for CPU accelerator. FPGA implementation of SP, DP, and QP merged multiplier/adder is expected to provide a good solution to the QP operations. In addition, the unified architecture can be used in scientific and engineering applications where DP or SP operations are required.

The design of a multiple-precision FMA unit can also be considered. Compared to separate multiplier and adder units, the FMA unit has several advantages: 1) one multiplication and addition operation is performed with only one rounding instead of two. Therefore, the overall delay and the rounding error can be reduced; 2) the multiplier and the adder can share components with each other. Therefore, the total area can be reduced. As a result, in a single FPGA chip, more computing units can be placed that is expected to further improve the computing performance. Moreover, many applications contain large amount of $A + (B \times C)$ operations. These applications can be efficiently executed using FMA architecture. The merged-precision architecture can be explored in order to efficiently support multiple-precision operations for various kinds of applications.

In addition to the full precision FMA unit, the approximate FMA with slightly reduced precision can be investigated. In video and audio applications, because of human's perception, a slightly reduced precision will not affect the quality of service (QoS). Moreover, some applications, such as machine learning and biological applications, can tolerate the inexact data processing and produce acceptable results. With approximate computing architecture, significant area and power benefit can be obtained and the performance improvement can be expected.

References

[1] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2008 ed., IEEE, Aug. 2008.

[2] S. Oberman and M. Flynn, "Design issues in division and other floating-point operations," *Computers, IEEE Transactions on*, vol. 46, no. 2, pp. 154–161, Feb 1997.

[3] A. D. BOOTH, "A Signed Binary Multiplication Technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951. [Online]. Available: http://qjmam.oxfordjournals.org/content/4/2/236.abstract

[4] M. Uya, K. Kaneko, and T. J. Yasui, "A CMOS floating point multiplier," *Solid-State Circuits, IEEE Journal of*, vol. 19, no. 5, pp. 697–702, Oct. 1984.

[5] S.-R. Kuang, J.-P. Wang, and H.-Y. Hong, "Variable-Latency Floating-Point Multipliers for Low-Power Applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 10, pp. 1493–1497, Oct. 2010.

[6] P. M. Farmwald, "On the Design of High Performance Digital Arithmetic Units," Ph.D. dissertation, Stanford, CA, USA, 1981.

[7] S. F. Oberman, "Design Issues in High Performance Floating Point Arithmetic Units," Stanford, CA, USA, Tech. Rep., 1996.

[8] S. F. Oberman, H. Al-Twaijry, and M. Flynn, "The SNAP project: design of floating point arithmetic units," in *Computer Arithmetic, 1997. Proceedings., 13th IEEE Symposium on*, Jul. 1997, pp. 156–165.

[9] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. C. Lim, "Reduced latency IEEE floating-point standard adder architectures," in *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on*, 1999, pp. 35–42.

[10] A. M. Nielsen, D. W. Matula, C. N. Lyu, and G. Even, "An IEEE compliant floating-point adder that conforms with the pipeline packet-forwarding paradigm," *Computers, IEEE Transactions on*, vol. 49, no. 1, pp. 33–47, Jan. 2000.

[11] P.-M. Seidel and G. Even, "Delay-optimized implementation of IEEE floating-point addition," *Computers, IEEE Transactions on*, vol. 53, no. 2, pp. 97–113, Feb. 2004.

[12] D. H. Bailey, "High-precision computation: Applications and challenges [Keynote I]," in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, Apr. 2013, p. 3.

[13] D. Tan, C. E. Lemonds, and M. J. Schulte, "Low-Power Multiple-Precision Iterative Floating-Point Multiplier with SIMD Support," *Computers, IEEE Transactions on*, vol. 58, no. 2, pp. 175–187, Feb. 2009.

[14] A. Akka and M. J. Schulte, "Dual-mode floating-point multiplier architectures with parallel operations," *Journal of Systems Architecture*, vol. 52, no. 10, pp. 549–562, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762106000361

[15] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient multiple precision floating-point multiplier," in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, Dec. 2011, pp. 153–156.

[16] M. K. Jaiswal, R. C. C. Cheung, M. Balakrishnan, and K. Paul, "Unified Architecture for Double/Two-Parallel Single Precision Floating Point Adder," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 61, no. 7, pp. 521–525, Jul. 2014.

[17] A. Akkas, "Dual-Mode Quadruple Precision Floating-Point Adder," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, 2006, pp. 211–220.

[18] A. Akka, "Dual-mode floating-point adder architectures," *Journal of Systems Architecture*, vol. 54, no. 12, pp. 1129–1142, 2008.

[19] S. Kestur, J. D. Davis, and O. Williams, "BLAS Comparison on FPGA, CPU and GPU," in *VLSI (ISVLSI), 2010 IEEE Computer Society Annual Symposium on*, Jul. 2010, pp. 288–293.

[20] A. H. T. Tse, D. Thomas, and W. Luk, "Design Exploration of Quadrature Methods in Option Pricing," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 5, pp. 818–826, May 2012.

[21] D. Strenski, "FPGA Floating Point Performance," Jan. 2007. [Online]. Available: http://www.hpcwire.com/2007/01/12/fpga_floating_point_performance/

[22] M. C. Smith, J. S. Vetter, and X. Liang, "Accelerating Scientific Applications with the SRC-6 Reconfigurable Computer: Methodologies and Analysis," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, Apr. 2005, pp. 157b–157b.

[23] W. N. Chelton and M. Benaissa, "Fast Elliptic Curve Cryptography on FPGA," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 198–205, Feb. 2008.

[24] K. C. C. Loi and S.-B. Ko, "Scalable Elliptic Curve Cryptosystem FPGA Processor for NIST Prime Curves," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2015.

[25] M. K. Jaiswal and R. C. C. Cheung, "Area-efficient architectures for double precision multiplier on FPGA, with run-time-reconfigurable dual single precision support," *Microelectronics Journal*, vol. 44, no. 5, pp. 421–430, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269213000591

[26] A. Karatsuba and Y. Ofman, "Multiplication of Many-Digital Numbers by Automatic Computers," in *Proceedings of the USSR Academy of Sciences*, vol. 145, 1962, pp. 293–294.

[27] *LogiCORE IP Floating-Point Operator v5.0 Product Specification*, DS335 ed., Xilinx, Mar. 2011.

[28] *Floating-Point Megafunctions User Guide*, UG-01058-7 ed., Altera, Nov. 2013.

[29] N. T. Quach and M. J. Flynn, "An Improved Algorithm for High-Speed Floating-Point Addition," Stanford University, Tech. Rep. CSL-TR-90-442, 1990.

[30] *Virtex-5 FPGA User Guide*, UG190(v5.4) ed., Xilinx, Mar. 2012.

[31] P. Zicari and S. Perri, "A fast carry chain adder for virtex-5 fpgas," in *MELECON 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*, April 2010, pp. 304–308.

[32] *7 Series FPGAs Configurable Logic Block User Guide*, UG474(v1.7) ed., Xilinx, Nov. 2014.

[33] *Virtex-5 FPGA XtremeDSP Design Considerations User Guide*, UG193(v3.5) ed., Xilinx, Jan. 2012.

[34] *Virtex-5 Data Sheet: DC and Switching Characteristics*, DS202(v5.4) ed., Xilinx, Dec. 2014.

[35] *XST User Guide for Virtex-4 , Virtex-5, Spartan-3, and Newer CPLD Devices*, UG627(v14.5) ed., Xilinx, Mar. 2013.

[36] *7 Series DSP48E1 Slice*, Xilinx, Nov. 2014.

[37] *Arria 10 Core Fabric and General Purpose I/Os Handbook*, Altera, May 2015.

[38] *Arria 10 Native Fixed Point DSP IP Core User Guide*, UG-01163 ed., Altera, Dec. 2014.

[39] *Arria 10 Device Datasheet*, Altera, May 2015.

[40] S. Banescu, F. de Dinechin, B. Pasca, and R. Tudoran, "Multipliers for Floating-point Double Precision and Beyond on FPGAs," *SIGARCH Comput. Archit. News*, vol. 38, no. 4, pp. 73–79, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1926367.1926380

[41] J. D. Bruguera and T. Lang, "Rounding in floating-point addition using a compound adder," University of Santiago de Compostela, Tech. Rep., Jul. 2000.

[42] M. S. Schmookler and K. J. Nowka, “Leading zero anticipation and detection-a comparison of methods,” in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, 2001, pp. 7–12.

[43] *Virtex-5 Libraries Guide for HDL Designs*, UG621(v14.7) ed., Xilinx, Oct. 2013.

[44] *Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for HDL Designs*, UG768(v14.7) ed., Xilinx, Oct. 2013.