Clemson University
TigerPrints

All Theses | Theses

12-2009

THE DESIGN OF AN IC HALF PRECISION FLOATING POINT ARITHMETIC LOGIC UNIT

Balaji Kannan
Clemson University, balaji.n.kannan@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Part of the Electrical and Computer Engineering Commons

Recommended Citation
Kannan, Balaji, "THE DESIGN OF AN IC HALF PRECISION FLOATING POINT ARITHMETIC LOGIC UNIT" (2009). *All Theses*. 689.
https://tigerprints.clemson.edu/all_theses/689

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

THE DESIGN OF AN IC HALF PRECISION FLOATING POINT ARITHMETIC LOGIC UNIT

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Balaji Navalpakkam Kannan
December 2009

Accepted by:
Dr. Kelvin Poole, Committee Chair
Dr. William Harrell
Dr. Rajendra Singh

ABSTRACT

A 16 bit floating point (FP) Arithmetic Logic Unit (ALU) was designed and implemented in 0.35µm CMOS technology. Typical uses of the 16 bit FP ALU include graphics processors and embedded multimedia applications.

The ALU of the modern microprocessors use a fused multiply add (FMA) design technique. An advantage of the FMA is to remove the need for a comparator which is required for a normal FP adder. The FMA consists of a multiplier, shifters, adders and rounding circuit. A fast multiplier based on the Wallace tree configuration was designed. The number of partial products was greatly reduced by the use of the modified booth encoder. The Wallace tree was chosen to reduce the number of reduction layers of partial products. The multiplier also involved the design of a pass transistor based 4:2 compressor. The average delay of the pass transistor based compressor was 55ps and was found to be 7 times faster than the full adder based 4:2 compressor. The shifters consist of separate left and right shifters using multiplexers. The shift amount is calculated using the exponents of the three operands.

The addition operation is implemented using a carry skip adder (CSK). The average delay of the CSK was 1.05ns and was slower than the carry look ahead adder by about 400ps. The advantages of the CSK are reduced power, gate count and area when compared to the similar sized carry look ahead adder. The adder computes the addition of the multiplier result and the shifted value of the addend.

In most modern computers, division is performed using software thereby eliminating the need for a separate hardware unit. FMA hardware unit was utilized to

perform FP division. The FP divider uses the Newton Raphson algorithm to solve division by iteration. The initial approximated value with five bit accuracy was assumed to be pre-stored in cache memory and a separate clock cycle for cache read was assumed before the start of the FP division operation. In order to significantly reduce the area of the design, only one multiplier was used. Rounding to nearest technique was implemented using an 11 bit variable CSK adder. This is the best rounding technique when compared to other rounding techniques. In both the FMA and division, rounding was performed after the computation of the final result during the last clock cycle of operation.

Testability analysis is performed for the multiplier which is the most complex and critical part of the FP ALU. The specific aim of testability was to ensure the correct operation of the multiplier and thus guarantee the correctness of the FMA circuit at the layout stage. The multiplier's output was tested by identifying the minimal number of input vectors which toggle the inputs of the 4:2 compressors of the multiplier. The test vectors were identified in a semi automated manner using Perl scripting language. The multiplier was tested with a test set of thirty one vectors. The fault coverage of the multiplier was found to be 90.09%.

The layout was implemented using IC station of Mentor Graphics CAD tool and resulted in a chip area of 1.96mm^2. The specifications for basic arithmetic operations were met successfully. FP Division operation was completed within six clock cycles. The other arithmetic operations like FMA, FP addition, FP subtraction and FP multiplication were completed within three clock cycles.

DEDICATION

To my beloved parents and God

ACKNOWLEDGEMENTS

I thank Dr. Kelvin Poole for showing me the path of higher learning. His faith in my abilities instilled a sense of confidence in me to pursue this thesis. The timely guidance helped me hone my skills in digital IC design.

I am grateful to Dr. William Harrell and Dr. Rajendra Singh for helping me understand the fundamental concepts of micro-electronics and for serving in the committee.

I owe my deepest gratitude to my parents for their support and faith in my abilities. They have provided me immense love and courage to face the challenges in life. I thank my friends Govind, Mrinal, Nandhu and Barath for their support and encouragement. Finally, I would also like to thank Pravin for his suggestions and comments during my graduate study at Clemson University.

TABLE OF CONTENTS

Page

TITLE PAGE i

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

LIST OF FIGURES x

LIST OF TABLES xiii

CHAPTER

I. INTRODUCTION 1

THESIS SPECIFICATION 1
CHAPTER OVERVIEW 2

II. FLOATING POINT NUMBER SYSTEM 4

IEEE FLOATING POINT NUMBER SYSTEM 4
HALF PRECISION FLOATING POINT NUMBER FORMAT 5
SPECIAL NUMBERS 6
Signed Zero 6
Subnormal numbers 7
Infinite and NaN numbers 7
ROUNDING TECHNIQUES 7
CONCLUSION 9

III. FUSED MULTIPLY ADD UNIT 10

CHOICE OF FMA 11
ALGORITHM TO PERFORM FMA BASED COMPUTATION 13
Step1 (Multiplication and Right Shift) 13
Step 2 (Addition and Count of Leading Zeros) 14
Step 3 (Normalization and Rounding) 14
CONCLUSION 14

IV. COMPONENTS USED IN THE FMA 15
Memory using D register 16
Multiplier 18
Implementation of multiplier 18
Components designed for use in the multiplier 21
Exponent Computation of multiplier 24
Thirty four bit right shifter 27
Addition of multiplier sum and carry 29
Thirty five bit adder 36
Inputs to the adder 37
Reasons for carry skip adder 39
Design of 35 bit adder 39
Leading zero counter 40
Implementation of LZC for FP ADD instruction 41
Algorithm for leading zero counters 43
Implementation of LZC 44
Implementation of normalization 48
Exponent computation before Rounding 49
Implementation of rounding 52
Final Exponent computation 53
Conclusion 55

V. FMA 56
Operation modes of the ALU 56
Design of a 3 bit counter 56
Non overlapping clocks 59
Local clock for addition and count of leading zeros 59
Clock for D register based memory 60
Clock gating for addition and leading zero counter 61
Trigger for D register based memory 63
Cycle based working of FMA 64
First Clock cycle of operation 65
Second clock cycle of operation 67
Third Clock cycle of operation 70
Conclusion 71

VI. TESTABILITY 72
Objective 72
Behavior model based Test 73
Identification of critical component 73
Multiplier testability 74
Hardware Circuit for test 74
Steps for identification of input vectors 75

CONCLUSION 78

VII. FP DIVISION 79

INTRODUCTION TO DIVISION 79
Inputs for FP division 79
Iterative division 79
Faster Division 81
DIVISION ARCHITECTURE 82
Inputs selection to the multiplier 83
CYCLE BASED WORKING OF FP DIVISION 85
First clock cycle of operation 85
Second clock cycle of operation 88
Third clock cycle of operation 91
Fourth clock cycle of operation 92
Fifth clock cycle of operation 92
Sixth Clock cycle of operation 92
CONCLUSION 93

VIII. RESULTS AND CONCLUSION 94

4:2 COMPRESSOR 94
RESULTS OF FMA 95
Case 1 $(A * B + C)$ 95
Case 2 $(-A * -B - C)$ 99
RESULT OF FP DIVISION 100
FP EXCEPTIONS 102
FP Underflow 103
FP Overflow 104
CONCLUSION 106

APPENDICES 108

A. APPENDIX A 109

B. APPENDIX B 111

C. APPENDIX C 113

D. APPENDIX D 114

REFERENCES 115

LIST OF FIGURES

Figure 2-1 : Representation of 16 bit floating point number 5

Figure 3-1: Architecture of classic FMA 12

Figure 4-1: Trigger to D register for FMA mode of operation 17

Figure 4-2 : Trigger to D register for FP Division 17

Figure 4-3 : Inputs to the Booth Encoder 19

Figure 4-4 : Schematic of the Modified Radix-4 Booth Encoder 20

Figure 4-5 : Dot diagram of tree reduction of multiplier 21

Figure 4-6 : Schematic of the pass transistor based multiplexer 22

Figure 4-7 : Schematic of the full adder based 4:2 compressor 23

Figure 4-8 : Schematic of pass transistor based 4:2 compressor 24

Figure 4-9 : Dot diagram of the exponent computation 25

Figure 4-10 : Schematic of the exponent calculation of multiplier 26

Figure 4-11 : Schematic of anticipatory logic for exponent 27

Figure 4-12 : Schematic of 34 bit right shifter 28

Figure 4-13 : Schematic to compute the shift distance 29

Figure 4-14 : Schematic of a single carry skip block 36

Figure 4-15 : Schematic of inputs to the adder 38

Figure 4-16 : Example of input pattern to the 35 bit adder 42

Figure 4-17 : Block diagram of Leading Zero Counter for entire 35 bits of data 45

Figure 4-18 : Schematic of 35 bit leading zero counter 45

Figure 4-19 : Schematic of 3 bit LZC and zero flag 46

Figure 4-20 : Schematic of the adder to count the leading zeros....................................48

Figure 4-21 : Schematic of normalization output 50

Figure 4-22 : Procedure of final exponent computation...51

Figure 4-23 : Schematic of the circuit that performs $G\ (R + T + L)$.............................53

Figure 4-24 : Schematic of increment in exponent 54

Figure 4-25 : Schematic of computation of sign of result 55

Figure 5-1 : Schematic of the 3 bit counter 58

Figure 5-2 : Schematic of hold generation signal 58

Figure 5-3 : Timing diagram illustrating the non overlap time between various clocks...59

Figure 5-4 : Schematic of clock gating circuits. 61

Figure 5-5 : Clock gating for adder, leading zero counter and rounding logic for FMA..62

Figure 5-6 : Schematic of the control logic to select inputs to the multiplier 64

Figure 5-7 : Block diagram of FMA 65

Figure 5-8: Schematic of the control signals for selecting inputs to the multiplier from various sources...67

Figure 5-9 : Schematic of the control logic used for the adder.......................................68

Figure 5-10 : Schematic of the control logic for effective addition/subtraction...............69

Figure 6-1 : Layout and schematic symbol of 11 X 11 bit multiplier 78

Figure 7-1 : Block Diagram of division architecture 82

Figure 7-2 : Schematic of inputs to the multiplier for a single bit 84

Figure 7-3 : Schematic of inverse approximation's exponent calculation 87

Figure 7-4 : Schematic of exponent part in the rows of D registers 88

Figure 7-5 : Schematic of control signal for one bit left shift..90

Figure 7-6: Schematic of update of exponent for one bit left shift 91

Figure 8-1 : Layout of pass gate multiplexer..94

Figure 8-2 : Waveform showing the delay comparison 95

Figure 8-3 : Waveform of first clock cycle 96

Figure 8-4 : Waveform of the second clock cycle 97

Figure 8-5 : Waveform of the third clock cycle...98

Figure 8-6 : Waveform of $(-A\,X - B - C)$ 100

Figure 8-7 : Waveform of FP division 101

Figure 8-8 : Schematic of circuits that handle FP exceptions......................................102

Figure 8-9 : Schematic of update of exponent and significand in case of FP exception. 103

Figure 8-10 : Waveform of underflow exception in FMA 104

Figure 8-11: Waveform of overflow exception in FP division.....................................105

Figure B- 1: Block diagram of three path FMA …………………………………………...111

Figure B- 2: Block diagram of bridge FMA ……………………………………………111

Figure D- 1: Complete layout of ALU ………………………………………………..114

LIST OF TABLES

Table 2-1 : Table of various rounding modes in floating point arithmetic 8

Table 4-1 : Table of various components used in FMA and FP Division 15

Table 4-2: Table of modified Booth Encoder 19

Table 4-3 : Table of number of bits per group for various values of group number 35

Table 4-4 : Table of all possible operations in the adder 37

Table 4-5 : Table of number of bits per group for various values of group number) 40

Table 4-6 : Shift and predicted leading zero counter 43

Table 4-7 : Truth table of 3 bit counter 47

Table 4-8 : Table of possible cases of exponent of multiplier 50

Table 5-1 : Operational modes of the ALU 56

Table 5-2 : Table of counter values with corresponding clock cycles 57

Table 5-3 : Table of counter values for clock gating 60

Table 5-4 : Values stored in the D registers for the FMA operation 66

Table 6-1 : Table of percentage use of various hardware components for different operational modes 73

Table 6-2 : Table of input combinations to the 4:2 compressor 77

Table 7-1 : Table of inputs to multiplier 84

Table 7-2 : Table of rows of D registers that are used at appropriate clock cycles 85

Table 8-1 : Table of results for $A\,X\,B + C$ 98

Table 8-2 : Table of results for $(-A\,X - B) - C)$ 99

Table 8-3 : Table of results for FP division 101

Table 8-4 : Table of results of underflow in FMA..103

Table 8-5 : Table of overflow results in FP division..105

Table A-1: Table of 32 bit format for FP numbers ……………………………………..109

Table A- 2 : Table of 64 bit format for FP numbers………………………………………109

CHAPTER ONE

INTRODUCTION

The ALU represents the heart of the modern day computer which is a dedicated unit for performing arithmetic computations. The importance was outlined when it was proposed as a separate entity in the Von Neumann architecture. The ALU has evolved over the past few decades and has kept pace with advancements in the formats of numbering systems. The floating point ALU is used to perform scientific computations and graphics processing.

The fused multiply add unit is used to implement basic floating point arithmetic operations like addition, subtraction, multiplication and division. Various architectures are available to compute the fused multiply add operations. A half precision floating point ALU was designed with the available set of standard/custom cells to meet the specifications.

Thesis specification

The design a of half precision floating point ALU that computes the arithmetic operations like addition, subtraction, multiplication and division was given as the thesis specification.

Hardware compatible with FMA4 instruction set is specified. Half precision (16 bits) was implemented in this thesis because any increase in the operand size merely results in an increase in the size of the computational blocks. The method of computation is the same irrespective of the size of the operands. Inputs to the FP ALU are assumed to be normalized values with a hidden bit set as '1'.

The specification set the goal for addition, subtraction and multiplication as three clock cycles and the architecture for implementing the FMA as the IBM RS/6000.

FP division is the most complex among the basic arithmetic operations. The specification set the goal for FP division at a maximum of seven clock cycles. Five bit accuracy was used for the initial look up table. Thus 32 rows of memory are required to store the initial look up value in the cache. A single clock cycle for memory access was assumed before the actual computation for division. For simplicity, a single row of 11 bits of approximated value was stored in the memory using D registers.

A final step of rounding was performed all the above mentioned operations. Rounding to nearest technique was implemented to perform the rounding operation.

Chapter overview

The first chapter defines the thesis specification and provides a basic introduction about the ALU. The arithmetic operations that are completed as part of the thesis are briefly mentioned.

The second chapter briefly narrates the representation of half precision floating point numbers, various rounding techniques used in floating point operations and also the special numbers encountered in the 16 bit floating point representation.

The third chapter describes the FMA architecture used in this thesis. Operations like FP addition, FP subtraction and FP multiplication are implemented using the FMA architecture.

The fourth chapter describes the circuits used for the FMA. The choices of designs which were considered as part of the thesis are briefly mentioned. Components

like multiplier, adder, left/right shifters, leading zero counter and rounding logic are explained in this chapter.

Chapter five deals with the implementation of FMA. The operation of FMA is explained with a clock cycle based approach.

Chapter six deals with the implementation of FP division. The techniques for division are briefly mentioned. FP division is implemented using the existing FMA hardware unit.

The seventh chapter discusses the testability of the multiplier. The proper functioning of the multiplier will ensure the correctness of the FMA result.

The last chapter reports the simulation results of all operations. Layout and timing diagram using IC station of Mentor Graphics and EZwave are presented. The results of overflow and underflow FP exceptions are also presented for FMA and FP division. The theoretical background of exceptions is discussed in the second chapter.

The appendices describe the basic floating point number system, the various architectures in FMA, an example of capacitance calculation and the layout of the complete chip.

CHAPTER TWO

FLOATING POINT NUMBER SYSTEM

Floating-point arithmetic is a popular method of implementing real-numbers on modern computers. The radix/binary point is not fixed and can be placed anywhere with respect to the significand digits of the number. The main idea behind the floating point representation is that only a fixed number of bits are used and the binary point "floats" to wherever it is needed in those bits. As a matter of fact the computer only holds bit patterns. Floating point expressions can represent a wide span of numbers. When a floating point calculation is performed, the binary point floats to the correct position in the result. But the floating-point format requires slightly larger storage to represent the binary/radix point. The speed of floating-point operations is one of the important figures of merit for computers in many application domains. It is usually measured in FLOPS or floating point operations per second.

IEEE Floating point number system

The IEEE has standardized the representation for binary floating-point numbers as IEEE 754 and this standard is generally used in modern computers [1]. Precision denotes the accuracy of a given floating point number. There are several formats with increasing levels of accuracy. The format that is used depends on the end user experience and level of sophistication required for the application. In accordance with that, IEEE has updated the formats to meet various requirements and thus standardize calculations worldwide. The 16 (Half), 32 (Single), 64 (Double) and 128 (Quadruple) bits are the FP precisions

used in IEEE 754. The 32 and 64 bit implementations are the most widely used in the modern microprocessors.

Half precision floating point number format

The 16 bit functional units are used in graphics processors and for embedded multimedia applications [2, 3]. The precision is low for this format of floating point numbers. From a purely knowledge perspective, any FP operation performed in this format could be readily implemented in a higher format by increasing the size of the respective arithmetic blocks. Any such increase would result in minimal logical changes as the functionality would remain the same. Figure 2-1 shows the parts of any floating point number that conforms to the IEEE standard.

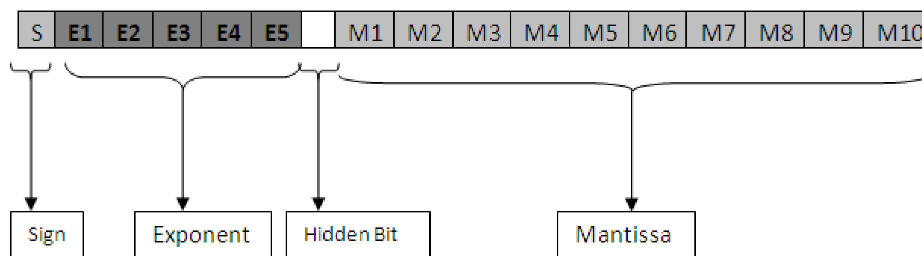


Figure 2-1 **:** Representation of 16 bit floating point number

Sign: The sign bit consists of a single bit. It can take one of the two values '0' or '1'. '0' indicates a positive number and '1' indicates a negative number

Exponent: The exponent consists of five bits. It indicates the exponent value with the bias.

Significand: The significand consists of eleven bits which includes a hidden bit. If significand is represented as 'x', the representation for a normalized number is '1.x'.

Range: Minimum value of Exponent is -14 and the Maximum value of Exponent is 15.

The bias for this format= 15

(Any value outside the range will be indicated by raising the exception flags)

True exponent value = Represented exponent - Bias

Example: If we represent exponent as

01101 = 1+4+8 =13, the actual/true exponent value is 13 - 15 which is -2.

The Bias term enables the wide range for a particular representation format. After completion of the FP operations, the final output should also be in the proper floating point format. In order to represent the final result in proper format, the result is normalized and rounded to the nearest floating point number.

Special numbers

There are several theorems and special cases which needed special attention for floating point numbers [4]. Such cases are documented in great detail in the literature leaving no room for confusion. Even small changes in the calculation of the final value can have disastrous effects with respect to the correct result. The floating point representation of zero and others special numbers will be discussed below.

Signed Zero

According to IEEE 754 standard floating point format it provides signed zeros, that "+0" and "-0". The exponent, significand bits are zero. Computation of division

might require signed zeros. A divide operation performed with positive zero results in positive infinity and the same operation with negative zero results in negative infinity. Divide by zero operation will be handled by the exception handler.

Subnormal numbers

Any non-zero floating point number which is smaller than the smallest normalized floating point number is called a sub-normal or denormalized number. These numbers are called 'denormalized numbers' in IEEE 754 and 'sub-normal numbers' in IEEE 854. The hidden bit of the subnormal numbers is always '0'.

Infinite and NaN numbers

Both positive and negative infinities can be represented using IEEE 754 floating point standard. All exponent bits values will be '1' and all the mantissa bits will be zero. The sign bit determines the sign of such numbers. If the exponent bits values are '1' and all the mantissa bits are not equal to '0' then it represents Not a Number (NaN).

Rounding techniques

A final step of rounding is performed for all FP operations. Table 2-1 shows the various rounding techniques available in FP arithmetic. Round to nearest technique is the most commonly used rounding technique and was implemented in this thesis. There are certain scenarios where the above mentioned approach is not applicable. When this rounding rule is used for $x >$ maximum value of the given FP format, two assignments are possible. X will be assigned the maximum value if the x is $<$ maximum value + ULP

(maximum value)/2. Otherwise, it is assigned either positive or negative infinity based on the result of the sign bit. The possible values of x are

$$x = \begin{cases} Maximum\ Value\ for\ x < Maximum\ Value + ulp * (Maximum\ Value)/2 \\ Infinity\ for\ x > \ Maximum\ Value + ulp * (Maximum\ Value)/2 \end{cases}$$

where ULP is unit last place, if x > 0, then ulp(x) is the difference between x and the next larger floating point number. If x < 0, ulp(x) is the difference between x and the next smaller floating point number.

ROUNDING TECHNIQUE	Round(x)
Round down – Rounding towards negative infinity	$round(x) = \ x-$
Round up – Rounding towards positive infinity	$round(x) = x+$
Round towards zero	$round(x) = x-$ $round(x) = x+$
Round to nearest (even)	$round(x) = x + or\ x-$ The correct result depends on the value of $round(x)$ that is closer to x. If there is a tie, the one with zero as the least significand bit is chosen.

Table 2-1 : Table of various rounding modes in floating point arithmetic

Conclusion

The theoretical background provided this chapter should be helpful in understanding the computations involved in the ALU. The following chapters deal with the approach to satisfy the specification of the thesis.

CHAPTER THREE

FUSED MULTIPLY ADD UNIT

FMA is implemented on the PowerPC and Itanium processor families. FMA was first implemented in 1990 by IBM RISC/System 6000 [5] and has been widely used ever since. FMA is a floating-point operation which computes multiply-add operation with a single rounding. The equation of the FMA is given by $A * B + C$ where A is the multiplicand; B is the multiplier; C is the addend. This type of design is typically faster than a multiply operation followed by an addition when implemented in a microprocessor. The FMA dataflow is usually designed to have many pipeline stages of latency to get a throughput of one instruction per clock cycle. Thus the dataflow could effectively support two operations (multiplication and addition) per clock cycle. The dataflow consists of a multiplier, aligner, adder, leading zero counter, normalizing logic and rounding logic. With this instruction there is no need for a separate divide or square root hardware unit since they can both be implemented using the FMA in software. A fast FMA can improve the performance of many computations which involve the accumulation of products such as dot product, matrix multiplication, polynomial evaluation etc.

As per specification, the FMA4 instruction set was used in this thesis. FMA3 and FMA4 are two different variants of FMA used in industry. The difference between FMA3 and FMA4 is the number of operands used in the instruction. The 4-operand form (FMA4) allows A, B, C and D to be four different registers, while the 3-operand form (FMA3) requires that D is the same register as either A, B or C. The 3-operand form

makes the code shorter and the hardware implementation slightly simpler while the 4-operand form allows more programming flexibility.

Choice of FMA

The three major architectures of FMA are classic FMA, three path FMA and bridge FMA. The IBM RS/6000 Classic FMA architecture shown in Figure 3-1 was implemented with minor modifications. Lesser hardware and increased accuracy is achieved in FMA when compared with the separate FP multiply and add units

The three path FMA architecture is equally comparable with the Classic FMA. In broad terms, the architecture combines the FP adder/multiplier functionality and it has three paths after the multiplier tree, close path, far adder path and far product path. This results in the comparison of the operands in the RISC processor before the start of addition computation. The marginal improvement in the energy dissipation when compared with the classic FMA is offset by the extra hardware and area required for the comparators and the different paths.

The bridge FMA architecture reuses the existing floating point adder and multiplier units and adds the FMA functionality to FPADD and FPMULT instructions. The disadvantage is the addition of separate rounding units for the adder as well as the multiplier. There is at least a 50% increase in area and 20% increase in the power consumption of the bridge FMA when compared with the classic of FMA [6].

The RS 6000 architecture uses the multiplier and an aligner in parallel and a separate adder resulting in the pipelined usage of resources. The classic FMA also negates need for a comparator at the start of alignment operation because the addend is aligned to the

left of the product array. The right shift of the addend is alone enough instead of a bidirectional shift. Appendix B shows the block diagram of the other architectures of FMA.

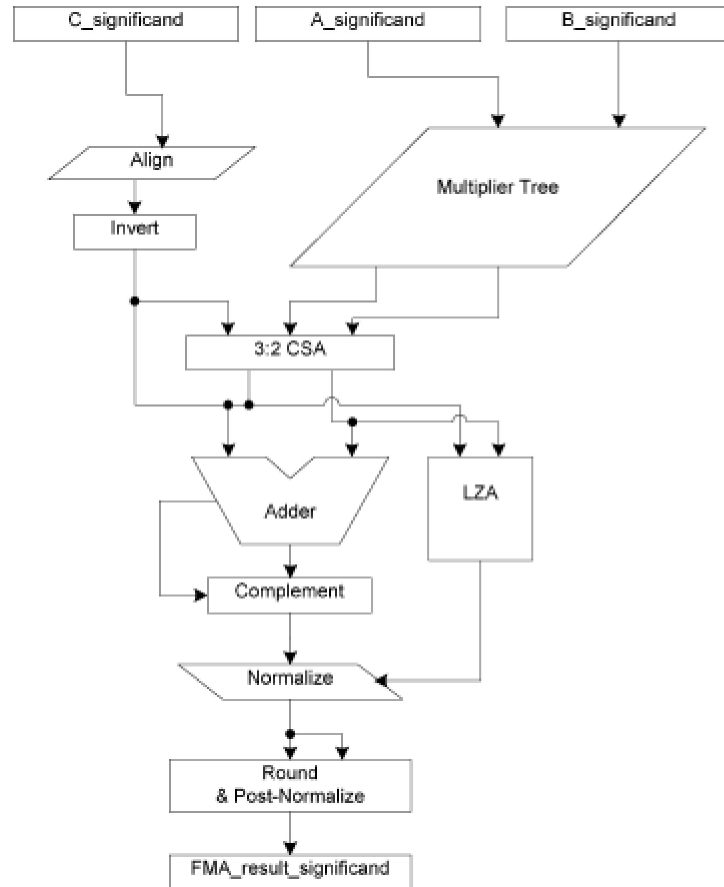


Figure 3-1: Architecture of classic FMA

Algorithm to perform FMA based computation

Let A, B & C be the input operands of the FMA. All the inputs are assumed to be normalized and signed floating point number. $(A * B) + C$ is the result of the FMA operation. Let us consider Ma, Mb and Mc to be the significands of A, B and C. Ea, Eb and Ec are the exponents of A, B and C.

Step1 (Multiplication and Right Shift)

Multiplication of the significands Ma and Mb results in '2m' bits of carry and sum vectors, where 'm' is the number of significand bits including the hidden bit. Addition of exponents of A and B is done by computing $Ea + Eb - B$ where B is the bias. Addition is performed by adding the exponents as represented and subtracting the bias for the corresponding floating point representation. The value of B is fifteen in the case of half precision floating point format.

Determination of the alignment shift amount of the addend is done in parallel to the multiplication. The resultant exponent $E_r = \text{Max}(Ea + Eb, Ec)$. This method results in bidirectional shift and might require both left and right shifter. In order to avoid the bidirectional shift, the addend is positioned 'm + 3' bits to the left of the multiplier product, and shifted only in right direction by the distance 'd'.

$$d = Ea + Eb - Ec + m + 3 - B$$

Shifting cannot be performed when the value of the shift amount is less than or equal to zero, and the maximum possible shift amount is '3m + 1'.

Step 2 (Addition and Count of Leading Zeros)

Addition of product and the shifted addend is done using two's complement addition. The result of the addition might result in certain number of leading zeros. The final result should always be represented in proper floating point format along with the hidden bit. Thus, the any left shift would also require the update of the exponents in parallel. The number of such zeros are counted with the help of a leading zero counter circuit.

Step 3 (Normalization and Rounding)

Normalization is performed with the help of a left shifter. After that rounding is performed based on the round-to-nearest technique, where it basically considers three extra bits guard (G), round(S) and sticky (S) after the least significant bit of the result. Another adder was used to implement the rounding logic. Any carry that is obtained from this addition might result in the update of the exponent.

Conclusion

FP operations might result in special cases like overflow and underflow which would always be checked irrespective of the cycle of operation. The choice of the FMA determines the algorithm that is used to implement the FMA operation. The theoretical idea provided in this chapter form the basis of understanding for the following chapters which deal with the implementation of the ALU in integrated circuits.

CHAPTER FOUR

COMPONENTS USED IN THE FMA

The components used in the FMA are discussed in this chapter. The choice of hardware and explanation of all the components gives a clear understanding of the actual operation of the FMA and FP Division which is handled in the forth coming chapters. Table 4-1 shows the various components used in the FMA and FP division.

COMPONENT	SIZE OF INPUTS	SIZE OF OUTPUT
Multiplier	$11\ bits\ X\ 11bits$	$22\ bits\ of\ sum\ and\ carry$
Right Shifter	$11\ bits$ (For alignment of addend)	$35\ bits\ of\ aligned\ data$
Left Shifter	$35\ bits$ (For normalization)	$35\ bits\ of\ aligned\ data$
Adder	$22\ bits$ (For addition of multiplier sum and carry)	$22\ bits\ of\ sum$
	$35\ bits$ (For addition of aligned addend and multiplier product)	$35\ bits\ of\ sum$
	$11\ bits$ (For rounding)	$11\ bits\ of\ sum$
Memory using D register	$11\ bits$ (For significand) $5\ bits$ (For exponent)	$11\ bits$ (For significand) $5\ bits$ (For exponent)

Table 4-1 : Table of various components used in FMA and FP Division

In addition to the above mentioned components, this chapter details the circuitry involved for computation of the exponent and the sign of the end result.

Memory using D register

A set of D registers were used to provide the inputs at the positive edge of the clock to the multiplier. The important point to be considered is the mode of operation of the ALU.

In the case of the FMA, the inputs from the D registers are accessed just once. Figure 4-1shows the trigger for the D register in the case of FMA. The 'SELECT_MEM_1' and 'SELECT_MEM_2' are used to trigger inputs operands A and B.

In the case of FP division, the same D registers are also used to store the output values of the multiplier and the normalized output. These are primarily intermediate results which are needed later for multiplication. This is explained in chapter seven which deals with FP division. A control logic was used to trigger the D registers at appropriate counter values to have multiple accesses for the same registers. In other words, the D registers were reused at corresponding clock cycles to provide inputs to the multiplier as shown in Figure 4-2. 'SELECT_MEM_1', 'SELECT_MEM_2' and 'SELECT_MEM_3' are the triggers for the inputs to the multiplier. The inputs to the multiplier are provided from operand B, look-up table, operand A, output of multiplier and the output of the adder.

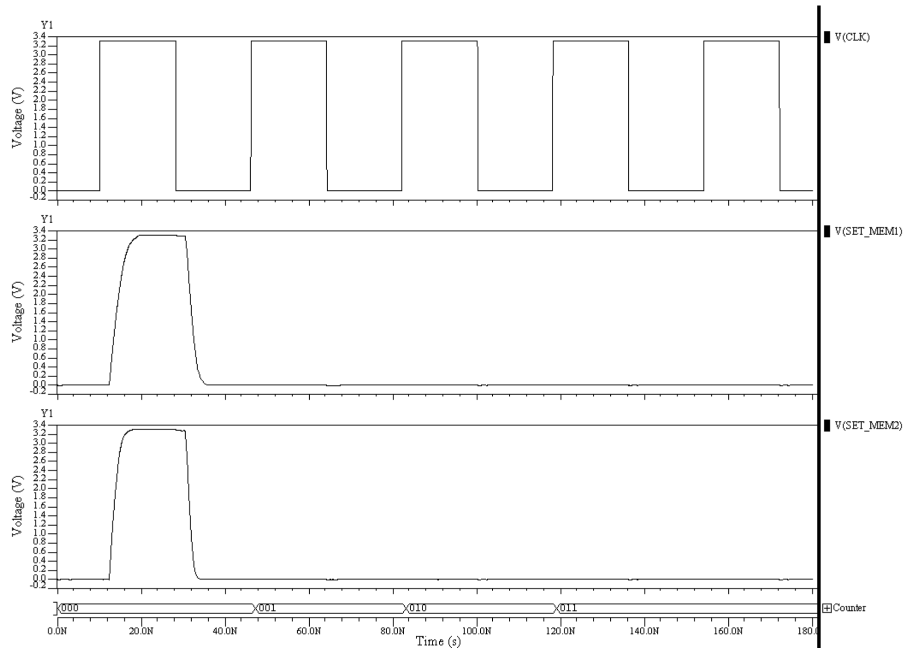


Figure 4-1: Trigger to D register for FMA mode of operation

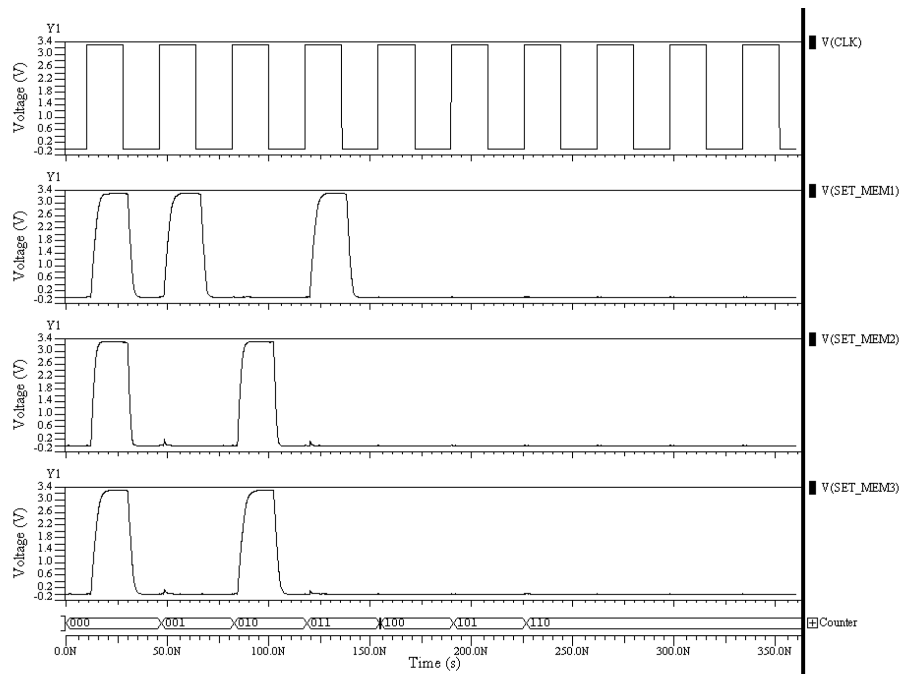


Figure 4-2 : Trigger to D register for FP Division

Multiplier

The multiplier is the most complex component of the ALU. The multiplier takes two 11 bit inputs and produces a 22 bit output in sum and carry format. The 11 bit input consists of a 10 bit significand and a hidden bit.

Implementation of multiplier

The partial products are produced in the first step using modified Booth's algorithm. The second step is the reduction of the partial products. Both Dadda and Wallace tree multipliers were extensively studied for tree reduction [7, 8]. The original idea was to implement reduced area multiplier in the ALU design [9]. In recent years, designers have used the 4:2 compressors to speed up the reduction of the multiplier tree. The steps associated with the multiplication are described in the following pages.

Step1

Radix 4 modified booth recoding was used for reducing the number of partial products. There are 6 partial products for an 11 bit significand. Figure 4-3 shows the inputs to the modified Booth Encoder. M1...M10 represent the 10 bit significand of the 16 bit FP number. The hidden bit is also accounted in the calculation of the partial product. Each set of three bits is used to assign a partial product as given in Table 4-2. The last partial product would always be positive because of the 0 beside the hidden bit. Figure 4-4 shows the schematic of the Modified Radix-4 Booth encoder.

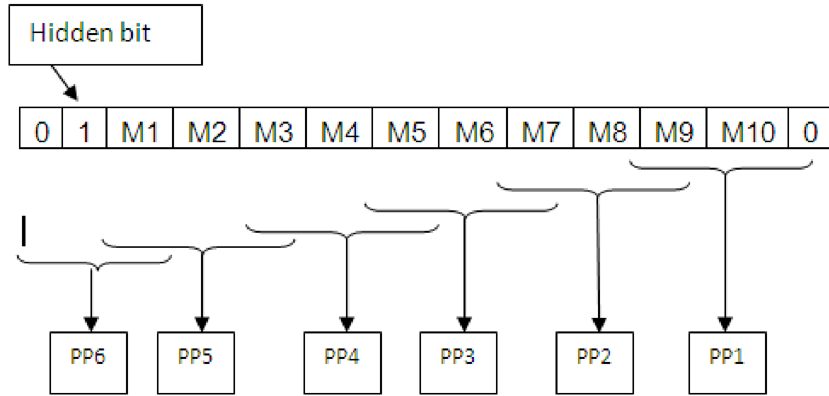


Figure 4-3 : Inputs to the Booth Encoder

RADIX -4 MODIFIED BOOTH ENCODING VALUES			
INPUTS			PARTIAL PRODUCT
X2I+1	X2I	X2I-1	PPI
0	0	0	0
0	0	1	Y
0	1	0	Y
0	1	1	2Y
1	0	0	-2Y
1	0	1	-Y
1	1	0	-Y
1	1	1	-0

Table 4-2: Table of modified Booth Encoder

Figure 4-4 : Schematic of the Modified Radix-4 Booth Encoder

Step 2

The bits are reduced after the partial product generation. This was done using 3:2compressors and 4:2 compressors. The single cycle multiplication was possible by

means of fast reductions using the reduction tree. The 3:2 compressors produced a two-third reduction in the height of the tree. Thus, the first stage of reduction produced four rows of bits. The 4:2 compressors reduced the height of second stage into two rows of sum and carry. In total, there are twenty 3:2 compressors and eighteen 4:2 compressors in the multiplier tree. This step is explained diagrammatically using the dot diagram shown in Figure 4-5. The logic levels of the multiplier and the aligner are comparable. An 11 bit input operand requires a 34 bit shifter in the FMA.

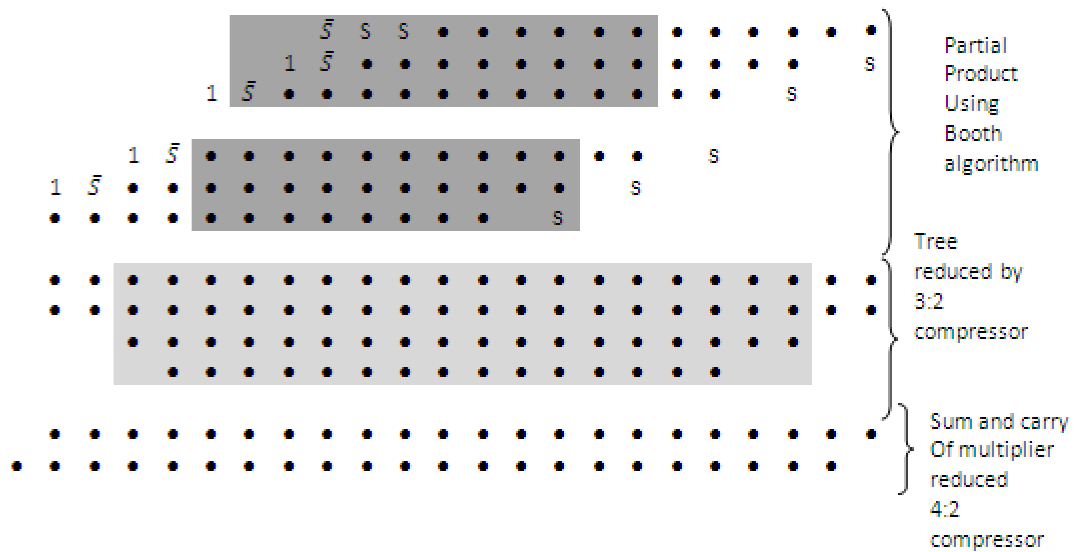


Figure 4-5 : Dot diagram of tree reduction of multiplier

Components designed for use in the multiplier

Pass gate multiplexer

The pass transistor based multiplexer shown in Figure 4-6 was designed for the 4:2 compressors. It is faster than the standard cell two inputs multiplexer [10]. One obvious feature is the parallel resistor combination. If we size $R_p = R_n = R$ then the path

resistance is merely R/2 which is lesser than the standard cell's resistance. Another obvious advantage is the presence of fewer transistors in the pass transistor multiplexer. The associated parasitic capacitances will be high in the standard cell library's' multiplexer.

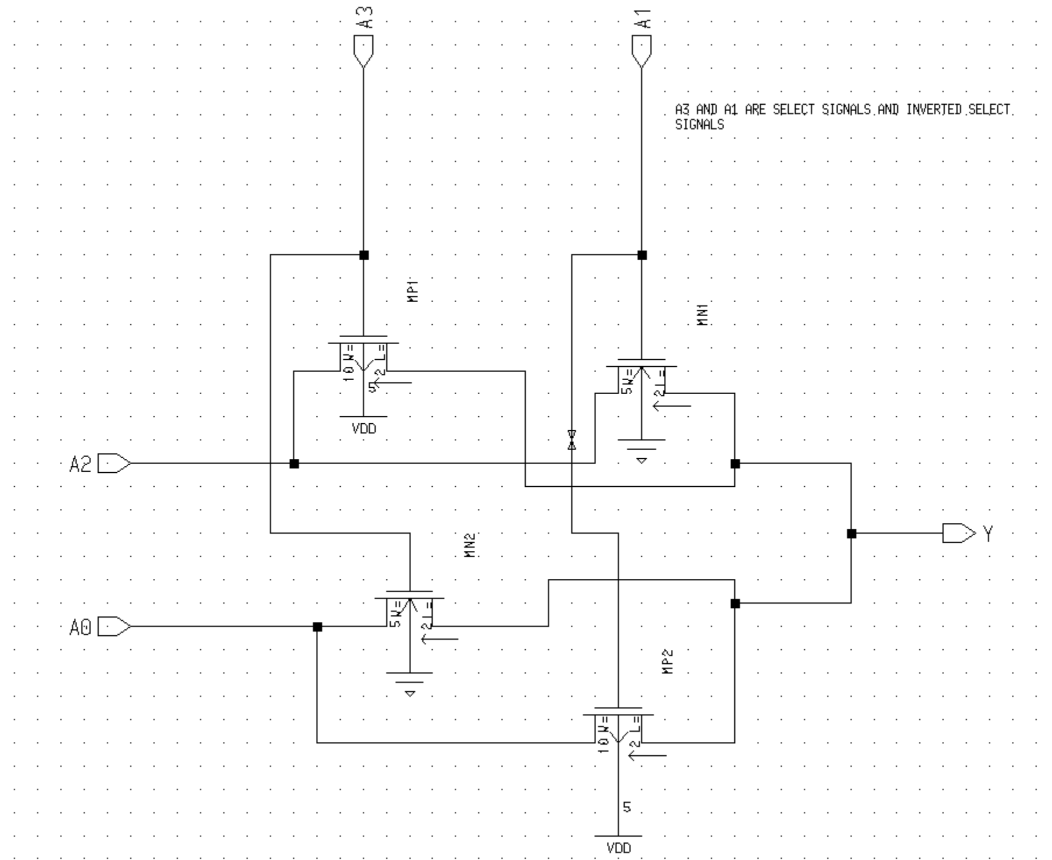


Figure 4-6 : Schematic of the pass transistor based multiplexer

4:2 Compressors

The 4:2 compressor was first introduced in 1981 and has been in use in most of the modern multipliers. The simplest 4:2 compressor consists of 2 full adder cells as shown in Figure 4-7. The most significand bit (MSB) and carry out (C_{out}) are of higher weight than least significand bit (LSB) in the 4:2 compressor. The important advantage of

the 4:2 compressors is that there is no ripple of carry. Thus bits of the same weight produce a carry out which is used in producing the LSB of the next 4:2 compressor. In other words, carry in (C_{in}) from previous stage does not propagate to the next 4:2 compressor.

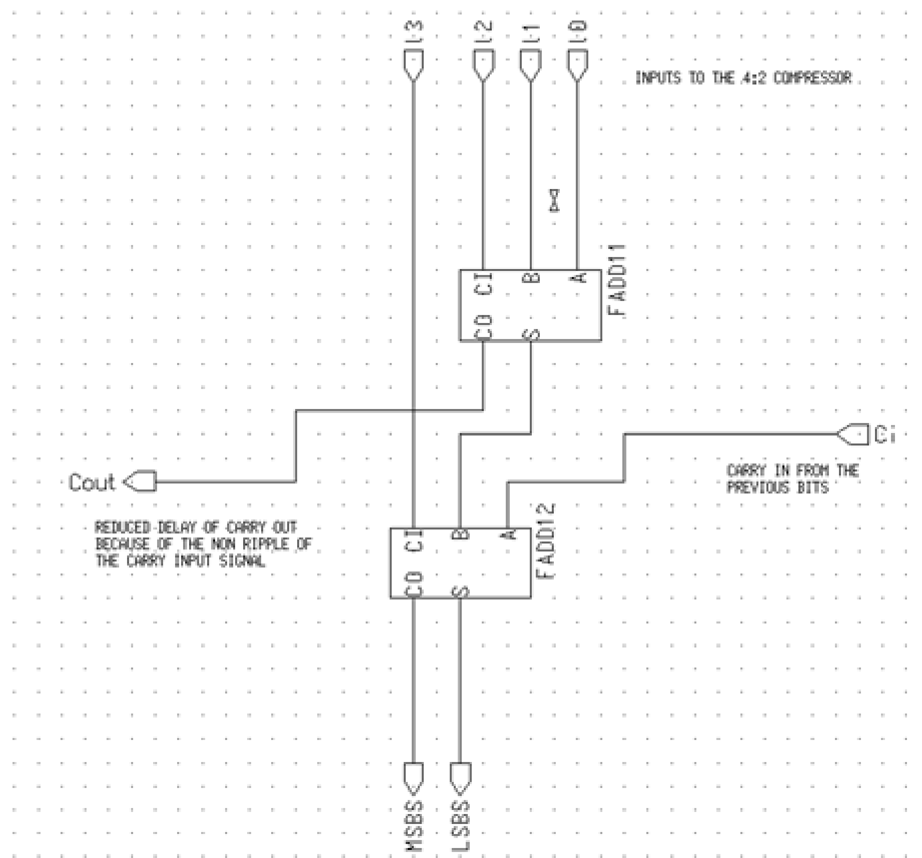


Figure 4-7 : Schematic of the full adder based 4:2 compressor

Figure 4-8 shows the schematic of pass transistor based 4:2 compressor. The symbol 'passgatemux' is the symbol of the schematic in Figure 4-6. l_3, l_2, l_1, l_0 are inputs to the 4:2 compressor and are of the same weight. LSBS, MSBS and Cout as the outputs

of 4:2 compressor. Comparison of the results between the full adder and pass transistor based multiplexer are given in chapter eight.

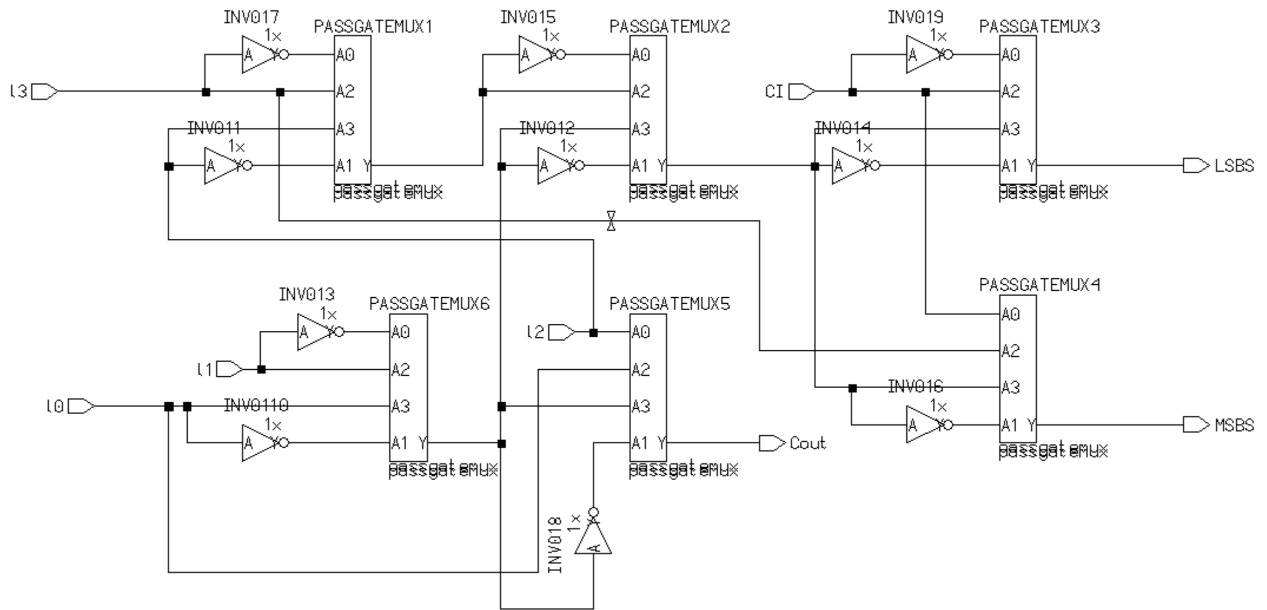


Figure 4-8 : Schematic of pass transistor based 4:2 compressor

Exponent Computation of multiplier

The exponent of the operands A and B are added to compute the exponent of the multiplier output. This is required for FP division which stores the intermediate values in the D register. The bias (B) is subtracted from the added result of the exponents. This is due to biased representation of the exponent.

Exponent of multiplier output = $E_A + E_B - B$

In the case of FP operations, the hidden bit is always considered to be one for normalized values. Division requires storage of the output of the multiplier in the normalized format in the memory. Consider the following output of the multiplier

Output = 11.f, where f consists of twenty bits of data.

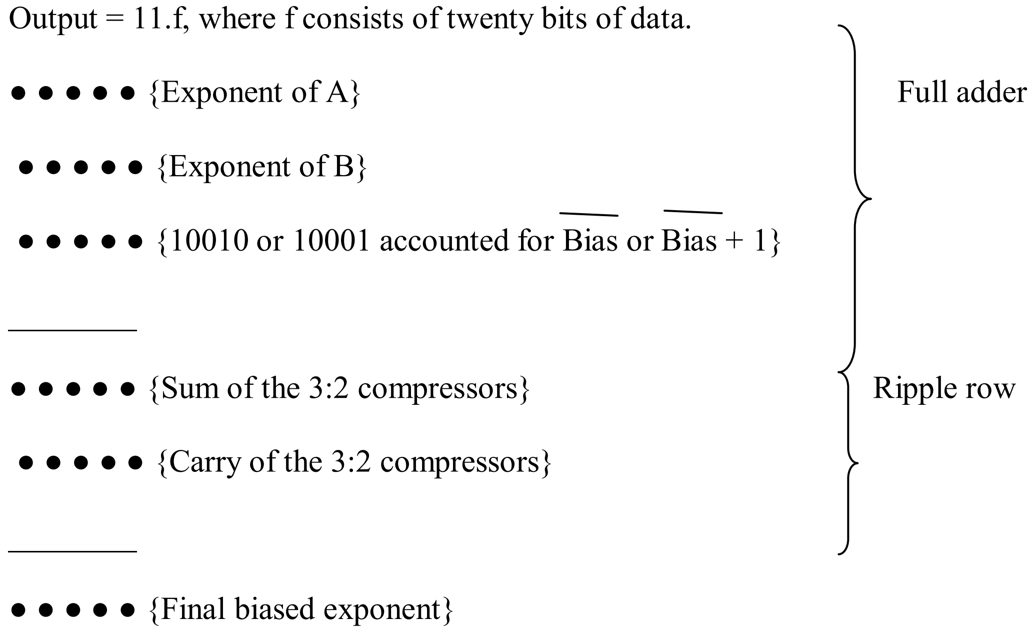


Figure 4-9 : Dot diagram of the exponent computation

When such values are stored in the memory, the exponent part of the multiplier output has to be incremented by one to account for the normalized multiplier output. The above correction was designed using a tree of carry save adders. Figure 4-9 shows the dot diagram of the exponent computation. The MSB of the product of multiplier had to be checked and the corresponding addition of the exponent was performed by a tree of 3:2 compressors as shown in Figure 4-10.

Figure 4-10 : Schematic of the exponent calculation of multiplier

Instead of waiting for the output of the twenty two bit adder, the sum and the carry of the multiplier was used to pre-compute whether there is '1' at the MSB of the adder output as shown in Figure 4-11. This ensured that the correction of the exponent was done in parallel to the twenty two bit addition of the multiplier sum and carry.

ANTICIPATORY LOGIC TO DETERMINE THE PRESENCE OF '1' IN THE MSB OF MULTIPLIER

SUM AND CARRY FROM THE MULTIPLIER ARE USED TO ANTICIPATE '1' IN MSB

Figure 4-11 : Schematic of anticipatory logic for exponent

Thirty four bit right shifter

The right shifter is required to align the addend with respect to the output of the multiplier. The very first step is to determine the amount of shift that is required [11]. Based on the difference between the exponents of A, B and C, there is either a left or a right shift of the addend Mc to avoid a bidirectional shift, the addend is placed m + 2 bits to the left of the product of the multiplier where 'm' is the number of significand bits including the hidden bit.

Addend(11 bits)	00	Multiplier (22 bits)

The shift distance is given by d = $E_A + E_B - B$ + m + 3- E_C. The maximum right shift is 3m +1. The multiplexer from the standard cell library was used to implement the thirty four bit right shifter as shown in Figure 4-12. The 3:2compressors available in the standard cell library were used to compute the shift distance shown in Figure 4-13. The addend was shifted based on the shift amount. It should be remembered that no shift is performed when the addend very large compared to the result of the multiplier.

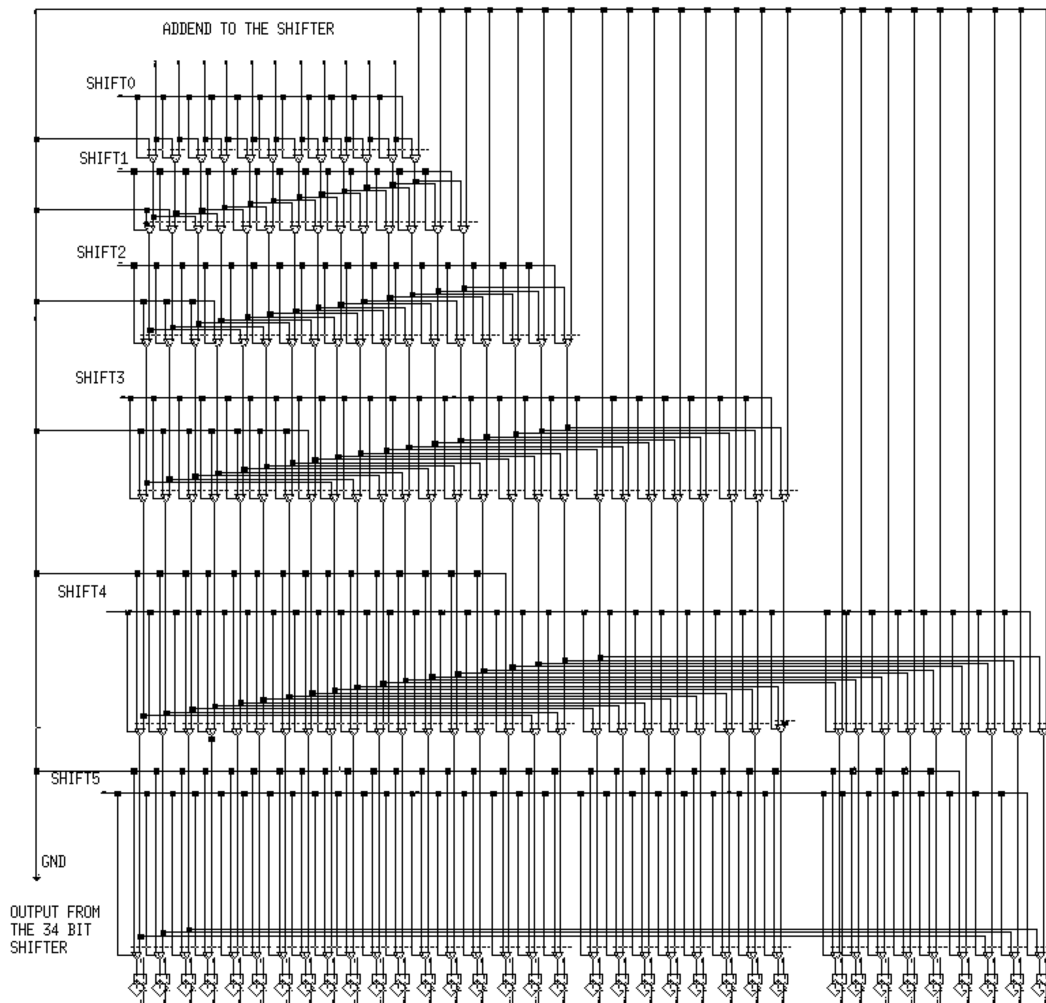


Figure 4-12 : Schematic of 34 bit right shifter

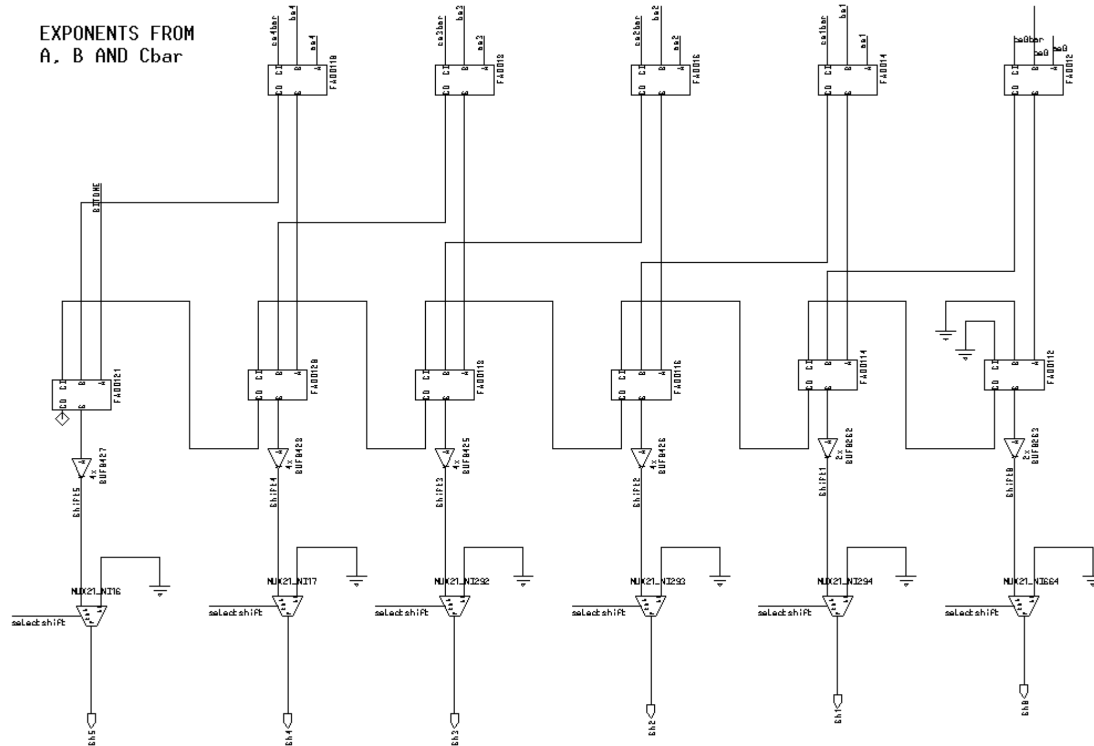


Figure 4-13 : Schematic to compute the shift distance

Addition of multiplier sum and carry

The multiplier output format from the reduction tree consists of two rows representing sum and carry. Usually in a FMA implementation, the sum and carry rows will not be combined immediately. They will be combined using the reduction tree along with the aligned addend. The reduction of latencies is an important criterion for FP division. FP division operations require the final multiplier result to be stored in the register for further calculations. An extra addition of sum and carry was performed using a 22-bit adder to eliminate the separate rows of memory for storing the sum and carry.

The resultant output was stored in the register. The size of the memory is reduced from six rows to three rows as a result of this addition.

In an alternate approach, the ALU could have been designed using a row of 3:2 full adders in carry save format to add the aligned addend, multiplier sum and carry. The output is then normalized and rounded. This is the traditional approach of the classic FMA/IBM RS 6000 architecture. This thesis represents a modification of existing architecture for a more application specific design.

The adder is the most critical component in a dedicated FP adder and thus designing fast adders would have been very important [12, 13]. Since, the multiplier is the most area and power intensive component of the FMA, the 35 bit adder was designed for moderate performance when compared to the multiplier.

Types of adders

The FMA operation was performed with the help of two successive additions by means of a 22 bit adder and a 35 bit adder. Various configurations of adders were analyzed and the choice was made based on the latency requirement and regularity in implementation.

The designers use carry look ahead and carry skip adder configurations for design of larger adders. The carry select adder was rejected because of the excessive area and necessity of pre-computed values [14]. Implementation of this adder in layout results in an area intensive and power hungry circuit. The carry look ahead addition is a very widely used concept of addition and it causes a logarithmic reduction in delay by looking ahead at the carry [15]. All the adder designs focus on the generation of the carry which

is the most critical signal of the adder. The fundamental equation for the generation of carry is given by

$G_{i:0} = G_i + P_i * G_{i-1:0}$, where G_i is the generate signal, P_i is the propagate signal, $G_{i-1:0}$ is the carry from the previous bits.

The Kogge Stone adders present a fanout of 2 and a reduced delay of $\log_2 N$ where N is the number of bits involved in the addition. The disadvantages of Kogge Stone adder are the long wires and the power consumption [16]. The delay of the Kogge Stone adders is given by

$T_{tree} = T_{pg} + [\log_2 N].T_{AO} + T_{xor}$, where

T_{pg} is the time delay for the generation of generate and propagate signals,

T_{AO} is the time delay for the generation of signals from the group PG cells and

T_{xor} is the time delay for the generation of the sum.

The Brent Kung adders present almost twice the number of logic levels as that of the Kogge Stone adders but it has been found from previous projects that Brent Kung adders offer reduced power dissipation when compared with other tree adders. This has been attributed to the reduced gate count and reduced length of the wires. There is also a regularity of layout which reduces area significantly. This is very significant in the case of long adders which are needed for 64 bit and 128 bit adders because of the reduced wiring capacitance [17]. The delay of such an adder is given by

$T_{tree} = T_{pg} + [2\log_2 N - 1].T_{AO} + T_{xor}$

The Han Carlson adder offers moderate wiring capacitance with respect to the Kogge Stone adder and moderate number of logic levels when compared with the Brent

Kung type of adders [18]. This type of adder is a combination of advantages of both Brent Kung and Kogge Stone adders with the same fanout. The delay of such an adder is given by

$T_{tree} = T_{pg} + [\log_2 N + 1].T_{AO} + T_{xor}$

Various tree adders have been proposed in the literature and present significant advantages [19, 20, and 21]. As already mentioned, the best architecture is chosen based on the specific application and the size of the design. Usually, long additions in terms of the number of bits are performed using the tree/parallel adders. Significant improvements can be obtained by using dynamic style instead of static CMOS logic or multiplexer based adder [20, 22].

The general practice is to use a carry skip configuration for 8-16 bit lengths of the addition operands. For larger operands like 32, 64 and 128 bit lengths, the tree adders are used. The last few years have been dominated by designs requiring a balance in delay as well as power dissipation. The variable carry skip adder offers the advantage of improvements in speed and optimal delay in the carry and ripple paths. [23]. The literature is rich with information regarding the optimal adder designs based on the energy delay product and the options that confront the designer before the appropriate design is chosen [24].

Reasons for Carry Skip Adder

Various adder configurations were analyzed from the literature before the design of the half precision ALU [25]. The choice was made on the regularity of layout, energy delay of the adder, criticality of the adder and the operand size. The first cycle of

operation involves a single 22 bit adder and carry look ahead and carry skip adder configurations are very comparable for smaller operand size. A fast adder for 22 bit addition is critical since the delay caused by the multiplier is large during the first clock cycle. To tackle such design issues, a fast pass transistor based multiplier was used in this thesis. The pass transistor based multiplexer is the fastest multiplexer and has been widely used by designers for high speed computations. The variable skip adder configuration was used to design the 22 bit adder. The comparable delay as well as regular layout with respect to the parallel prefix adder was the reason to choose the 22 bit variable carry skip adder architecture [26].

Algorithm for design of Variable carry skip adder

The original carry skip adder had equally sized blocks. Further enhancements in design in the form of variable block sizes ensured that the delay is comparable to the parallel prefix adders [27].

The algorithm to design such adders is as follows

Assume 'T' to be the time delay for the carry signal to skip over the block/group of bits.

Step 1

Calculate the value according to the following formula corresponding to each group numbered from 1, 2…

$Value = i + \frac{iT}{2} + \frac{i^2T}{4} + (1 - \frac{(-1)^iT}{8})$, where T is the Time delay and i is the Group number

Step 2

Choose the number M such that

$$N < 2\sum_{i=1}^{[M-1]/2}\{Value\} + \frac{1-(-1)^M}{2}\left\{\left[\frac{M}{2}\right] + .5*\left[\frac{M}{2}\right]T + .25*\left[\frac{M}{2}\right]^2 + (1-(-1)^M)\frac{T}{8},\right.$$

where N is the total number of bits in the adder and M is the optimal number of blocks The adder is now designed using the optimum number of blocks and the number of bits in each block.

<u>Design of 22 bit adder</u>

The design of a fast 22 bit adder is very important because of the importance of first cycle which determines the frequency of operation of the three clock cycle FMA operation. Frequency of operation is obtained by accounting for the maximum time delay of a hardware component.

As mentioned earlier, the first step is the calculation of value of various groups of i. The optimal value of T has been placed between 2 and 7. In actual practice, T is a variable that is dependent on the size of the blocks. It was found that the variation in T is very minimal for various sizes and so T was assumed constant.

Since, a very fast adder is required; T was kept at a minimum value of 2 to account for the least time delay. The corresponding value for various groups of i are shown in Table 4-3.

I(GROUP NUMBER)	VALUE(NUMBER OF BITS PER GROUP)
1	3
2	6
3	11

Table 4-3 : Table of number of bits per group for various values of group number

The number of blocks $(M)$was chosen such that it conforms to formula in the algorithm. In this case, $N \leq 2\{3+6\}+\ 11$, where N = number of bits of adder = 22. The distribution of the variable block is 3, 6, 11, 6, and 3. The extra 7 bits in the middle group can be removed to form the following sizes of 3, 6, 4, 6 and 3. The middle group has to be the largest size for the maximum skip of the carry signal. Thus middle three groups are resized to have an overall size of 3, 5, 6, 5 and 3. The delay of such an adder was found to be 400ps slower than the similar sized carry look ahead adder. The area of the CSK was $51.8 * 10^{-9}m^2$ and is 50% lesser than the Carry look ahead adder. Thus an adder with comparable delay to that of the parallel prefix adder was designed to meet the goals of the thesis. Figure 4-14 a single carry skip block having a group size of three. The twenty two bit carry skip adder is built using such blocks with variable group sizes.

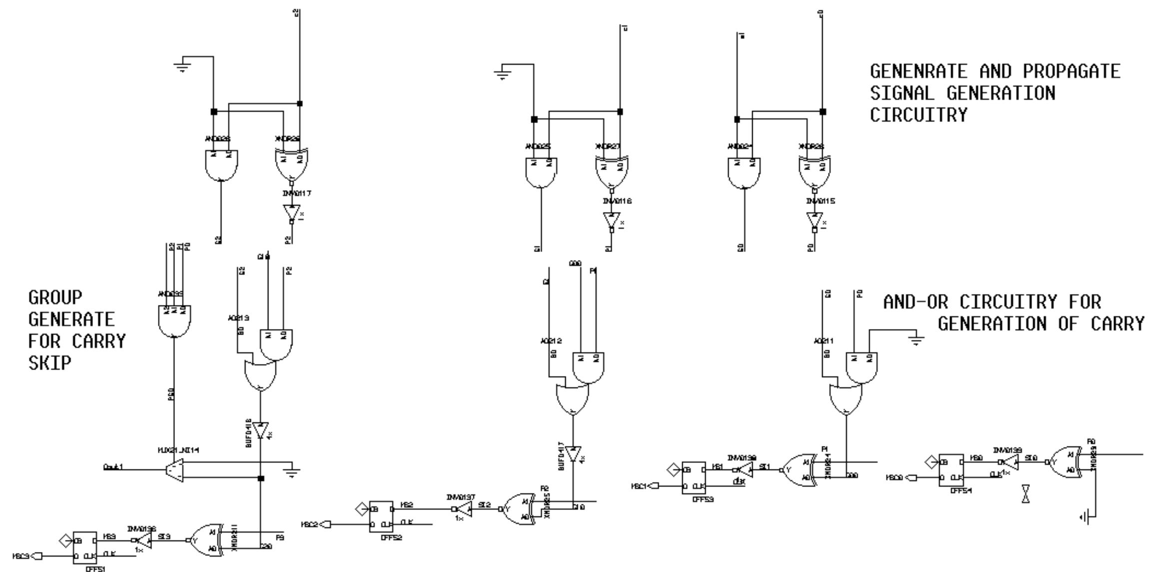


Figure 4-14 : Schematic of a single carry skip block

Thirty five bit adder

The adder was designed by using gated clocks which trigger the inputs to the adder only for the appropriate clock cycles. Since there are two additions in the case of two's complement subtraction, there is a separate local clock for the addition. Clock gating would reduce the energy dissipation because the adder is activated only for corresponding clock cycles. More information about clock gating is provided in chapter five. This was achieved by feeding back the output of the first addition as one of the inputs to the adder.

Inputs to the adder

There are actually three cases of additions that have to be enforced. Table 4-4 shows the possible operations using the adder. As mentioned earlier, A , B and C are the input operands of the FMA.

Two additions are necessary to calculate the final sum using two's complement subtraction. An adder clock is a local clock that has been used to compute two additions. The first addition uses values from the aligned addend and multiplier.

OPERATION	SIGN OF RESULT	ACTION PERFORMED
$A\,X\,B\,+\,C$	Positive	This is an effective addition. All inputs are fed directly to the adder.
$A\,X\,B - C$ or $(-A\,X\,B)\,+\,C$ or $(AX-B)\,+\,C$	Positive or Negative	This is an effective subtraction. The negative operands are inverted and fed as inputs to the adder
$(-A\,X\,B) - C$ or $(A\,X - B)\,-\,C$	Negative	This is an effective addition. All inputs are fed directly to the adder.

Table 4-4 : Table of all possible operations in the adder

A control circuitry using the sign bits was used to determine whether the addition is an effective addition/subtraction. In the case of effective addition, the signals from the multiplier output and the aligned addend are fed directly to the 35 bit adder. In the case of

effective subtraction, the negative input operand is inverted and fed as an input to the adder. An XOR gate is used to find out the effective sign of $A\,X\,B$. In case, the effective sign of $A\,X\,B$ is negative, the output of the multiplier is inverted. The operand C is inverted if it is the negative operand in the effective subtraction.

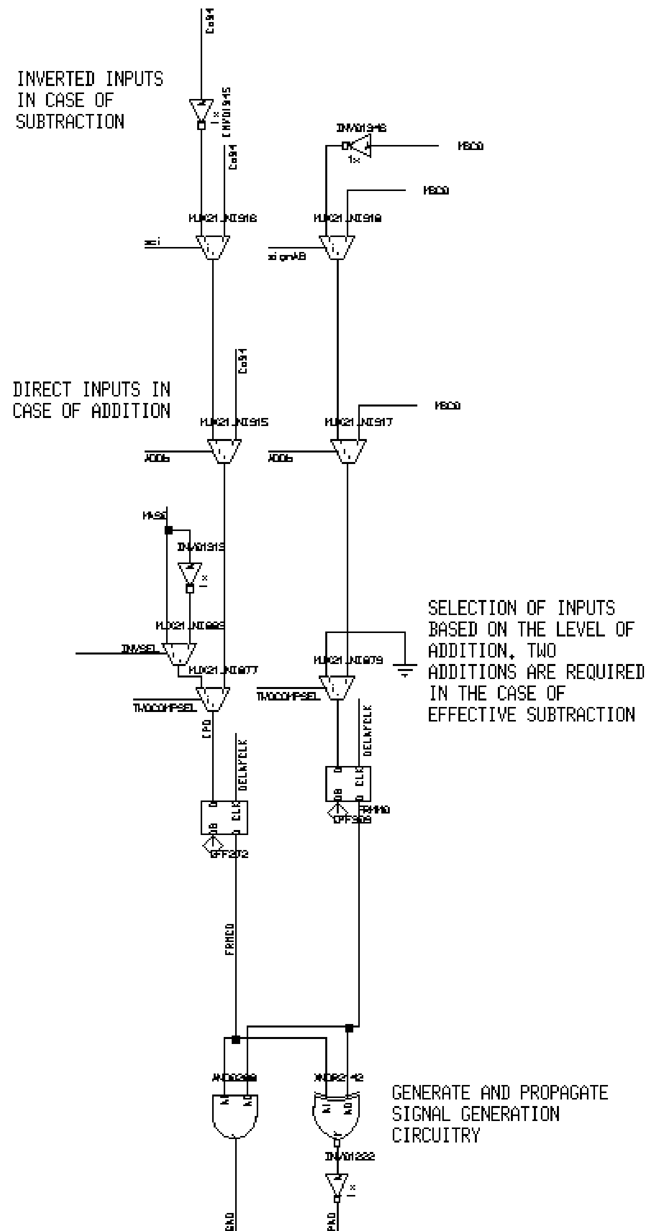


Figure 4-15 : Schematic of inputs to the adder

The output from the first addition is fed back as input for the second addition. Originally end around carry adder was thought of as a design option but the sign of the result is easily detected using 2's complement addition. Figure 4-15 shows the selection of inputs for a single bit of addition. This is replicated for other bits also.

Reasons for carry skip adder

There is a need to perform a second addition to incorporate the FP SUB instruction in the ALU. The literature states that carry look ahead adders are used widely for high speed additions for operand size greater than or equal to 32 bits. The variable carry skip adder for the 35 bit operation was used because of the reduced importance of this addition in the latency of the FMA operation. Since the multiplier delay determines the latency of operation of the FMA, a fast adder was not really critical in the second cycle. Regular implementation is one of the most important advantages of carry skip adder. An alternate approach would have been a two cycle multiplication which might have resulted in the design of a very fast parallel prefix adder for the 35 bit addition. The multiplication is performed within in a single clock cycle and thus necessity of a successive fast addition was negated.

Design of 35 bit adder

The 35 bit adder is designed in the same manner as the 22 bit adder. The time delay is the time required for the carry signal to propagate through a block of bits. The time delay of the carry signal for a single block (T) was kept at a minimum value of 2. The literature states that ' T' could be kept at minimum and could be a constant

irrespective of the size of the adder. The following table shows the values of the number of bits with respect to the group numbers.

I(GROUP NUMBER)	VALUE(NUMBER OF BITS PER GROUP)
1	3
2	6
3	11
4	16

Table 4-5 : Table of number of bits per group for various values of group number)

The value of number of blocks (M) is chosen such that $N \leq 2\{3+6+11\}+16$, where N is the number of bits = 35. The distribution of the group sizes was 3, 6, 11, 16,11,6 and 3.The 11 bits in the middle was removed to form a group distribution of 3, 6, 11, 7,11,6 and 3. The middle group was rearranged to form a group distribution of 3, 5, 6,7,6,5 and 3. A single block is similar to Figure 4-14 and similar blocks of varying sizes were implemented for the entire adder.

Leading zero counter

The leading zero counter (LZC) counts the number of zeros in the result of 35 bit adder. The result of LZC indicates the amount of left shift for normalization. The literature states that the leading one bit had to be predicted in parallel with the computation of the large adder. This prediction is usually done with the help of leading zero anticipatory circuit (LZA) or a leading zero predictor (LZP). The algorithmic approaches to solve such design problem have been proposed in the literature [28]. The original approach was to include a leading one predictor in parallel with the 35 bit

addition and then use the result to calculate the count of the leading zeros [29]. There are specific speed ups that could be achieved for special architectures of FMA [30].

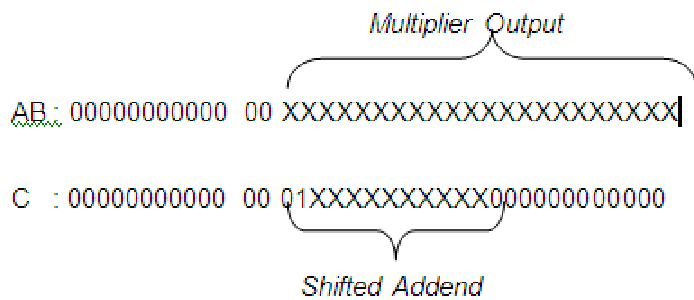
A careful look at the circuitry and the latency trend shows that the FMA operation doesn't require a LZA unit. The FMA operation includes multiplication and shifting in its first clock cycle and then addition in the second clock cycle. The delay of the actual addition and the LZA was almost comparable by looking at the number of logic levels involved in both operations. In the case of the LZA, there might be a need to perform a single bit shift whenever there is an anomaly from the desired formatted result.

The benefits were minimal for operands of size of 16 bits and therefore a separate LZP or LZA was discarded. Also, addition is not the critical component that determines the latency of the FMA operation in this design. This provided an affordability to compute the count of leading zeros without prediction of the leading ones. The leading zero count was performed once the result of the adder was fed as input to the D register clocked with a separate clock derived from the local clock. This made sure that the signals that arrived at the input of the LZC were stable and time invariant after the maximum adder delay.

Implementation of LZC for FP ADD instruction

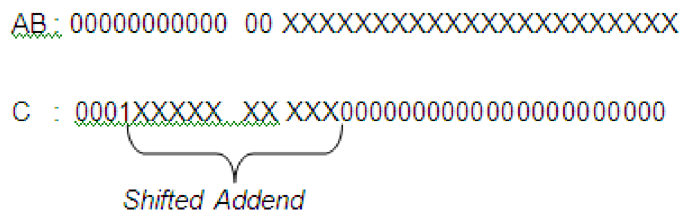
The thesis also explored the concept of determining the leading zeros using a pattern of inputs to the 35 bit adder. This worked very well for FP addition and could have been used as part of low power design of FMA. The number of leading zeros for an FP ADD instruction is fixed depending on the amount of shift for the addend. If the shift amount is less than 12, the count of leading zeros was shift value – 1. For the other case,

the count of leading zeros was 12 or 13 or 14. Another advantage of this design is that it involves a smaller left shifter when compared with the generic 35 bit left shifter. Before the final step of rounding, a check was performed to check if the MSB of the shifted value was 0. In that case, a single left shift was again performed to have the hidden bit as the MSB of the computed result. A separate logic block was designed to compute the leading zeros and then this was fed as input to the left shifter for normalization. Figure 4-16 and Table 4-6 shows the number of leading zeros for the above mentioned cases.



Final addition would involve a maximum of 12 or 13 or 14 leading zeros.

Consider the following pattern of inputs to the adder



The final addition would involve a maximum of 2 or 3 leading zeros.

Figure 4-16 : Example of input pattern to the 35 bit adder

NUMBER OF RIGHT SHIFTED BITS	PREDICTED ZERO COUNTER
≥ 13	12 or 13 or 14
<13	Number of right shifted bits - 1

Table 4-6 : Shift and predicted leading zero counter

The advantage of the above logic was that it need not have a separate design of conventional LZA or LZC. The disadvantage is that this could be implemented in the case of an effective addition and not in the case of an effective subtraction. In other words, there is a need to compute the count of leading zeros in the case of FP SUB instruction. The leading one bit could appear anywhere among the 35 bits of adder output in the case of effective subtraction. FP subtraction is also a critical component of the thesis and a non generic design which is very specific to the instruction was discarded from the overall design. The complexity of FP SUB also introduced an extra latency in the overall FMA. Thus a generic design which involved a LZC was used.

Algorithm for leading zero counters

Various architectures for the LZC were analyzed before the actual implementation [31].LZC used in this thesis is based on an algorithm proposed by Oklobdzija. The algorithm consists of the following steps

Step 1

The entire n bit output is broken into groups of adjacent 2 bit groups

Step 2

The count of leading zeros was determined for each group and this is combined with the adjoining group. This is expanded to all the groups and the entire operation involves a delay consisting of five logic levels.

Step 3

The OR of all the bits was generated to see if all the bits were zero. In such cases, the zero flag was raised.

Implementation of LZC

A problem was encountered during the implementation of the above algorithm. The above algorithm worked for bits of the magnitude of powers of 2. In other words, the above algorithm worked for 2, 4,6,8,16,32 … The adder output resulted in a 35 bit output which was an odd number and adjacent groups of bits could not be expanded for the entire 35 bit data.

The solution to this problem was the design of a separate 3 bit LZC along with the 32 bit LZC as illustrated in Figure 4-17. Figure 4-18 shows the schematic of the 32 bit LZC. The count of leading zeros for 32 bit data and the remaining 3 bits were computed in parallel. A check was performed to see whether these two values were to be added and fed as input to the left shifter. Figure 4-19 shows the schematic of the 3 bit LZC. The LZC from the 3 bit data is used if the count of leading zeros is greater than 32 and this case is possible only in the case of an effective subtraction. If all the bits are zero, the zero flag is raised.

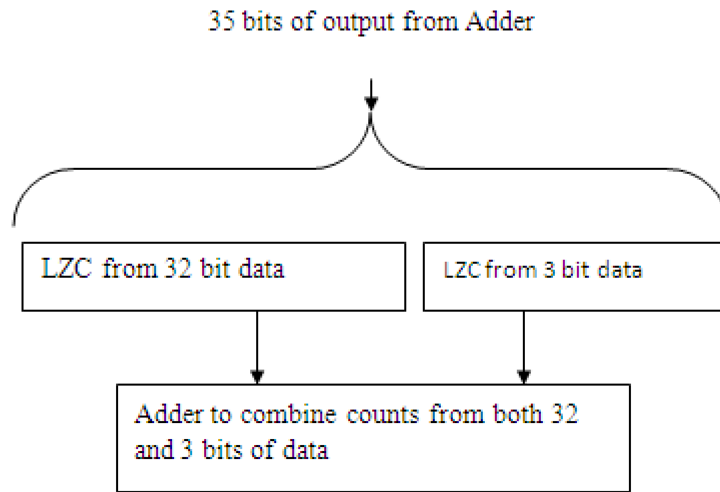


Figure 4-17 : Block diagram of Leading Zero Counter for entire 35 bits of data

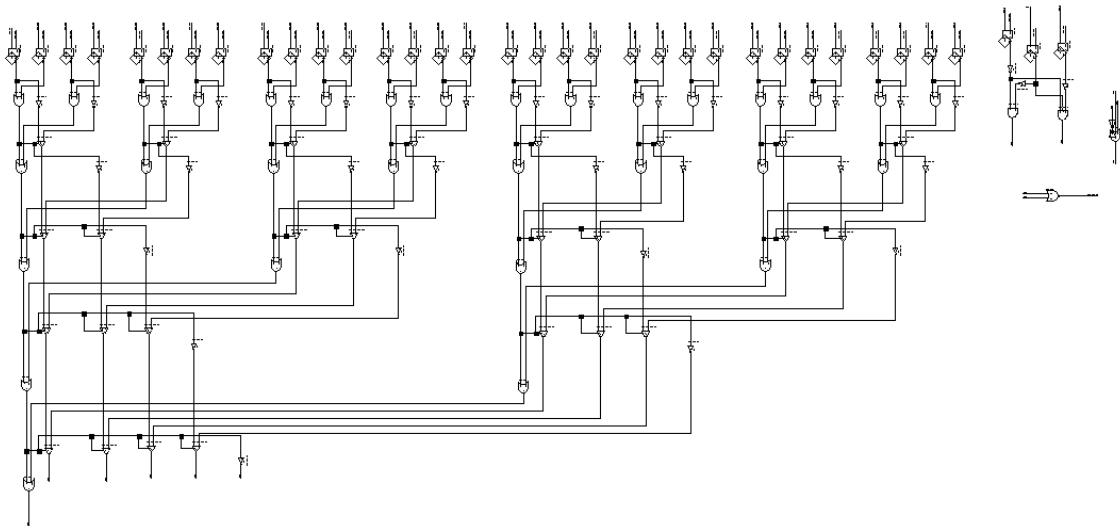


Figure 4-18 : Schematic of 35 bit leading zero counter

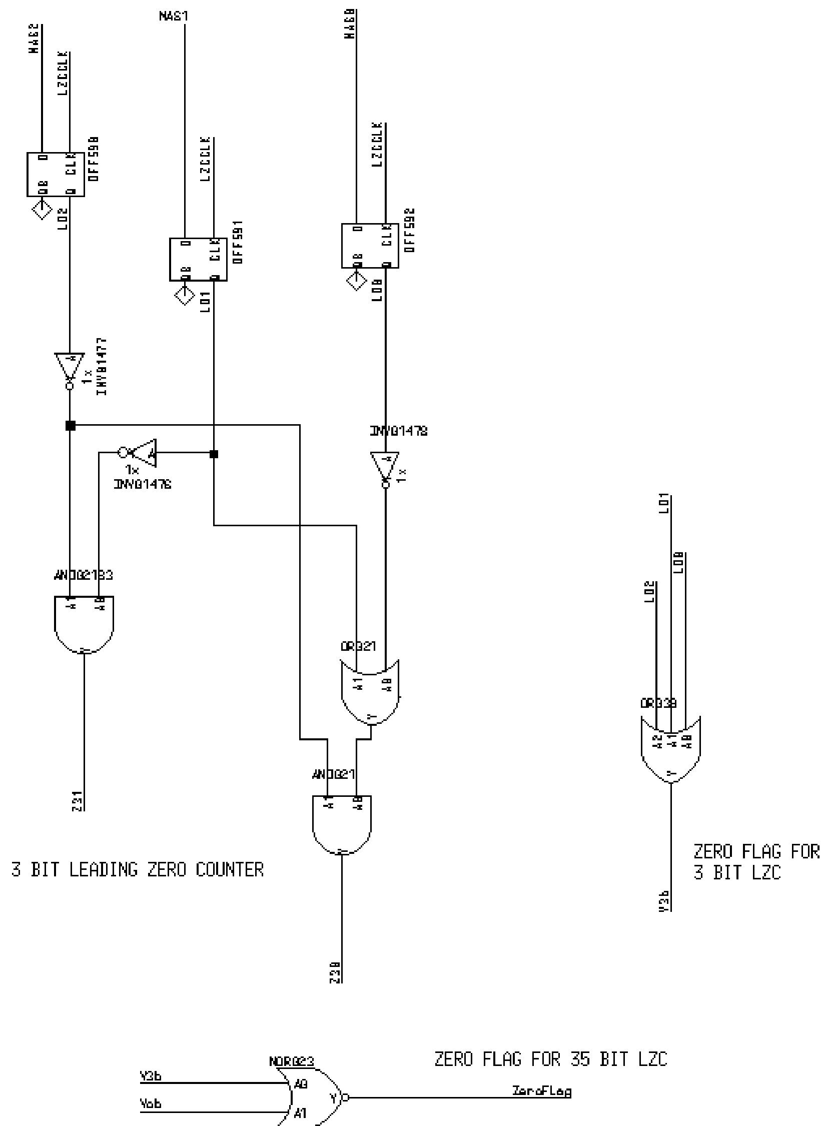


Figure 4-19 : Schematic of 3 bit LZC and zero flag

The implementation of the 3 bit LZC is discussed in the next two pages. Let us assume a set of three bits of data as $LD2\ LD1\ LD0$ in binary format. The logic equation used to compute the three bit leading zero counter is given below.

$$Lead\ Zero\ Count\ LSB\ = \overline{LD2}\,(LD1 + \overline{LD0})$$

$$Lead\ Zero\ Count\ MSB = \overline{LD2}\,\overline{LD1}$$

The following truth table was solved using Karnaugh map.

INPUT COMBINATION	MSB	LSB
000	1	1
001	1	0
010	0	1
011	0	1
100	0	0
101	0	0
110	0	0
111	0	0

Table 4-7 : Truth table of 3 bit counter

In addition to the above logic, all the bits were also fed to an OR gate. In the case of all the bits being zero, the zero flag signal for the three bits is placed at logic high.

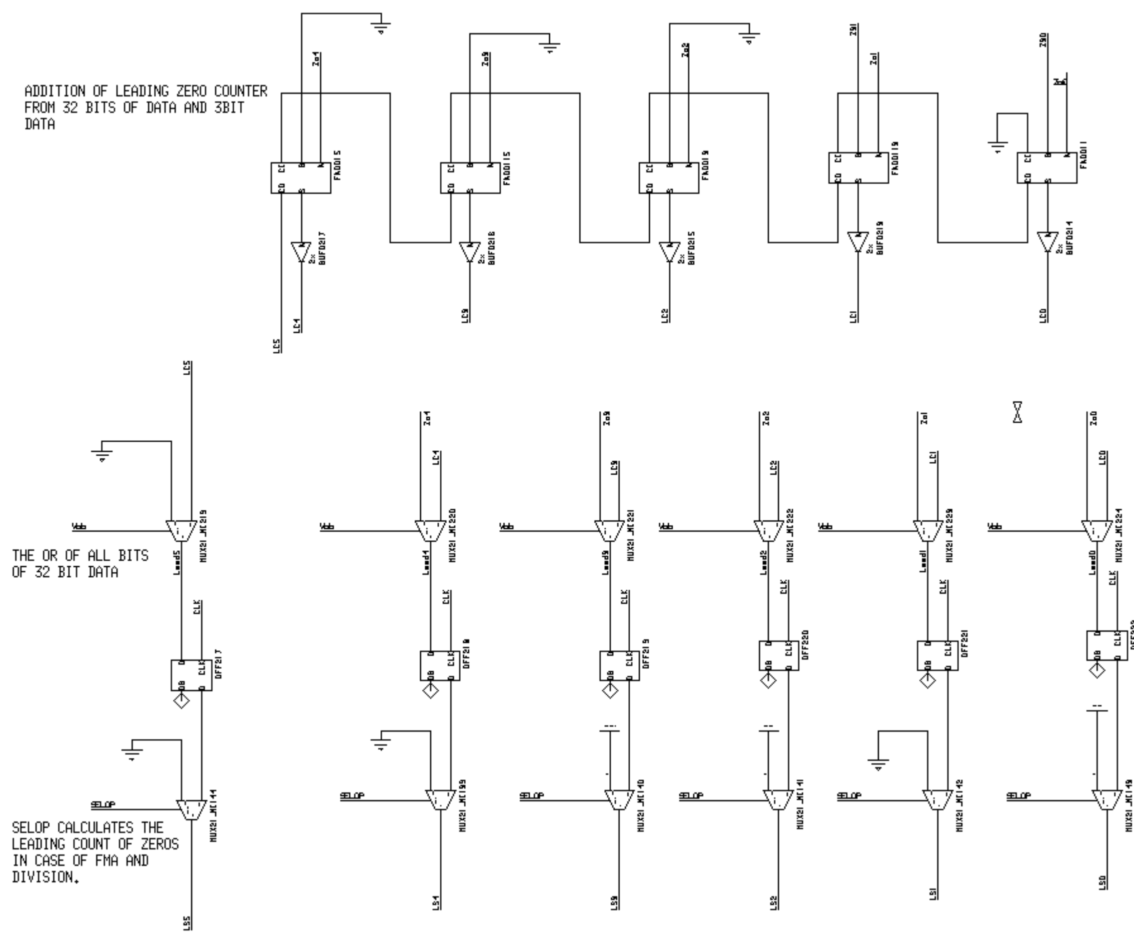


Figure 4-20 : Schematic of the adder to count the leading zeros

The results of 3-bit LZC are ignored if there is a leading ‘one’ encountered within the group of 32-bits. Otherwise, the result of the 3 bit LZC are added to the result of 32 bit LZC using 3:2 compressors as shown in Figure 4-20.

Implementation of normalization

Normalization consists of a 34 bit left shifter. The original plan was to complete normalization using the right shifter that was used earlier. Yet, the complexities involved in the selection of inputs proved to be difficult in the integration of both right and left

shifts in the same hardware component. Another problem was the necessity of the shifted addend to remain stable during the clock cycle. A separate left shifter was necessary in FP division because there is a possibility that a single clock cycle might have both right and left shifts at the same time. The thirty four bit left shifter computes the normalized output based on the count of leading zeros that was calculated using the LZC. The normalization circuit was implemented using the multiplexer from the standard cell library as shown in Figure 4-21.

Exponent computation before Rounding

The output of the multiplier exponent was used in the calculation of the exponent before rounding. In the case of the FMA, the exponent was calculated by knowing the multiplier exponent and number of leading zeros. The exponent computation was performed using a row of 3:2 full adders in CSA format and a final row of ripple carry adder. The computation of the exponent is according to the following formula

$$Exponent = Exponent\ of\ AB - No\ of\ Zeros + m + 3 - Bias$$

A careful look at the above equation does not have any explicit representation of exponent of C. This has been incorporated in the structure of FMA due to the right shift of significand of C. The bias has to be subtracted because there is an addition of two biased floating point exponents. The value of 'm' was assumed to be 11 as it includes the significand bit along with the hidden bit. 2's complement subtraction was used to perform the above mentioned calculation

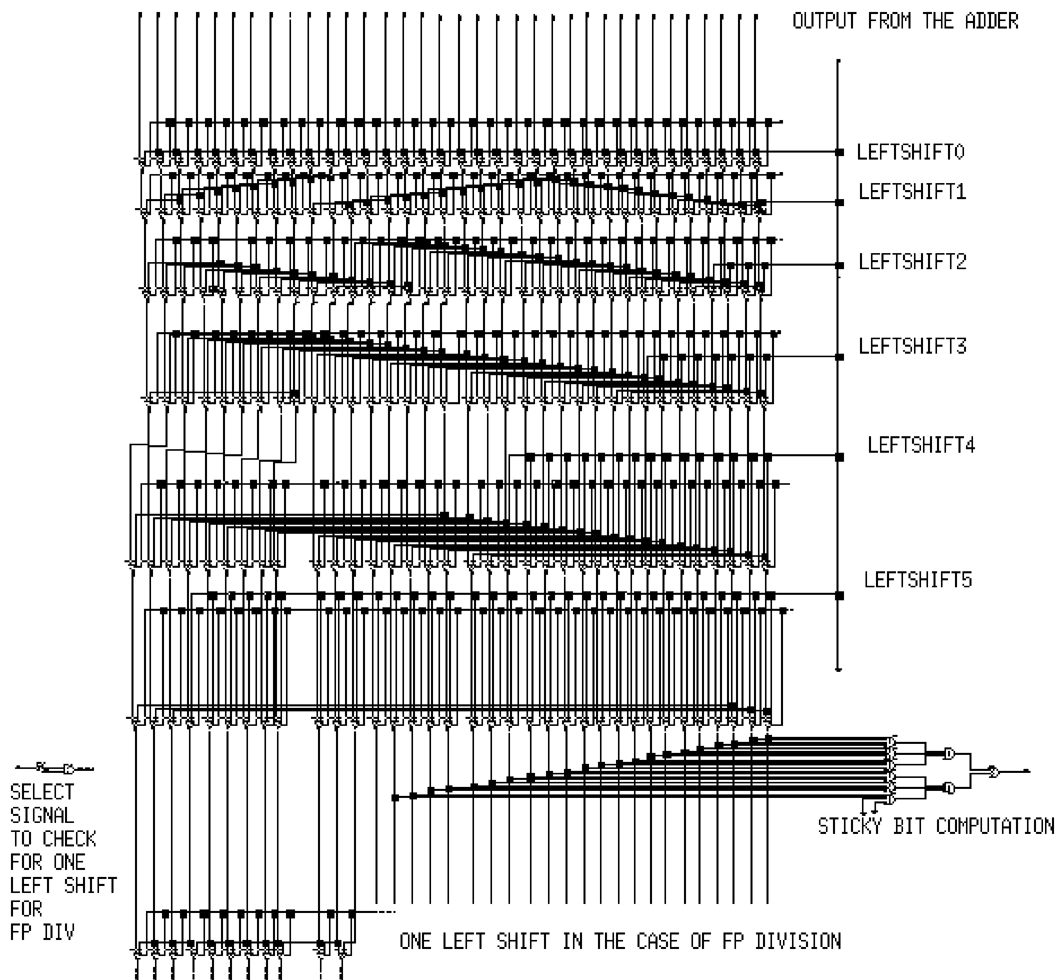


Figure 4-21 : Schematic of normalization output

SIGN	SIGNIFICAND	ACTUAL EXPONENT
X	01.XXXXXXXXXXXXXXXXXXXXX	YYYYY
X	11.XXXXXXXXXXXXXXXXXXXXX	YYYYY+ 1

Table 4-8 : Table of possible cases of exponent of multiplier

The computation of exponent of $A\ X\ B$ was explained in the multiplier section. The exponent of $A\ X\ B$ is 'YYYYY' or 'YYYYY + 1' depending upon the placement of

the decimal in the significand bits as shown in Table 4-8. This is analogous to the following scenario where X denotes 'don't care' and 'YYYYY' denotes any arbitrary valid biased exponent according to IEEE 754 floating point standards.

The exponent recalculation was necessary in the case of division because of the storage of the exponent values in the memory. The floating point rules state that the decimal point must always be represented after the hidden bit. In order to ensure the correct answer, the exponent should be recalculated based on the placement. A multiplication of two (1 + f) bit numbers produces an output with the decimal point after the 2f bits of data. For the exponent computation, the exponent of the multiplier output was then fed as one of the inputs to the 3:2 compressors. The count of leading zeros is fed as the other set of inputs. In order to reduce the computation, the direct value of the sum of 'm + 3' is hardcoded as one of the inputs to the 3:2 compressors.

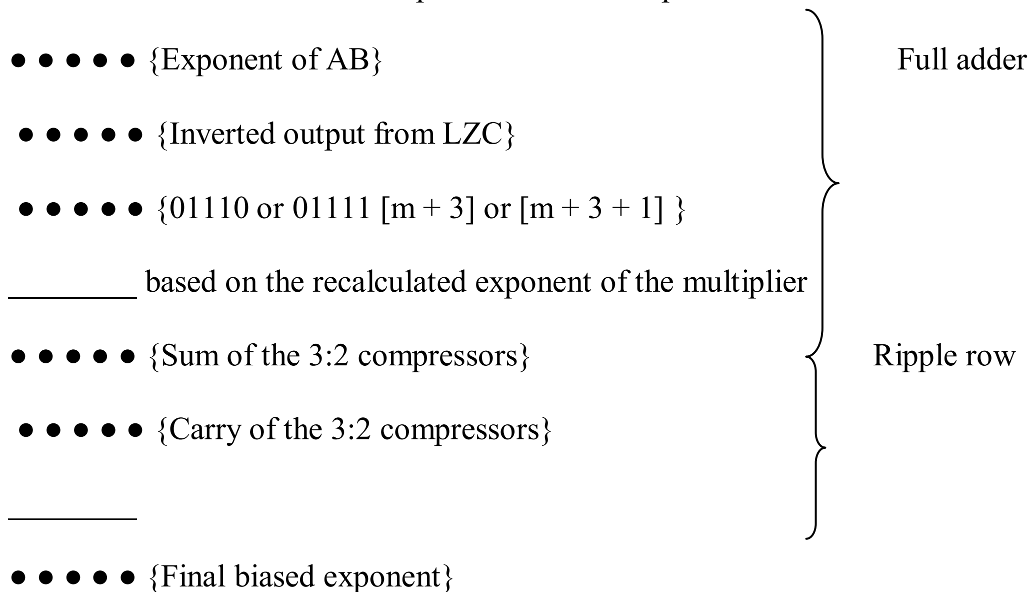


Figure 4-22 : Procedure of final exponent computation

Implementation of rounding

The rounding logic was implemented using an 11 bit adder. The method of rounding is to be decided prior to any rounding of the result. Round to nearest was achieved as part of the thesis as this is the most widely used rounding technique. The guard, round and sticky bits are fixed before actual rounding was performed. Consider the following number with a 10 bit significand. The rest are the remaining fractional bits

$L \ GRT$

1.XXXXXXXXXX XXX ,where L is the LSB of the normalized data, G is the Guard bit, R is the Round bit and T is the Sticky bit.

The rounding to nearest consists of addition of *rnd* bit to the LSB of the final output. The *rnd* bit is calculated according to the following formula

$$Rnd = G(R + T) + G(R + T)'L = G\,(T + L + R)$$

G, T, L and R are the guard, sticky bit, least significant bit and round bit of the 35 bits of normalized output. The *rnd* is added only if the guard bit is 1. Rounding was performed with the help of an 11 bit variable carry skip adder. The choice of the variable block sizes was chosen to have minimum ripple at the start and the end of the 11 bit adder and maximum skip at the middle blocks. Thus a group distribution was chosen as 1, 2,3,3,2. Figure 4-23 shows the schematic of above described logic. The OR of all bits apart from the significand bits is considered as the sticky bit. The sticky bit could be computed in parallel with the normalization step. The normalized output was used to calculate the actual rounded result. In certain cases, there might be a result of the form

1X.f where f consists 10 bits. In such cases there is a need for a one bit right shifter and a corresponding increase of the resultant exponent by 1.

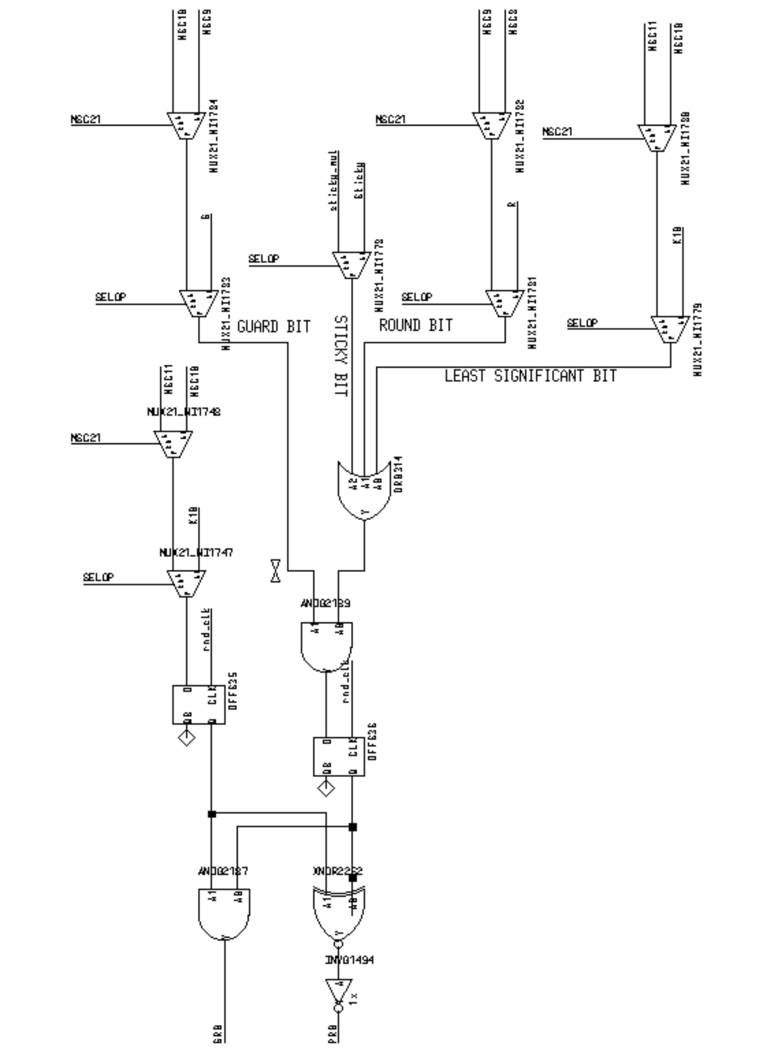


Figure 4-23 : Schematic of the circuit that performs $\boldsymbol{G}\ (\boldsymbol{R}+\boldsymbol{T}+\boldsymbol{L})$

<u>Final Exponent computation</u>

The final exponent is calculated at the end of rounding operation. Rounding might cause a final one bit right shift which would involve a corresponding increment of the exponent by one.

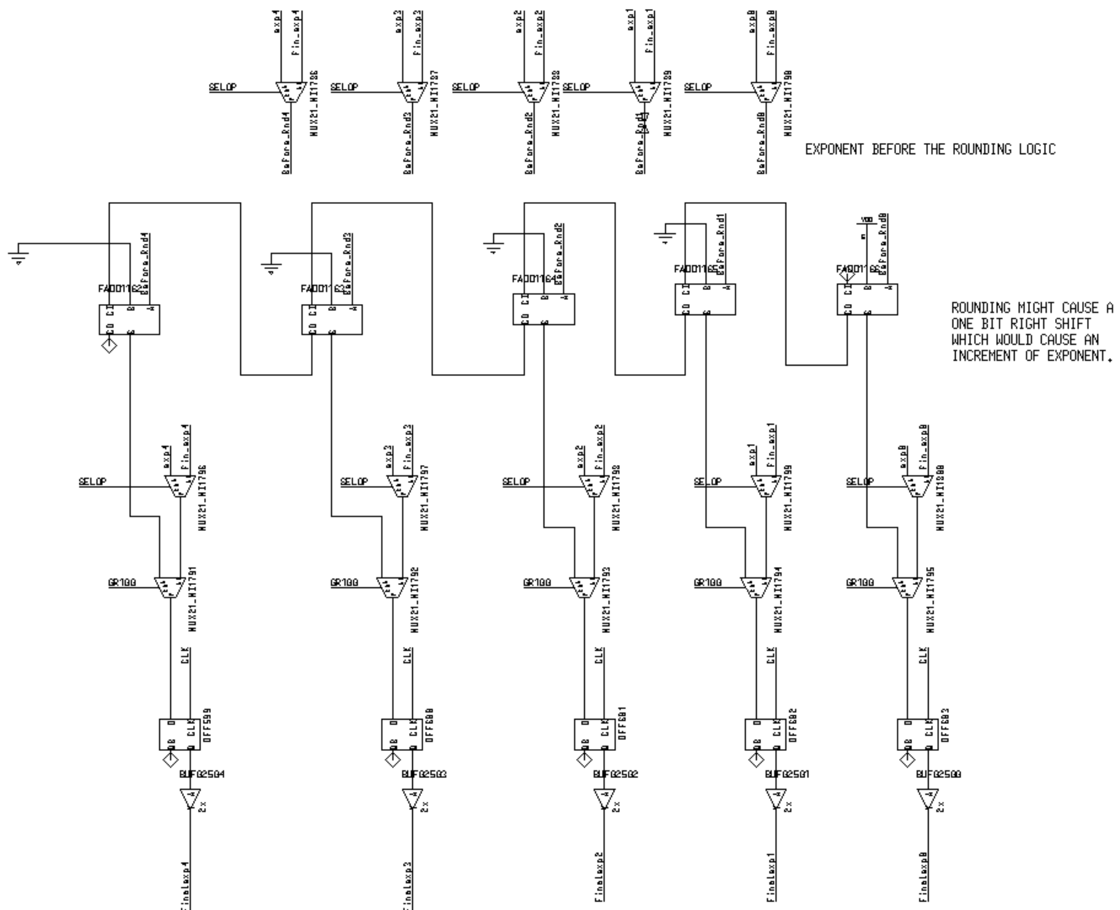


Figure 4-24 : Schematic of increment in exponent

The rounding logic is completed with the computation of exponent. The sign of the result completes the logic for the computation of the final result. Figure 4-25 shows the schematic of the circuit that computes the sign of the result.

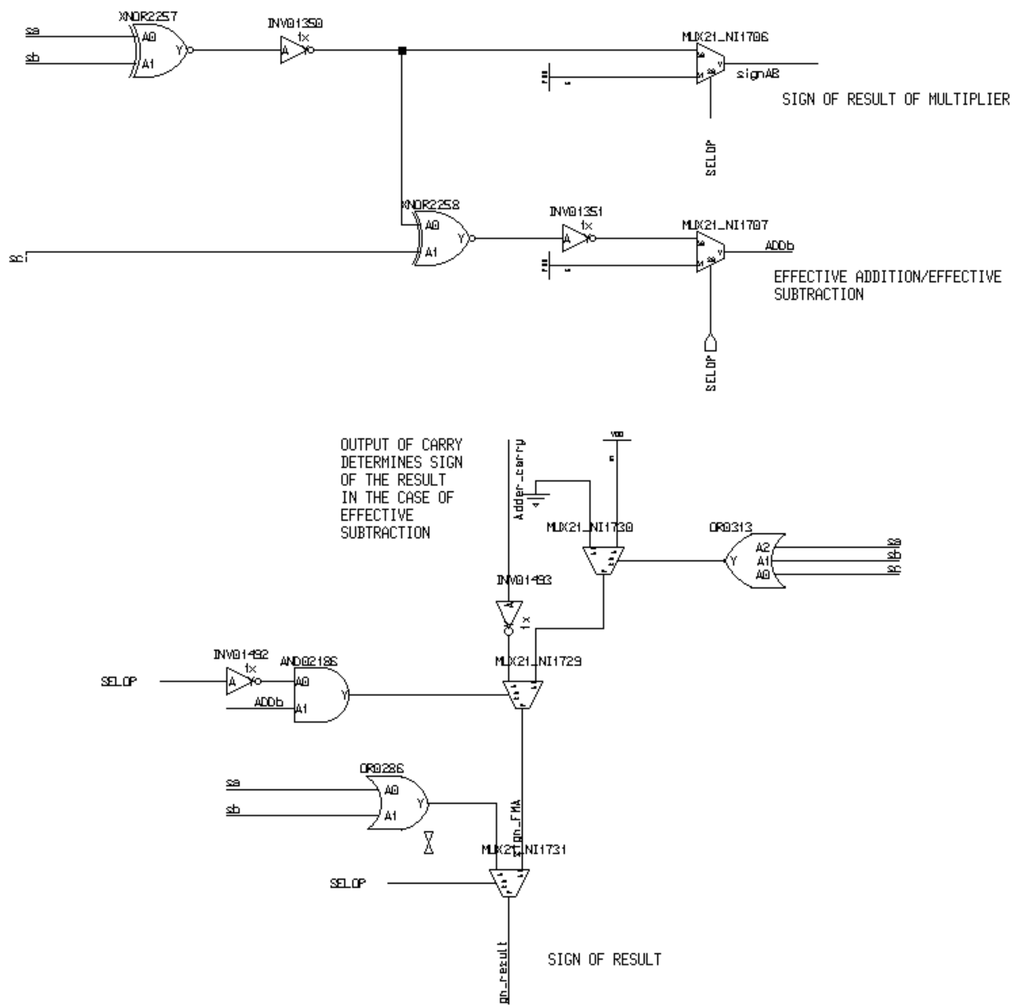


Figure 4-25 : Schematic of computation of sign of result

Conclusion

This chapter discussed the hardware implementation of various components used in the FMA and FP division. The knowledge of the logic blocks explained in this chapter should aid in the better understanding of the ALU. All the components explained in this chapter are integrated together to form the ALU.

CHAPTER FIVE

FMA

This chapter describes the operations performed in the FMA based on clock cycles. The control logic that is required for the proper operation of the ALU is explained in this chapter. The components that are active for appropriate cycles are briefly mentioned. The final result is reached after three clock cycles.

Operation modes of the ALU

The two modes of operation for this FP ALU are FMA and FP division. The mode of operation can be selected using the "SELOP" signal. The ALU can be used to operate in the FMA or the FP Division mode as shown in Table 5-1. The FMA mode can also implicitly account for FP addition, FP subtraction and FP multiplication.

MODES OF OPERATION IN A FMA BASED ALU	SELOP
FP addition, FP subtraction, FP multiplication	0
FP Division	1

Table 5-1 : Operational modes of the ALU

Design of a 3 bit counter

The FMA operation was targeted to compute the final result in three clock cycles. The first step is the design of a counter which tracks the number of clock cycles in the ALU. The counter generates values that can be mapped to the operations performed in

the specific clock cycles. It provides the easy sequencing of the operations. The Counter was designed to be compatible with both division and FMA.

CLOCK CYCLE	COUNTER FOR FMA (CBA)	COUNTER FOR DIVISION (CBA)
1	000	000
2	001	001
3	010	010
4	011	011
5	-	100
6	-	101
7	-	110

Table 5-2 : Table of counter values with corresponding clock cycles

The three bits of the counter are denoted as C, B, A, where 'C' is the MSB and 'A' is the LSB. The values of the counter increase from 0 to 3 in the case of the FMA and from 0 to 6 in the case of FP division as shown in Table 5-2. The 3 bit counter using J-K flip flop is shown in Figure 5-1. The hold generation circuit shown in Figure 5-2 holds the counter value after the hardware unit reaches the final clock cycle of operation.

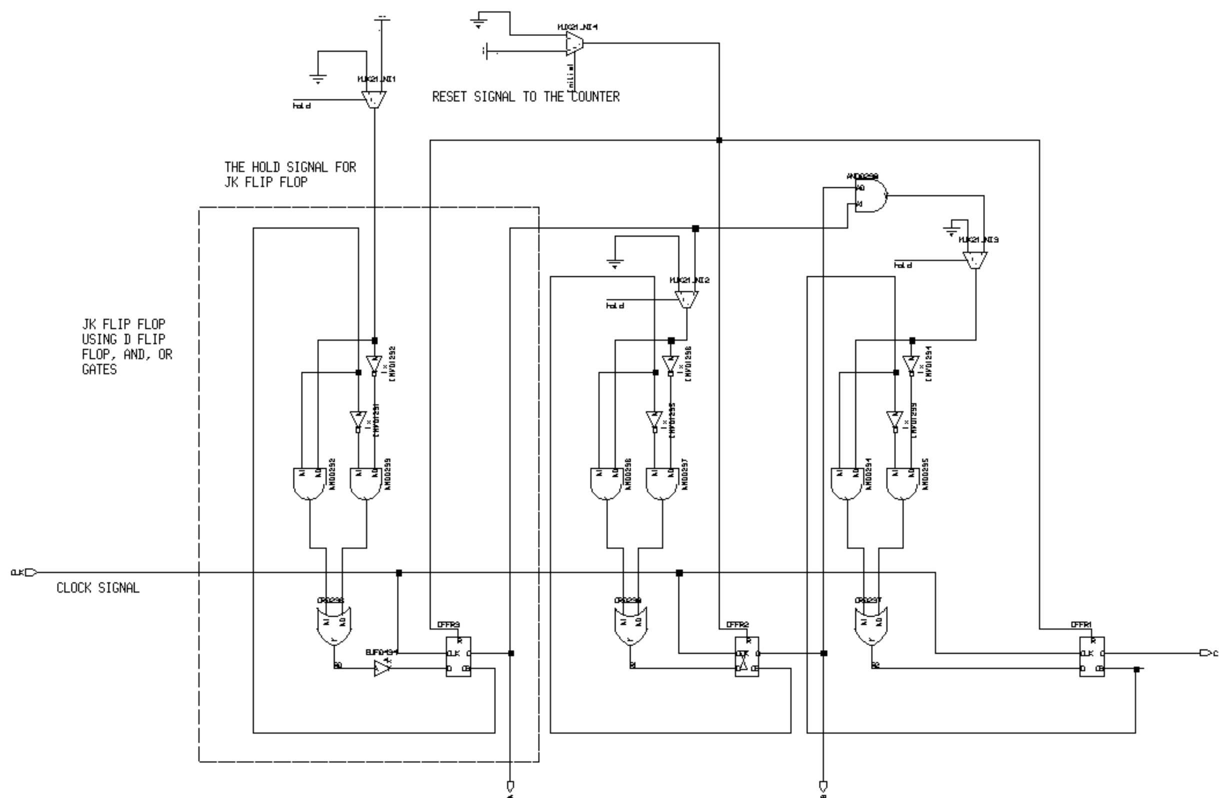


Figure 5-1 : Schematic of the 3 bit counter

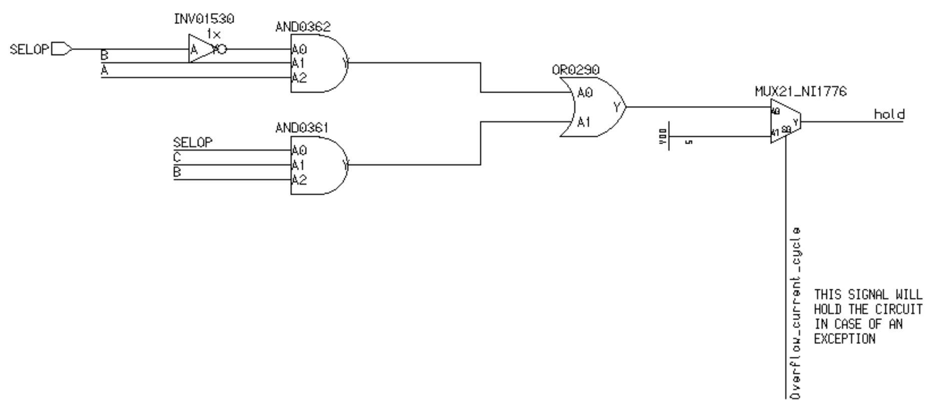


Figure 5-2 : Schematic of hold generation signal

Non overlapping clocks

The delay in the generation of the output of the counter at the start of every clock cycle resulted in the need for non-overlapping clocks edges. Non overlapping clocks for adder and the D register based memory were used for the thesis.

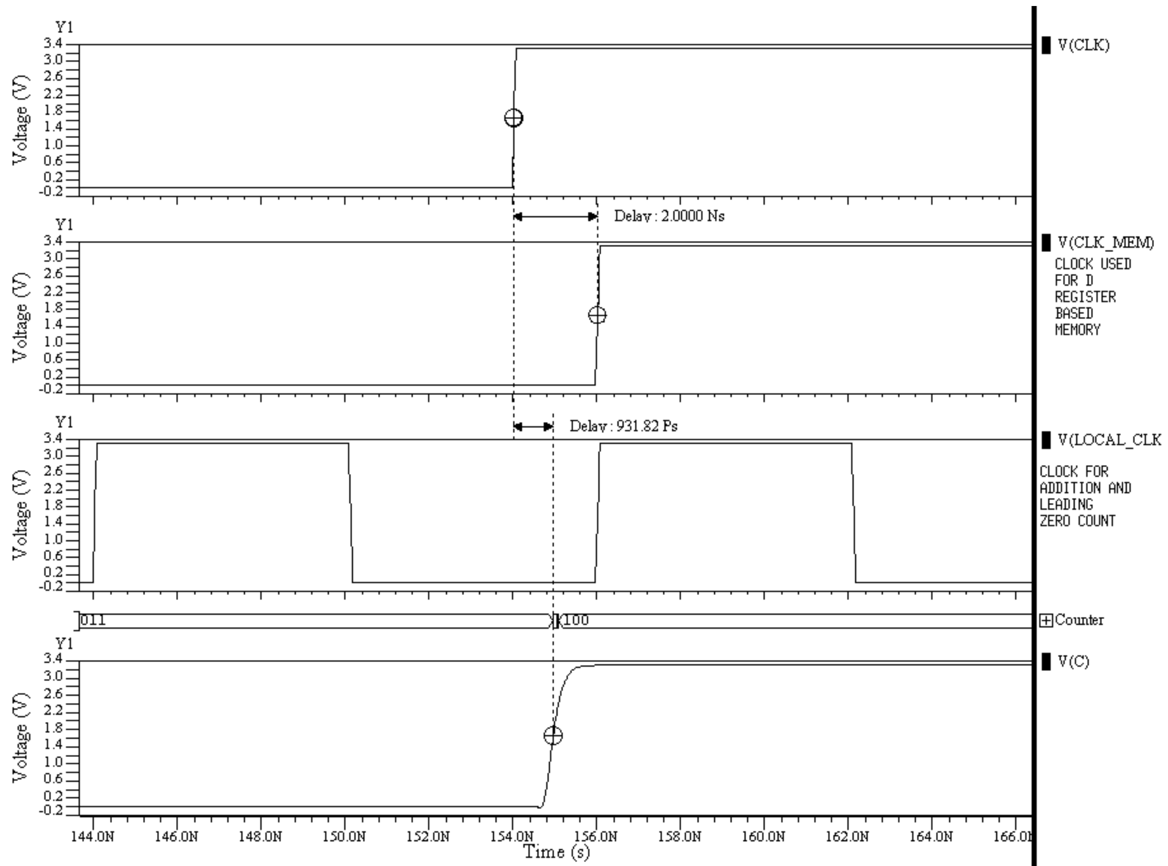


Figure 5-3 : Timing diagram illustrating the non overlap time between various clocks

<u>Local clock for addition and count of leading zeros</u>

The adder computes a two's complement based addition which requires a maximum of two additions. The FMA completes the addition and count of leading zeros within the same clock cycle. The generation of an asynchronous signal to denote the completion of the addition is not possible. This resulted in the need for a separate local

clock which can perform two serial additions and a count of leading zeros within a single clock cycle of the global clock. The time period of the local clock is one-third of the time period of the global clock.

There is a time delay of .93ns to generate the output of the counter as shown in Figure 5-3. The output of the counter dictates the operation to be performed after this minimum delay. Spikes were obtained when the local clock and the global clock were in phase. Non overlapped local clock was thus used to eliminate such spikes.

Clock for D register based memory

The clock for the D flip flop based memory is exactly the same as that of the global clock except that it is delayed by 2ns to account for the delay in the output of the counter. This is similar to the issue that was explained in the previous section. Figure 5-3 shows an illustration of the delay time that was used in the thesis. The delay time is greater than the delay in the output of 'C' obtained from the counter.

CLOCK	MODE OF OPERATION	VALUES OF COUNTER
Clock for addition	FMA operation	1 (Second clock cycle)
	FP division	1,3 (Second and Fourth clock cycle)
Clock for leading zero counter/Normalization	FMA operation	1 (Second clock cycle)
	FP division	1,3 (Second and Fourth clock cycle)
Clock for Rounding Logic	FMA operation	2 (Third clock cycle)
	FP division	5 (Sixth clock cycle)

Table 5-3 : Table of counter values for clock gating

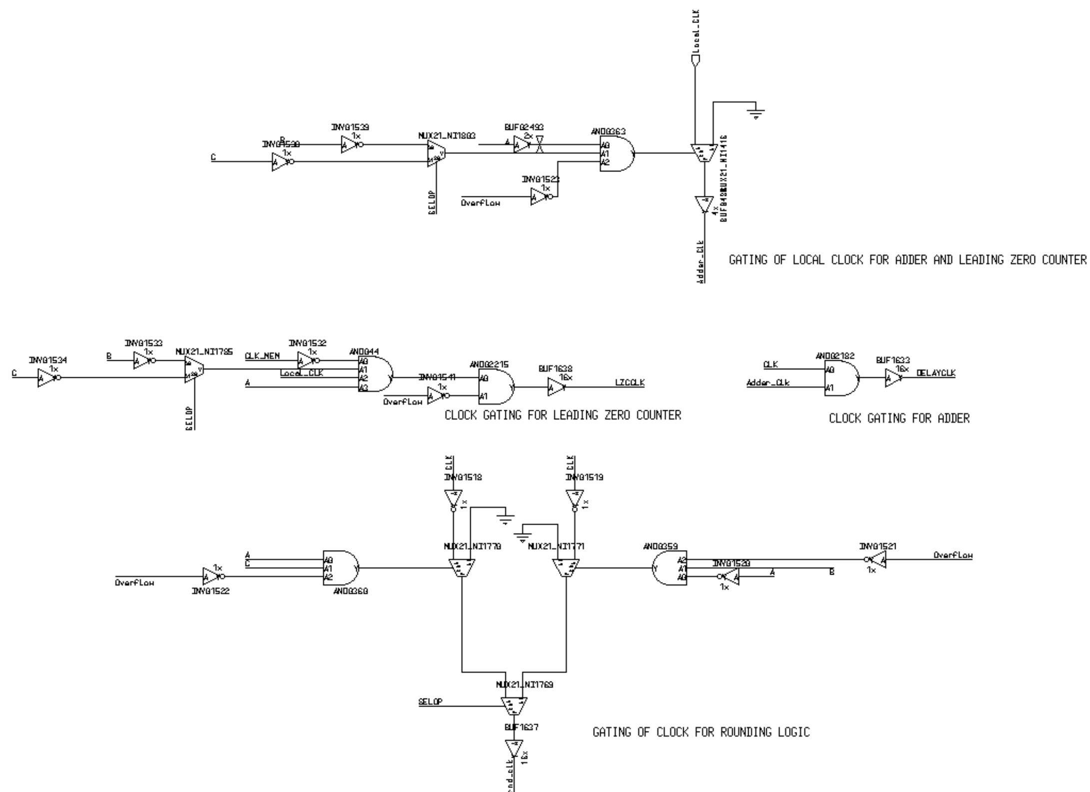


Figure 5-4 : Schematic of clock gating circuits.

Clock gating for addition and leading zero counter

The local clock that was explained in the previous section was gated to generate separate clocks for adder and leading zero counter. The global clock was gated to generate a clock for rounding logic. The schematic of the clock gating circuits is shown in Figure 5-4. The counter values for addition and leading zero count are known and this information was used to generate the corresponding clocks. Table 5-3 shows the counter values for which clocks are gated for individual operations. Figure 5-5 shows the clocks for the adder, leading zero counter and rounding logic. The set up time for the D register

is .05ns and the time period for the local clock is 12ns. Figure 8-4 of chapter eight shows the delay in output of 35 bit adder and the measured delay was 4.96ns.

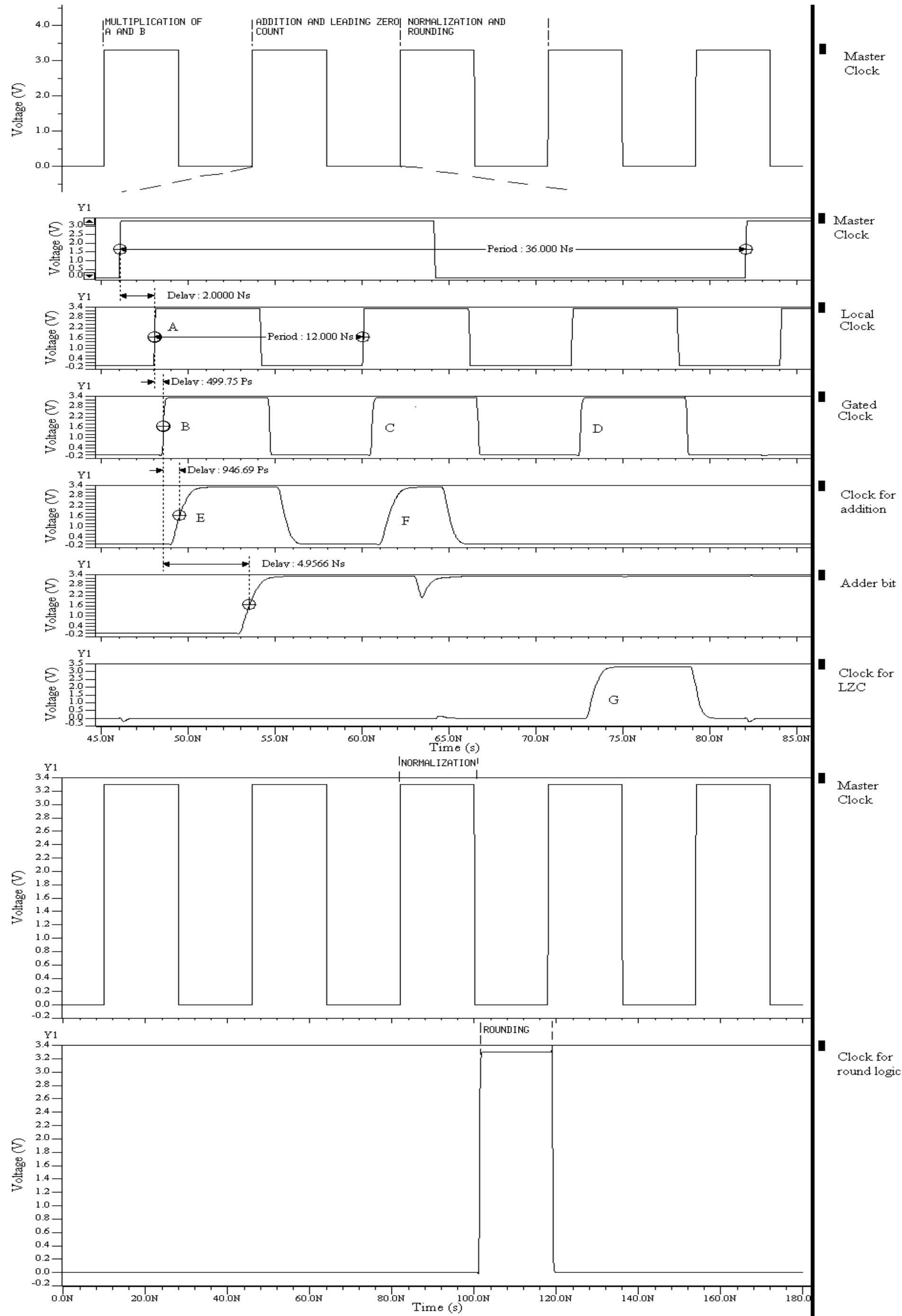


Figure 5-5 : Clock gating for adder, leading zero counter and rounding logic for FMA

The time period of the local clock was chosen to incorporate the three operations of two additions and one leading zero count in the second clock cycle. This provides three clock edges to perform the necessary operations. The delay of 2ns can be obtained using two stages of buffers. There is a 499.74ps delay to generate the gated clock from the local clock as shown by edge B in Figure 5-5. The gated clock has three clock edges. The rising edges (E and F of Figure 5-5) of first two are used to perform two's complement addition. The clock used for addition is obtained after a 946.69ps delay from the gated clock. The third clock edge denoted by G in Figure 5-5 is used to perform the count of leading zeros. These clocks were obtained by using the gated clock for appropriate operations.

Trigger for D register based memory

The trigger for the D register based memory depends on the values of counter. In the case of the FMA, only two rows of D registers are used to provide inputs to the multiplier. In the case of the FP division, all three rows are used to store intermediate values. These set of D registers hold both the exponent and significand values of the result.

Figure 5-6 shows the clocks for the three rows of memory. In case of an exception, all clocks to D registers are stopped using a multiplexer and the exception flag is raised. The timing diagram generated by the following circuit is given by Figure 4-2 in chapter four.

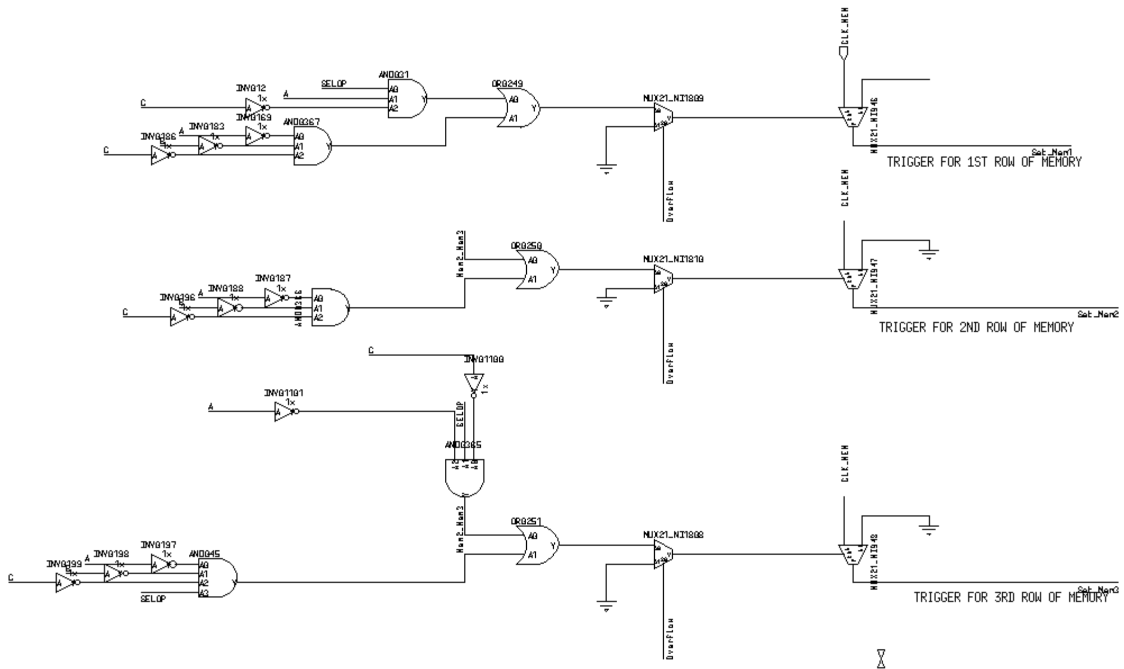


Figure 5-6 : Schematic of the control logic to select inputs to the multiplier

Cycle based working of FMA

The FMA and the FP Division operations are explained in detail based on clock cycles. Figure 5-7 shows the block diagram of the FMA operation from a clock cycle point of view. Let us consider the $AX\,B + C$ as an example of the FMA operation. A and B are inputs to the multiplier and C is the addend. The three operands conform to the IEEE FP standards and they are assumed to be normalized inputs.

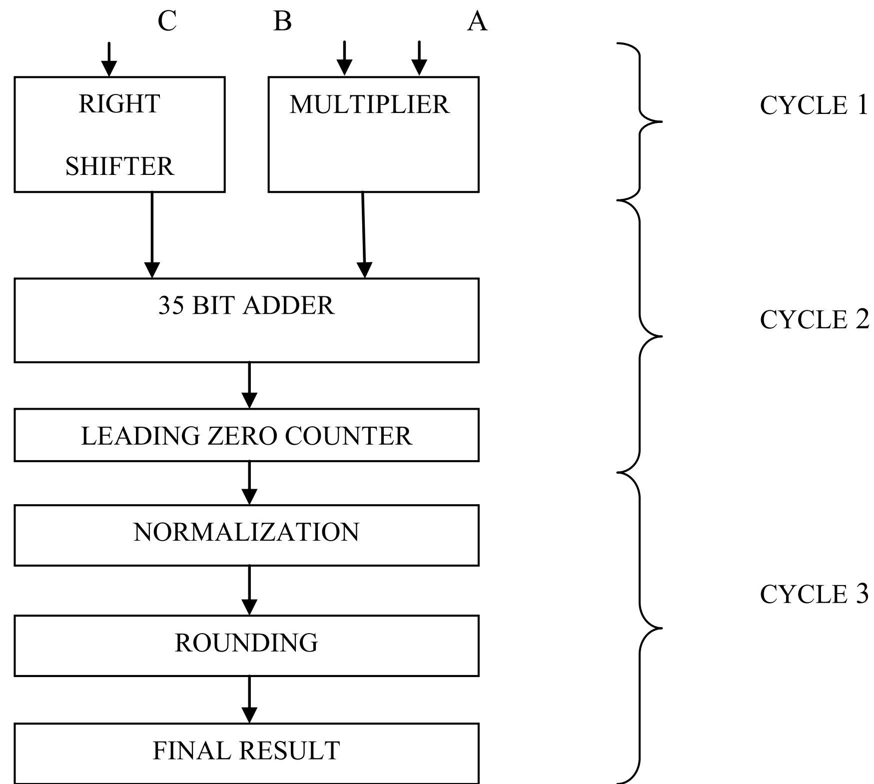


Figure 5-7 : Block diagram of FMA

First Clock cycle of operation

The first clock cycle involves simultaneous computation of multiplication and right shift operation. The components that are used in this clock cycle are

a) Wallace tree multiplier
b) Thirty four bit right shifter
c) Exponent Computation of the multiplier
d) Control logic for the first clock cycle

Multiplier and Right Shifter

The computations involving the 22 bit multiplier and the thirty four bit right shifter is done in parallel. The alignment of the addend is determined from the shift distance calculated using the exponents of $A, B\ and\ C$. The right shifter takes 11 bits of addend as the input and results in an output of 35 bit data. The 22 bit multiplier takes significands of A and B as inputs and produces a 22 bit sum and carry. The sum and the carry are added together using the 22 bit adder to form the product of multiplier. The design and implementation of the above mentioned components was explained in detail in corresponding sections of chapter four.

ROW OF D REGISTERS	INPUTS FOR MULTIPLIER
Row 1	Input A
Row 2	Input B
Row 3	Unused for FMA operation

Table 5-4 : Values stored in the D registers for the FMA operation

Exponent computation of multiplier and control logic

The exponent of the multiplier is also computed during the first clock cycle of operation. This was explained using dot diagram shown in Figure 4-9. The input to the multiplier is obtained from the rows of D registers as explained in chapter four. Table 5-4 shows the rows of D registers that are used as inputs to the multiplier. The control logic for the first clock cycle involves the selection of A and B. The control signals 'Initial' and 'Mem_Input_Sel' are used to select the appropriate inputs from the set of D registers as shown in Figure 5-8.

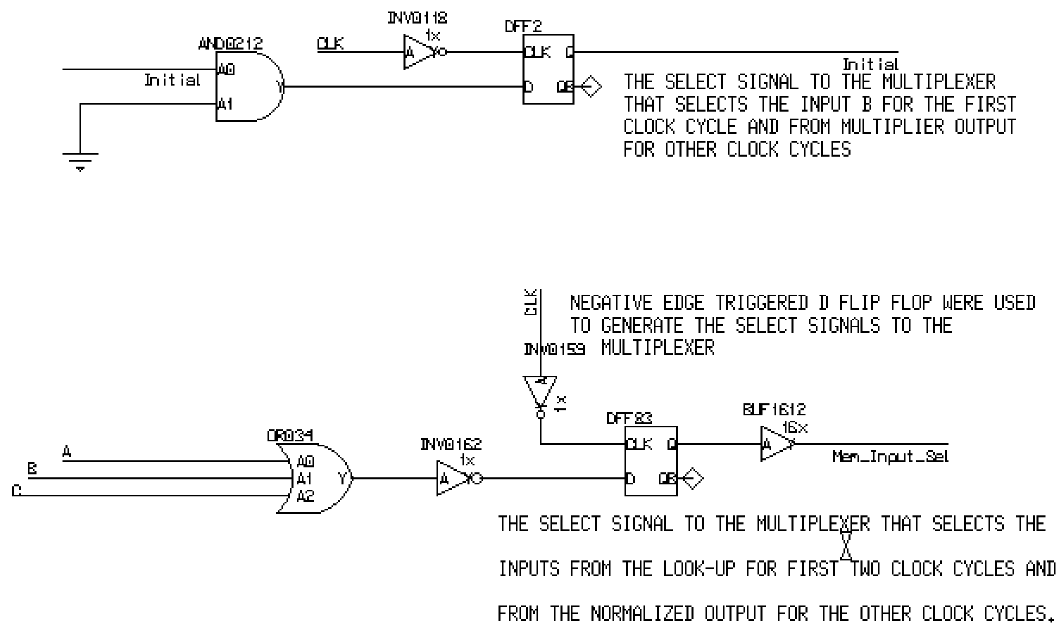


Figure 5-8: Schematic of the control signals for selecting inputs to the multiplier from various sources

<u>Second clock cycle of operation</u>

The addition of the aligned addend and the product of the multiplier are performed in this clock cycle. Normalization requires the number of leading zeros present in the 35 bit adder result. The components used in the second clock cycle are

a) 35 bit Adder

b) LZC

A local clock was used to compute the two additions and the count of leading zeros within a single clock cycle of the global clock. The working of ‘35 bit adder’ and the ‘leading zero counter’ was explained in Chapter four.

35 bit Adder

The adder computes the addition of a 35 bit aligned addend and 22 bits of product of multiplier. There is a control logic which determines the operations that are performed using the adder. ‘TWOCOMPSEL’ was the signal used to select the level of addition. In this thesis, level of addition denotes the number of additions required to account for two's complement addition. In effect we will have two levels of addition for two's complement operation. The outputs of multiplier and the shifted addend are selected if the value of ‘TWOCOMPSEL’ is ‘zero’. The output from the thirty five bit adder is selected if the value of ‘TWOCOMPSEL’ is ‘one’. After the addition is completed, this signal is reset back to zero at the negative edge of the adder clock.

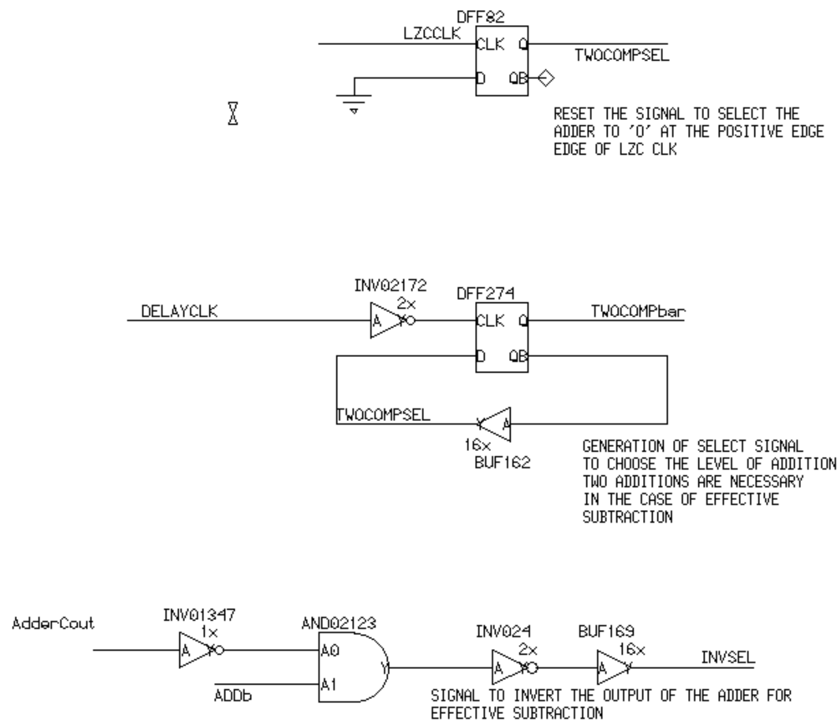


Figure 5-9 : Schematic of the control logic used for the adder

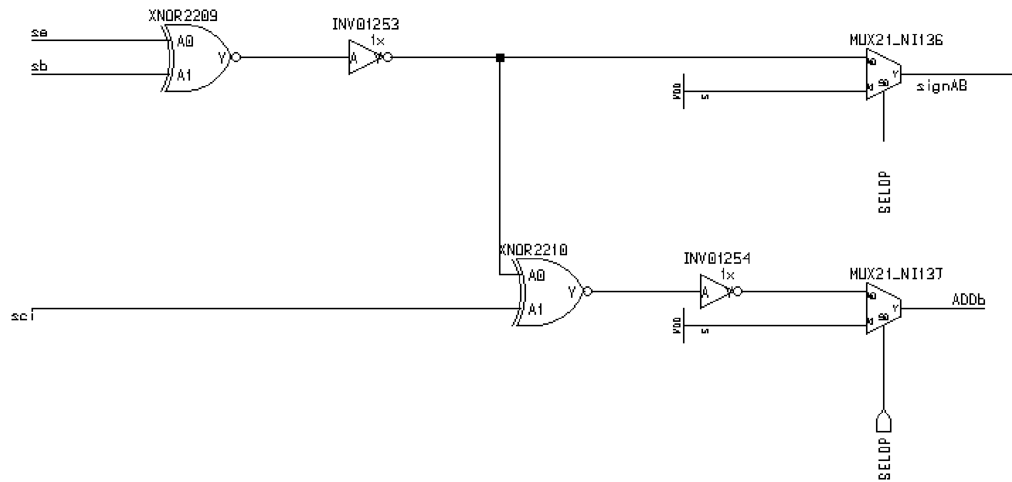


Figure 5-10 : Schematic of the control logic for effective addition/subtraction

The signal ‘INVSEL’ was used to select the inverted output from the adder for two’s complement subtraction. Figure 5-9 shows the schematic of the control logic used for the 35 bit addition.

The signal ‘ADDB’ denotes effective addition or subtraction of the FMA operation. Operations like $A\,X\,B + C$ and $(-A\,X\,B) - C$ are treated as effective addition operations. The signal ‘signAB’ denotes the effective sign of the multiplier output.

Leading Zero Counter

The final step during the second clock cycle is the calculation of the count of leading zeros. This is done using a LZC and its working was detailed in chapter four. The output of the 35 bit adder was triggered at the positive edge of the LZC clock. Figure 5-5 gives the timing diagram of clocks of adder and the leading zero counter during the second clock cycle.

Third Clock cycle of operation

The final clock cycle of the FMA operation involves normalization and the rounding logic. The computation of the final exponent is also done during the third clock cycle. The operations performed in this clock cycle are

a) Normalization

b) Rounding

c) Computation of final exponent

Normalization

Normalization is performed using a 34 bit left shifter. The output of the 35 bit adder is fed as the input to the left shifter. The amount of shift is decided by the LZC. The sticky bit used for rounding is calculated at the end of the left shift

Rounding

The final step in the FMA involves the rounding of the normalized output. Rounding to nearest was performed using the 11 bit adder. The details of rounding logic were provided in chapter four. The rounding was performed only once after the completion of all computations. This was possible with the help of a gated round clock as shown in Figure 5-5. The output of rounding logic is the result of $A\,X\,B + C$ in normalized FP format.

Computation of final exponent

The final exponent is computed as shown in Figure 4-22 and is explained in the section preceding the mentioned diagram. The final exponent may be incremented by one

because of a single one bit right shift that might be necessary to denote the result in normalized format.

Conclusion

This chapter described the operation of the FMA as a sequence of events starting from the first clock cycle to the final result in the third clock cycle. The output of the FMA is obtained at the start of the fourth clock cycle. The results of the FMA operation are given in chapter eight.

CHAPTER SIX

TESTABILITY

Testability can also be defined as the controllability and observability of a circuit to determine defects. Failures in the operation of digital circuit are addressed in the testing process. Testing is performed either at the transistor level or logical level. The standard fault model for logic level testing is a stuck at model. There are also advanced self testing schemes like Built-In Self Testing (BIST) schemes which are widely used in memory testing. The presence of dedicated hardware testing methodology like BIST in a microchip is area intensive. Testability also involves the identification of minimum test patterns to a logic circuit [32].

Objective

Some hardware techniques for testing were analyzed but the identification of hardware faults is time consuming as well as very exhaustive. Such an approach is possible for critical parts of the design. Even then, it is unlikely to confidently predict hardware component to be working functionally. The best practice for larger designs is to prove the correct functionality of the circuit for all possible input combinations.

In the case of large designs, the usual practice is to identify the most critical part of the design and associate a testing method with that critical circuit. This approach simplified the time for testing and satisfied the proper working of the logic block that is most frequently used.

Behavior model based Test

The complexity of the floating point operation increases the number of cases to be tested [33]. In order to speed up the process of finding the initial vectors, the behavioral model of the ALU using Verilog HDL was used. The actual hardware was mimicked in the software and the required nodes were checked for the occurrence of a pattern of data. [34]

Identification of critical component

The multiplier block is the most frequently used component in this thesis. This is because part of FMA hardware unit was used in the division operation.

OPERATIONAL MODES OF FMA BASED ALU	MULTIPLIER	ADDER	LEADING ZERO COUNTER	SHIFTER
FP addition, FP subtraction, FP multiplication	33.33%	22.22%	11.11%	66.66%
FP Division	83.33%	22.22%	-	50%

Table 6-1 : Table of percentage use of various hardware components for different operational modes

The first step is the identification of the critical elements of the design. Table 5-1 shows the activity of each hardware component with respect to the total number of clock cycles as a percentage. The rounding logic was not included in the above analysis

because rounding is the final step in all the operational modes. The exponent addition logic block is also not included because the delay in the calculation is very less compared to the other critical components.

The multiplier is active for the first clock cycle in FMA operation and is active for five clock cycles of operation in the case of FP division. The adder is active for the second clock cycle in FMA and for two clock cycles of operation in the case of FP division. The shifter is active for first as well as third clock cycle in FMA and for three clock cycles in the case of FP division.

In addition to the above consideration of activity, special mention of the complexity of a component should also be taken into consideration. The logic levels of the multiplier and the shifters are almost similar but the complexity of the multiplication is far greater than the shifter. The adder was also not considered because it is dependent on the output of the multiplier. The normalizing shifter is dependent on the output of the adder. Stated otherwise, it is imperative that testing is performed for the multiplier.

Multiplier testability

Hardware Circuit for test

The multiplier consists of Booth encoder, 3:2 compressors and 4:2 compressors. The different logic levels of the multiplier are reduced by means of compressors. The approach was to test the single most important hardware circuit which will definitely satisfy the testing goals. Fault identification should also be easier for such tests.

The modified booth encoder was also investigated as an option to be tested but was rejected because of the number of such components (72) and the difficulty in arriving

at a minimal set of vectors for the same. The 3:2 compressors occupy the middle of logic levels in the multiplier tree. This made it very difficult to identify a pattern. Also, the tree structure consists of two levels of 3:2 compressors. Thus, a test involving the middle logic level would be difficult as well as non-generic.

The 4:2 compressor forms the final logic level of the multiplier tree. This component produced the maximum reduction of bits. The 4:2 compressor is the only component that was custom designed in the multiplier. The rest of the components were basically taken from the standard cell library. The probability of a fault in the 4:2 compressor is greater than those components from the standard cell library. Thus, 4:2 compressors were used to test the operation of the multiplier.

Steps for identification of input vectors

The 11 X 11 bit multiplier can result in as much as 1 million combinations. The 4:2 compressors have 4 inputs of data, which implies there are a maximum of 16 input patterns that are possible. For e.g., The inputs were observed for the twenty 4:2 compressors in the ALU design by using the behavior model of the multiplier using Verilog HDL. These were arranged for each compressor and written in an output file. The output consists of a million combinations and the output file was analyzed to generate the least set of test vectors.

The output data from behavioral model was then fed to a Perl program. A Perl program was used to produce the combinations of inputs that appear for each 4:2 compressor. Another Perl program was then used to detect a set of vectors that have

unique data. But the nature of the output warranted the iterative approach to solve the problem resulting in the overlap of certain data in a few compressors.

Thus a partial automation was used to create a set of vectors that correspond to all combinations of data that occur in each 4:2 compressor. A maximum of three iterations were used to identify thirty one vectors which can pseudo exhaustively test the 4:2 compressors in the multiplier.

If there is any fault in a particular column, we can easily track the upper layers of 4:2 compressors and the modified booth encoder for errors. This type of test cannot claim to be the most effective as the identification of minimum number of vectors is dependent on the semi automation. Yet, this method can definitely claim to include the complete test of the multiplier block.

The twenty bits of multiplier output are based on the output of 4:2 compressors. Also, careful analysis of the data found that certain compressors (10-17) are the most critical and have large variability of outputs. For e.g., the compressors (10-17) result in the combinations of 0-15 combinations whereas certain others like 20th compressor result in just 2 or 3 as combinations of data.

$$\text{Fault coverage of test} = \frac{\text{Number of output bits tested}}{\text{Total number of output bits}} = 90.09\%$$

Testability in Layout

The multiplier block was created in layout and was tested for the test set by providing a piece wise linear input. The input capacitance calculation is shown in the Appendix C.

PRIMARY INPUTS		4-2 COMPRESSOR OUTPUTS																			
X	Y	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
11010101000	10100101111	3	0	7	8	1	14	8	9	9	5	7	4	1	5	1	8	0	8	0	6
11010101010	11000100101	3	1	6	0	13	2	12	10	10	8	3	14	2	7	4	9	7	9	2	5
10010000000	10000001010	2	3	4	2	1	5	7	12	0	12	4	8	0	0	4	12	0	0	0	0
10010000001	11011110001	2	3	5	10	1	13	15	0	4	4	8	1	13	0	1	13	1	1	0	0
10010000011	11111000100	2	3	5	11	9	1	5	5	11	2	12	1	1	5	4	12	1	3	1	0
10110000110	10101101110	2	3	6	1	13	9	9	3	12	9	5	5	3	0	4	13	3	2	0	0
10110101000	10100000011	2	3	6	9	1	2	4	1	5	13	11	14	2	6	2	8	2	8	0	6
10011101011	10100111011	2	3	4	3	13	10	8	10	13	11	13	11	8	1	7	12	0	10	0	5
10011110111	10101111011	2	3	4	3	13	14	8	15	13	11	13	10	8	3	6	12	0	10	2	1
10101110111	10100011111	2	3	4	7	5	6	4	7	13	11	13	15	8	2	6	0	3	10	2	0
10111101110	10101110011	2	3	6	9	9	10	12	14	3	14	10	12	3	6	6	12	2	11	3	6
10101010000	10111111111	2	3	4	7	13	13	11	13	12	13	13	13	9	1	5	5	4	4	0	0
11010011000	11101000001	3	1	7	0	5	3	6	4	13	8	13	9	8	1	1	13	6	2	0	6
11110010111	10110100101	3	0	7	3	13	5	7	14	6	14	13	3	12	5	2	12	2	5	3	0
10101010101	10011101111	2	3	4	6	13	9	3	13	12	9	15	10	8	3	7	4	7	6	3	3
10011011101	10001110011	2	3	4	2	5	15	8	13	1	6	6	15	8	0	2	8	7	3	0	4

PRIMARY INPUTS		4-2 COMPRESSOR OUTPUTS																			
X	Y	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
10011111001	10101010001	2	3	4	3	13	6	14	4	15	0	15	1	15	1	7	13	6	15	0	6
10001111101	10110101101	2	3	4	13	1	10	2	6	12	3	10	0	7	3	6	0	5	12	3	5
10010100111	10110111111	2	3	4	11	1	12	7	14	8	15	9	1	5	5	3	8	2	14	2	0
11011100111	11010000001	3	1	6	1	13	0	13	11	11	10	14	8	11	0	7	12	1	13	3	1
10001101010	10110101000	2	3	4	13	1	10	0	6	5	5	3	4	2	0	3	0	5	14	3	7
10010011000	11000111111	2	3	5	2	5	11	10	8	12	8	11	2	4	5	1	13	4	0	0	6
10010011101	11110110101	2	3	5	11	1	15	2	14	8	1	10	6	10	2	0	12	7	1	2	5
10010100111	11000110000	2	3	5	10	1	0	13	11	11	3	0	0	6	7	7	12	4	12	1	1
11000000000	10000000100	3	0	4	6	5	5	7	12	0	4	0	4	12	0	0	0	0	0	0	0
11000000010	10110000001	3	0	5	7	1	1	5	5	7	12	2	4	9	0	1	1	1	1	1	3
11000000100	11000011101	3	1	4	4	5	13	11	0	4	1	5	4	8	0	0	1	2	2	0	2
11000001010	11010010001	3	1	4	5	5	9	1	7	1	13	1	5	11	2	0	1	2	7	3	5
11000011001	10011110000	3	0	4	15	1	7	4	1	2	6	6	4	13	0	3	0	3	3	2	6
11000100010	10001100001	3	0	4	14	9	4	3	2	14	9	4	7	10	3	0	5	1	13	1	3
11000100011	11001011101	3	1	4	12	9	8	9	3	6	7	12	1	14	4	3	4	3	14	0	2

Table 6-2 : Table of input combinations to the 4:2 compressor

Table 6-2 shows the input patterns occurring in the 4:2 compressors. Any error could be traced by observing the data at the input and output of the 4:2 compressors. The error can be easily found in the upper reduction levels if the expected bit pattern is not obtained at the input of 4:2 compressor. The other scenario could be an error in the 4:2 compressor itself. In such cases, the output of the multiplier can be checked for the correctness.

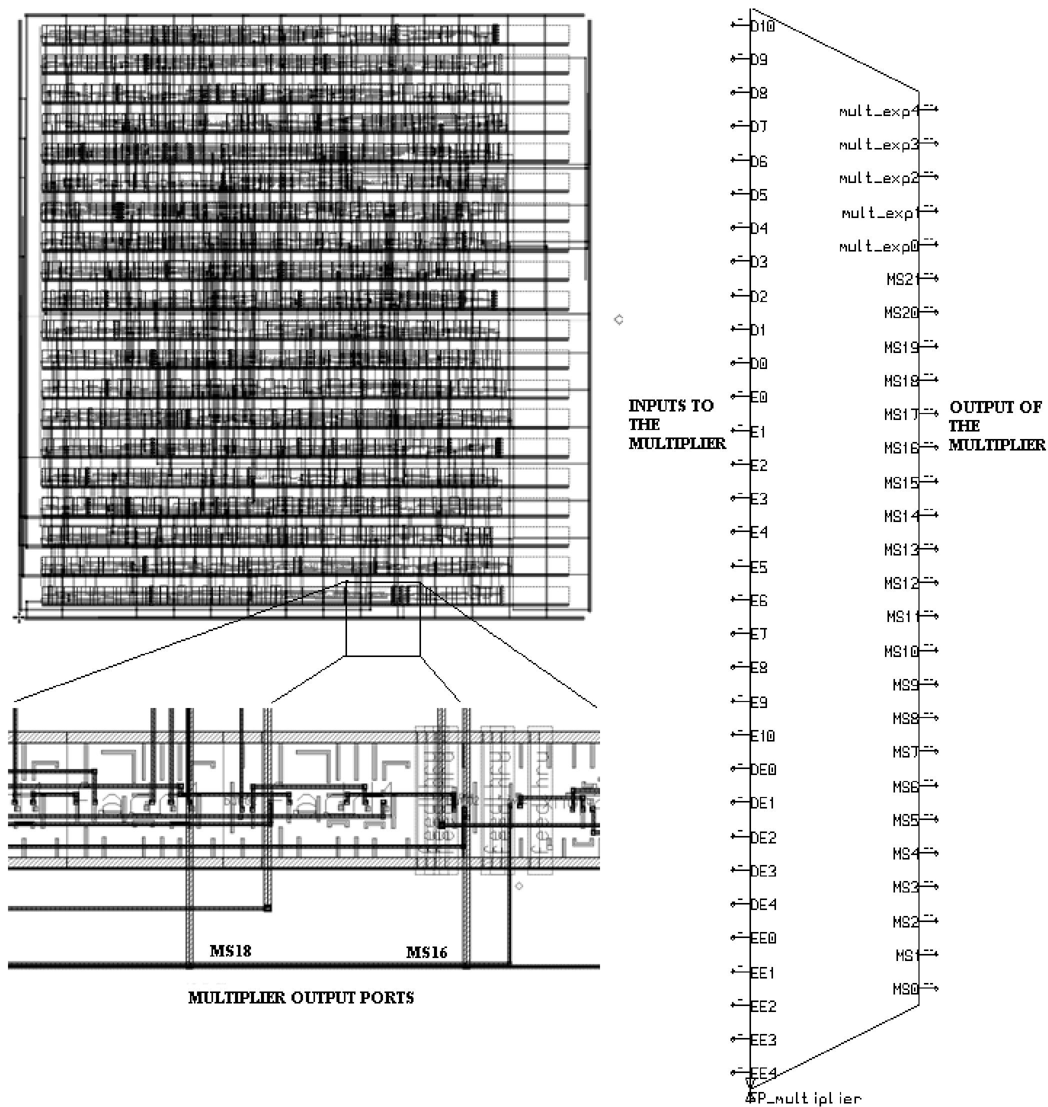


Figure 6-1 : Layout and schematic symbol of 11 X 11 bit multiplier

Conclusion

Testability forms a critical part of any design. This chapter dealt with the generation of a minimum number of test vectors which proved the correct functionality of the multiplier used in this thesis.

CHAPTER SEVEN

FP DIVISION

This chapter deals with the description of hardware circuitry used in the FP division. The FP division was implemented in six clock cycles. The structure of this chapter is from the point of view of clock cycles and the computation of FP division is described progressively for each clock cycle.

Introduction to division

Inputs for FP division

Let us consider A to be the dividend and B to be the divisor operands.

S_A , S_B denote the sign bits of A and B. M_A, M_B denote the significand bits of A and B. E_A, E_B denote the exponents of A and B.

Iterative division

Newton-Raphson algorithm uses Newton's method to converge to the quotient [35]. The strategy of Newton-Raphson is to find the reciprocal of the divisor and multiply it by the dividend to find the final quotient Q. In numerical analysis, Newton's method can often converge quickly, especially if the iteration begins "sufficiently near" the desired value.

Algorithm for Newton Raphson

The steps of Newton-Raphson are:

Step 1: Compute successively more accurate estimates of the reciprocal ($X_1 ... X_i - 1 ... X_k$)

Step 2: Compute the quotient by multiplying the dividend by the reciprocal of the divisor

Q = Dividend (A) * Reciprocal (X_k)

Assuming the divisor is scaled between $0.5 < B < 1$, the initial estimate for the reciprocal of the divisor with five bit approximation is

$$X_0 = \frac{1}{1.xxxxx + 2^{-6}} + 2^{-7}$$

$1.xxxxx$ - Decimal value for five bit approximated divisor value.

X_0 - Five bit initial reciprocal approximation.

The convergence of Newton's method doubles for each iteration. A 12-bit convergence is obtained within two iterations. This is because of the presence of the hidden bit. The 5 bit approximation also requires just 32 rows of memory to store the initial approximated value. It is assumed that memory read from cache takes place in the clock cycle prior to the start of any computation for FP division. `Successively more accurate estimates of the reciprocal can be calculated using

$$X_{i+1} = X_i\,(2 - BX_i)$$

The Newton-Raphson and Goldschmidt algorithms are the two fastest division algorithms used in the industry. Goldschmidt algorithm uses series expansion to converge to the quotient [36]. The strategy of Goldschmidt is to repeatedly multiply the dividend and divisor by a factor F to converge the divisor to 1. When the dividend converges to the quotient Q, the degree of the Taylor series rises and the algorithm approaches the correct answer. The hardware implementation of Newton-Raphson as well as Goldschmidt is very similar. Both methods need two multiplications per iteration and both require an initial look-up table.

Faster Division

There are three speedup methods for improving the time to complete FP division

1. Reduction of number of multiplications.
2. Usage of narrow multiplication.
3. Implementation of fast multiplications.

Only 2 iterations are required to find the quotient if the reciprocal approximation is within an error bound of 2^{-5} for a 16-bit FP number. The performance of division can be improved with the above mentioned speedup methods. This has made Netwon-Raphson's method a common choice for high-performance CPUs.

Iterations of Newton Raphson algorithm

The Newton Raphson approach is based on the bit accuracy of the initial approximation.

Initial approximation = X_0

First Iteration = $X_0 (2 - BX_0)$

Second Iteration = $X_0(2 - BX_0) [2 - BX_0 (2 - BX_0)]$

The value at the end of second iteration has to be multiplied by the divisor to compute the unrounded quotient. This quotient value will be rounded using rounded to nearest technique to produce the final result.

Division architecture

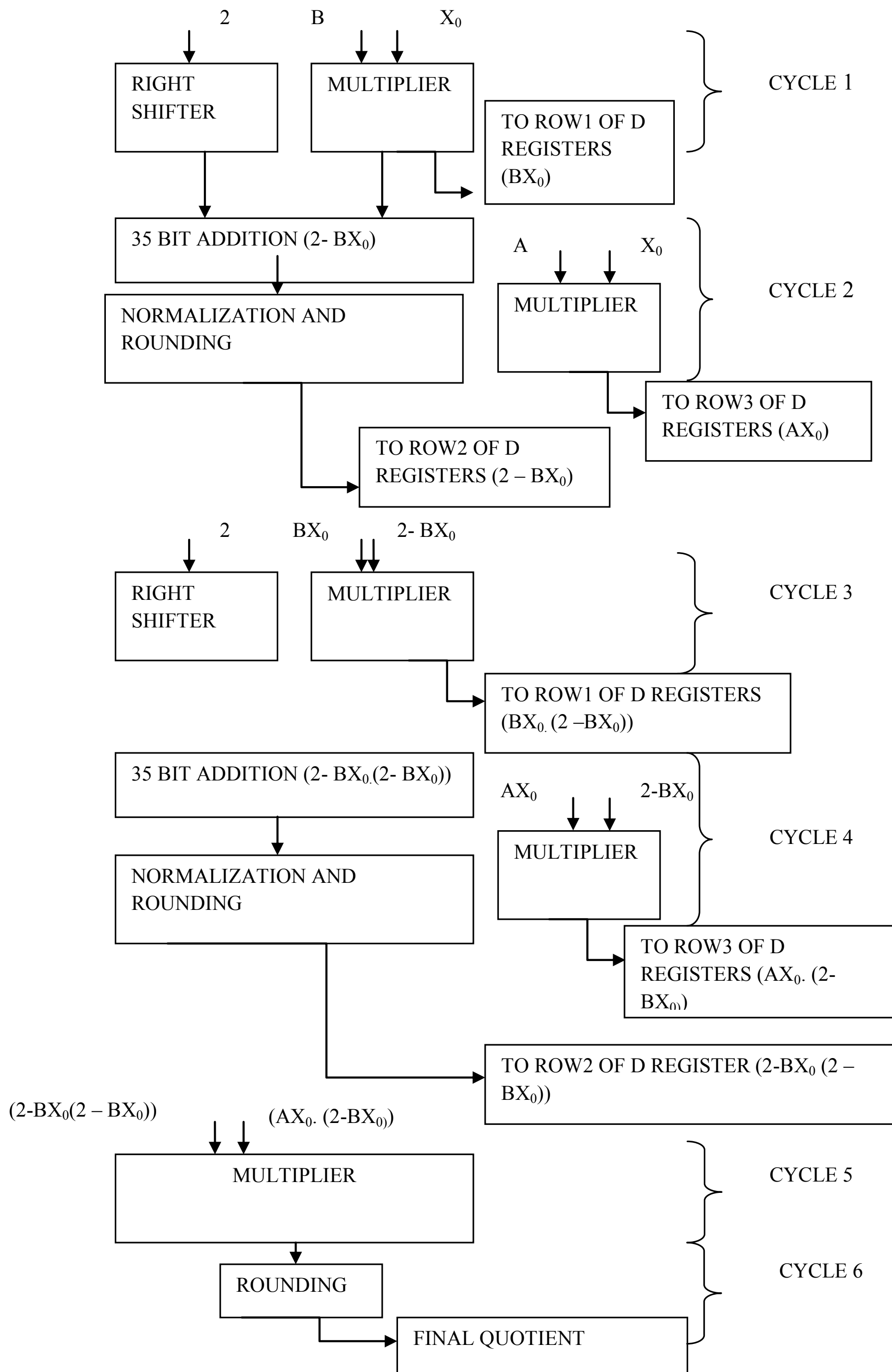


Figure 7-1 : Block Diagram of division architecture

Usually FP division is implemented using two multipliers working in parallel. A unique method of FP division using a single multiplier was used for the ALU. This approach would result in an extra clock cycle for computation when compared to the two multiplier approach.

The multiplier is the biggest and the most important block that is used in FP division. It can be seen that the multiplier is used in every clock cycle. The complexity is with respect to choice of inputs that are given to multiplier. This required a control logic which chooses the inputs to the multiplier for corresponding clock cycles.

The adder is active only for two clock cycles because of gated clocks. Normalization is performed during the addition cycle itself. This is different from the FMA approach. The number of bits for left shift/normalization is fixed and this eliminated the need for a leading zero counter in the case of the FP division. The trigger for normalization is derived from the gated clocks to perform 34 bit left shift within the same cycle.

Inputs selection to the multiplier

The inputs to the multiplier are fed using three rows of D registers. The values stored using the D registers are updated for appropriate clock cycles as illustrated in

Table **7-1**. Table 7-2 illustrates the row numbers that are used for specific clock cycles. A select signal is used to control the input to the multiplier. Figure 7-2 shows the input selection for a single bit. This is replicated for all the bits of the multiplier input.

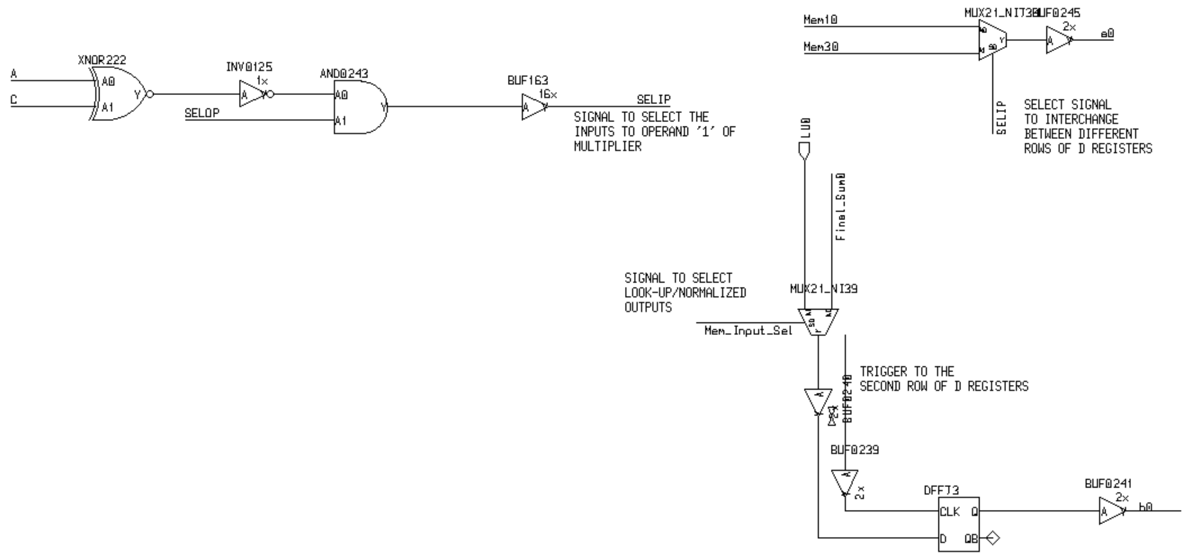


Figure 7-2 : Schematic of inputs to the multiplier for a single bit

ROW OF MEMORY	CYCLE 1 (000)	CYCLE 2 (001)	CYCLE 3 (010)	CYCLE 4 (011)	CYCLE 5 (100)
Row1	B	BX_0	BX_0	BX_0 $(2-BX_0)$	$BX_0(2-BX_0)$
Row2	X_0	X_0	$2-BX_0$	$2-BX_0$	$2-BX_0(2-BX_0)$
Row3	A	A	AX_0	AX_0	$AX_0(2-BX_0)$

Table 7-1 : Table of inputs to multiplier

COUNTER VALUE (CBA)	VALUE OF ROW NUMBER TO OPERAND ‘1’ OF MULTIPLIER	VALUE OF ROW NUMBER TO OPERAND ‘2’ OF MULTIPLIER
Cycle 1 (000)	1	2
Cycle 2(001)	3	2
Cycle 3(010)	1	2
Cycle 4(011)	3	2
Cycle 5(100)	3	2

Table 7-2 : Table of rows of D registers that are used at appropriate clock cycles

Cycle based working of FP division

As mentioned at the start of this chapter, the working of FP Division is explained from a clock cycle point of view. The entire operation of FP division takes six clock cycles to complete the computation.

First clock cycle of operation

The first clock cycle deals with the multiplication of the divisor and its inverse approximated value. The components that are used for the first clock cycle are

a) Multiplier
b) Shifter
c) Exponent computation of Multiplier

Multiplier

The significands of divisor and the inverse approximation are fed as inputs to the multiplier as shown in Table 7-1. It is assumed that there would be a complete clock cycle for reading the initial approximation from memory before the actual computation of the FP division. The Wallace tree multiplier computes the sum and carry. Usually, the FMA computes the addition by using 3:2 compressors for the next stage. Yet, the output of the multiplier is required for computation during the third clock cycle. So, this value of output is stored in the Row1 of D registers as soon as the counter values reach 001 as described in Figure 5-6 of chapter five. This case occurs if 'SELOP' is '1' indicating that FP division is the operational mode of the ALU.

Shifter

The second cycle of operation consists of two's complement subtraction ($2-BX_0$). The FMA is used to shift the value '2' to the appropriate distance based on the FMA shift distance formula given by $d = E_A + E_B - B + m + 3 - E_C$.

In the case of FP division, the value of exponent E_C is represented as 10000. The multiplication of any FP number and it's reciprocal in normalized form produces an exponent result of 1. The actual shift in the case of FP division becomes 12 because all the numbers are constant irrespective of the divisor used. The 34 bit shifter is hardwired with 12 as the shift input when 'SELOP' is '1' indicating a FP division mode of operation. The 34 bit right shifter described in chapter four was implemented using the multiplexer from the standard cell library.

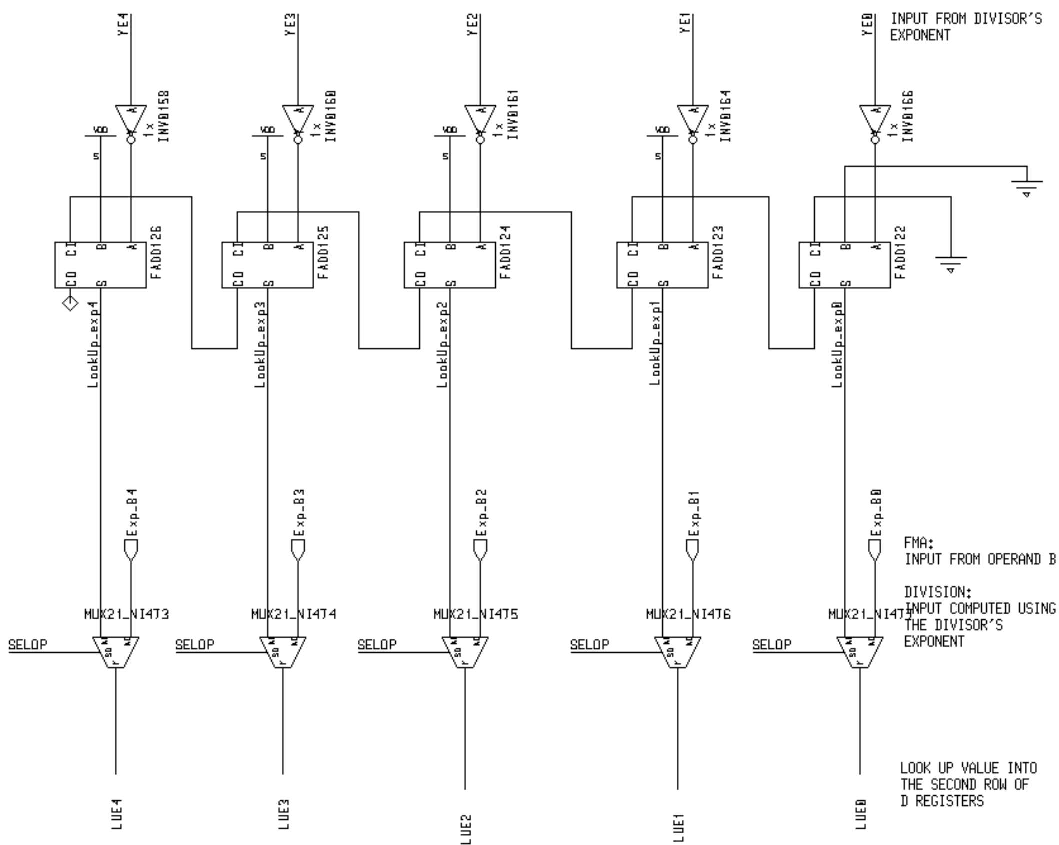


Figure 7-3 : Schematic of inverse approximation's exponent calculation

Exponent Computation of inverse approximation of divisor

The exponent of the inverse approximated value was computed from the exponent of the divisor.

Calculation of exponent of inverse approximation $=\overline{Divisor\ Exponent} - 00010$.

Figure 7-3 shows the computation using two's complement adder and the value is used as the exponent of the inverse approximation of the divisor. In other words, the look-up table is only for the storage of the significand of the numbers. The inputs to the row1 and row3 are controlled by 'INITIAL' signal and the row 2 of D register is

controlled by ‘MEM_INPUT_SEL’. ‘MEM_INPUT_SEL’ is used to maintain the initial approximated value(X_0) for two consecutive clock cycles as shown Table 7-1.

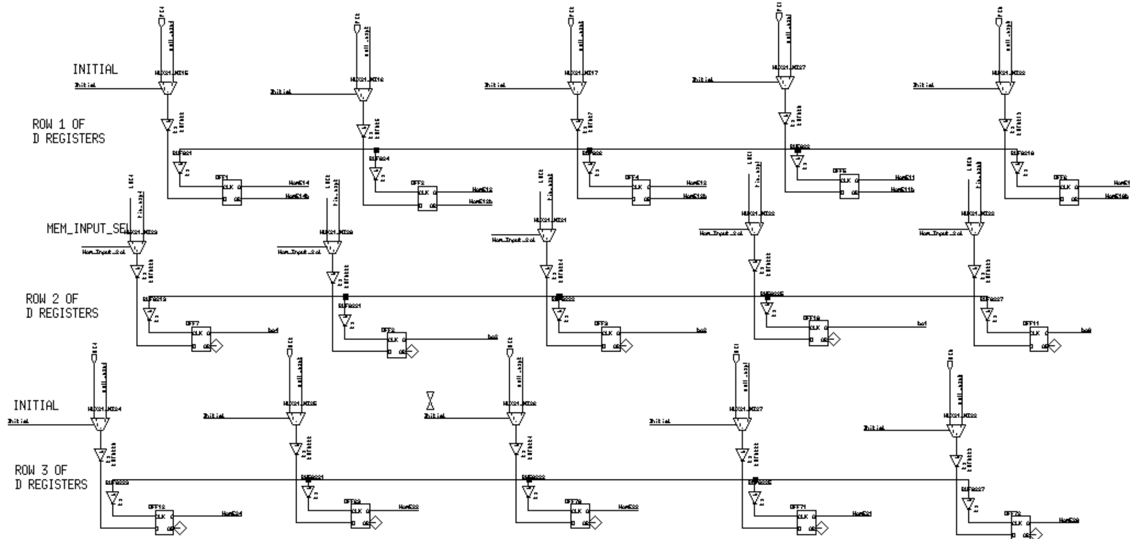


Figure 7-4 : Schematic of exponent part in the rows of D registers

Exponent Computation of the multiplier

The exponent computation of the multiplier is necessary for further computations and it is stored in the D registers. This calculation is also required in the final exponent computation as in the case of FMA. As soon as the counter changes to 001, the output from the multiplier is stored in the row 1 of D registers.

Second clock cycle of operation

The second clock cycle deals with the multiplication of ‘A’ and the initial approximated value ‘X_0’. The computation of $2\text{-}BX_0$ is also done during this clock cycle. This is analogous to the pipelined architecture of hardware components.

The components used in this cycle are

a) Multiplier

b) 35 bit Adder

c) 34 bit shifter

d) Exponent Calculation of $2\text{-}BX_0$

Multiplier

The significands of 'X_0 ' and 'A' are fed as inputs to multiplier. These values are from the second and the third row of D registers. The computed value of AX_0 is fed into the third row of D register when the counter value changes to 010.

35 bit adder

The computation of $2\text{-}BX_0 0$ is done during the second clock cycle. The value of B X_0 was obtained from the multiplier. The 35 bit adder of the FMA is used for two's complement subtraction. The inputs to the adder are exactly the same as that of FMA.

Leading Zero Count

There is a slight change in the calculation of leading zero count when compared to the FMA. In the case of the FMA, the leading zeros are counted using a leading zero counter. In the case of FP division, this is unnecessary because there are either 13 or 14 leading zeros. This is due to fixed shift of 12 which took place in the first clock cycle.

34 bit left shifter

At the positive edge of LZCCLK, the output of the 35 bit adder is normalized using a 34 bit shifter. There might be case where there is a leading zero in the normalized output. This might require a one bit left shift and a corresponding increment of final

exponent by 1. This final left shift should occur only for FP division and is controlled using the control signal shown in Figure 7-5. In case, the MSB of the 34 bit shifter is non zero, there is no need for a one bit left shift. As soon as the counter value changes to 010, this value is stored in the second row D registers.

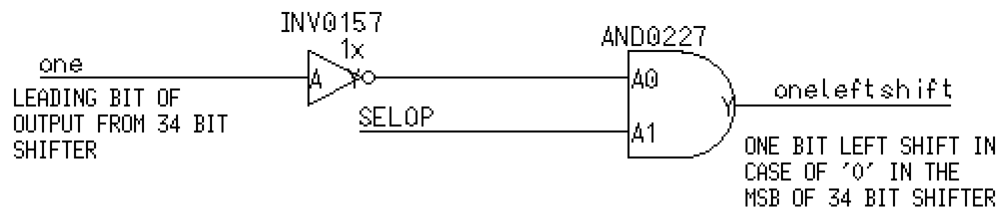


Figure 7-5 : Schematic of control signal for one bit left shift

Exponent Calculation of $2\text{-}BX_0$

Figure 4-22 of chapter four shows the procedure for calculating the exponent of the FMA. The one bit left shift necessitates an increment of the calculated exponent by 1 as shown in Figure 7-6. In case, there is no one bit left shift, the exponent computed using the FMA is the final exponent. As soon as the counter changes to 010, this value is stored in the second row of D registers.

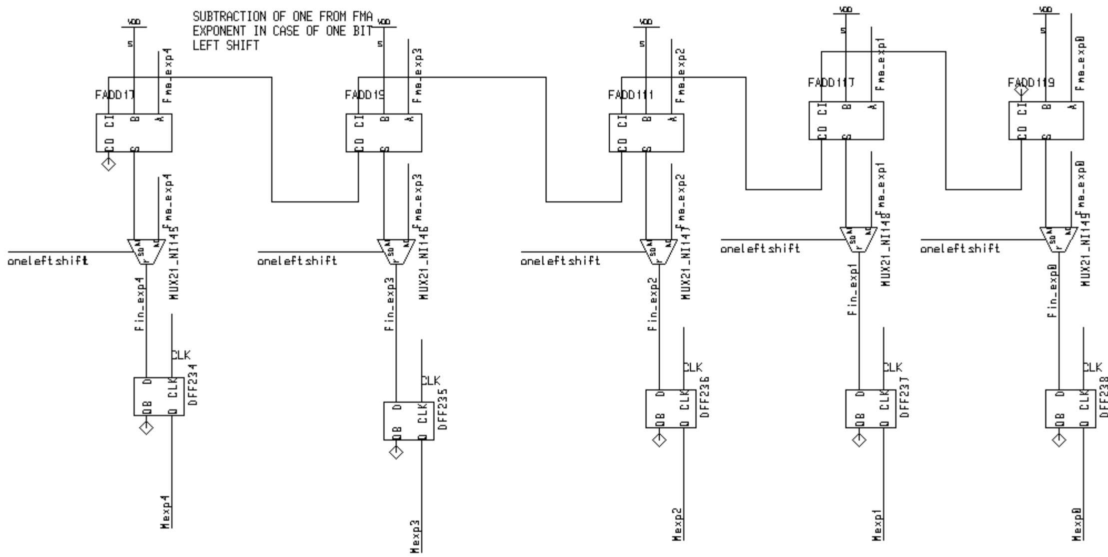


Figure 7-6: Schematic of update of exponent for one bit left shift

<u>Third clock cycle of operation</u>

The third clock cycle of operation deals with the computation of $BX_0.(2 - BX_0)$. BX_0 is available in row 1 of D registers and $(2 - BX_0)$ is available in row 2 of D registers. 'SELIP' signal is set to 0 which indicates that the inputs to the multiplier are available from the first row of D registers.

The adder waits for the output from the multiplier to compute the next set of operations. This multiplication is also very similar to previously explained techniques. The tricky aspect of FP division is the presence of many control signals when compared to the FMA. As soon as the counter value changes to 011, multiplier output is stored into the first row of D registers. The exponent of multiplier is updated in parallel.

Fourth clock cycle of operation

Addition and Normalization

The fourth clock cycle of operation consists of the computation of $(2 - BX_0 (2- BX_0))$. This is performed by means of the FMA adder as already explained in the second clock cycle of operation. Again, the clocks are gated only when counter values are '001' and '011'. At the end of clock cycle, the final result might need a one left shift similar to case mentioned in the second clock cycle. As soon as the counter changes to 100, the output of the shifter is stored in the second row of D registers.

Multiplication

The multiplication of $AX_0. (2-BX_0)$ is computed during the fourth clock cycle. As soon as the counter changes to 100, the output of the multiplier is stored in the third row of the D registers.

Fifth clock cycle of operation

The fifth clock cycle computes the multiplication of $AX_0.(2-BX_0) .(2- BX_0(2- BX_0)$. The output of this operation gives the quotient of the FP division in an unrounded form.

Sixth Clock cycle of operation

The final step in the FP division involves the rounding of the output of the multiplier. Rounding to nearest was performed using the 11 bit adder. The details of rounding logic were provided in chapter four. The rounding was performed only once after the completion all computations. This was possible with the help of a gated round

clock as shown in Figure 5-5. The output of rounding logic is the result of A/B in normalized FP format.

Conclusion

This chapter dealt with the implementation of FP division. The quotient is obtained at the start of the seventh clock cycle for normal FP division. FP division is the most complex operation in the ALU. The number of operations in a pipeline increased the complexity. The inputs to the multiplier were manipulated at appropriate clock cycles. The rest of the operation is similar to the FMA. The quotient is obtained immediately in case of an overflow or an underflow. Chapter eight gives a specific example of an overflow in FP division.

In special cases of an exception, the values are not stored in memory and the entire state of the ALU is maintained. These are controlled by the signals that activate the memory. The correct significand and the exponent values are also shown as the output. The next chapter deals with the results of the ALU obtained from IC station of Mentor Graphics.

CHAPTER EIGHT

RESULTS AND CONCLUSION

This chapter presents the results of the ALU and a comparison of the 4:2 compressors using full adder and pass gate multiplexer. There are two test cases which show the handling of an FP exception.

4:2 Compressor

The 4:2 compressor built using the pass gate multiplexer was found to be 7 times faster than the 4:2 compressor built using full adder. The performances of layout of these compressors were compared. The output of the 4:2 compressor is fed to a buffer in the actual multiplier circuit. While testing, the input capacitance of the buffer was used as the load for the 4:2 compressors.

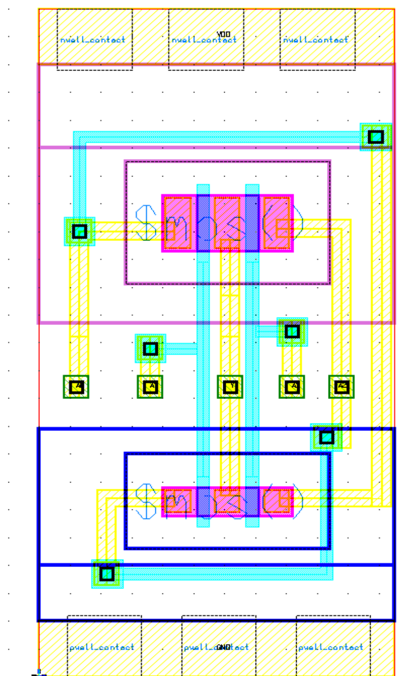


Figure 8-1 : Layout of pass gate multiplexer

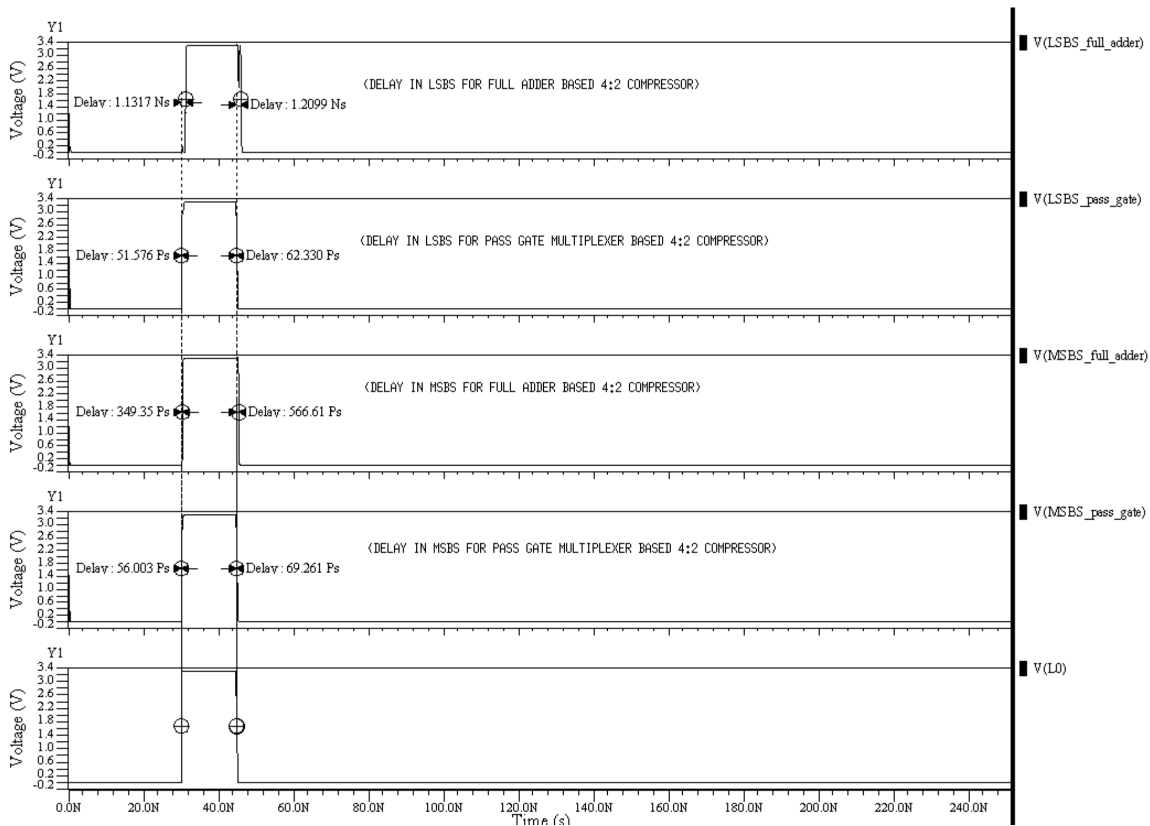


Figure 8-2 : Waveform showing the delay comparison

The possible reasons might be the difference in number of logic levels between the 4:2 compressors. Another reason is the sizing of full adder in the standard cell, number of transistors and the increased diffusion area causing increased parasitic capacitances.

Results of FMA

Case 1 ($A * B + C$)

The following test involves the computation of $A * B + C$

A: $1.912 * 2^{-2}$; B: $1.607 * 2^{-6}$;C: $1.332 * 2^{0}$

First clock cycle

Cycle 1 deals with the computation of $A * B$ and the shift of the addend as shown in Figure 8-3. The shift value that was computed is shown in the waveform as ‘Shift’ and the output of the multiplier is expressed as ‘Multiplier_sig’ and ‘Multiplier_exp’.

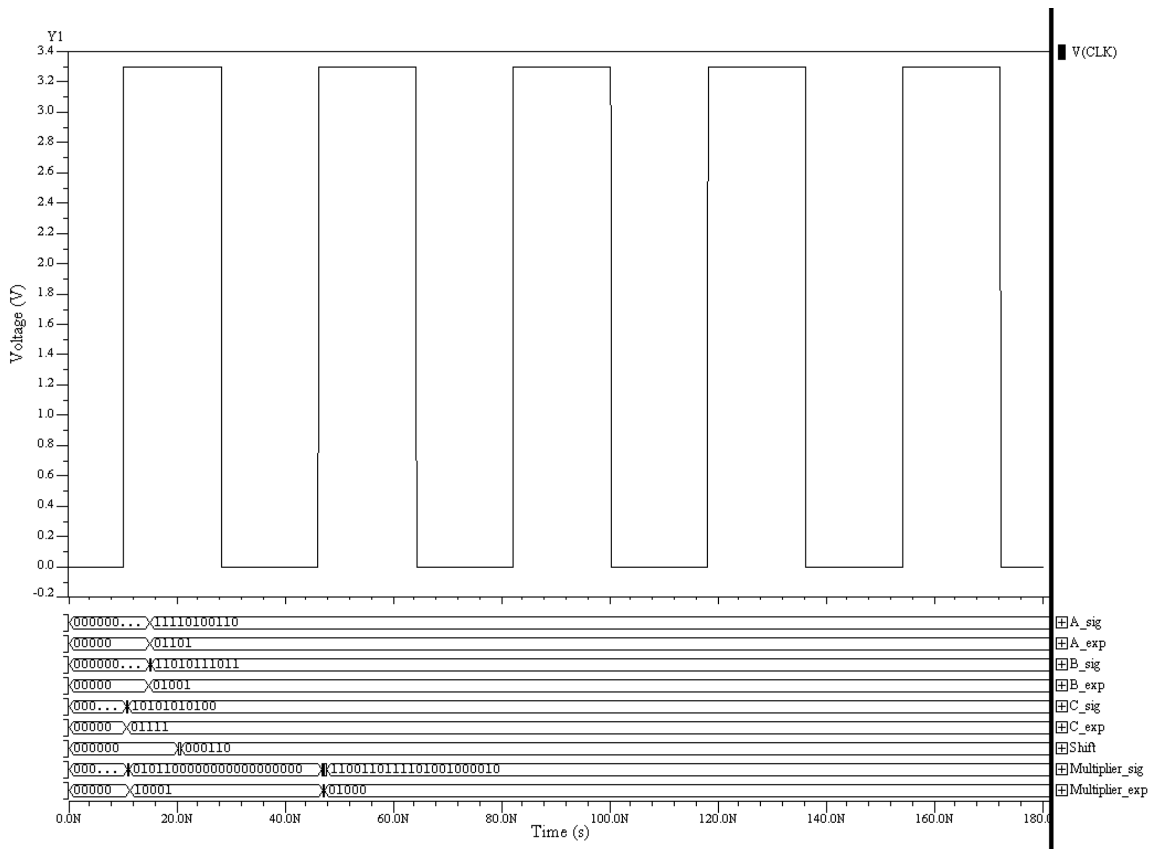


Figure 8-3 : Waveform of first clock cycle

Second clock cycle

The second clock cycle of operation deals with the addition of $A * B + C$ using a 35 bit adder. Clock gating was used to produce clocks for two additions and a leading zero counter. The waveform also shows control signal which controls the number of

additions that are required. For e.g., the signal's value would be '0' at the start of the clock cycle. In the case of 2's complement subtraction, another addition that is required. This is taken care by control logic that generates a signal shown in Figure 8-4. The delay of the adder circuit is 4.96ns. Leading zero counter is activated once the addition is completed and it computes the number of leading zeros in the adder result. This value is fed into the normalization logic at the start of third cycle.

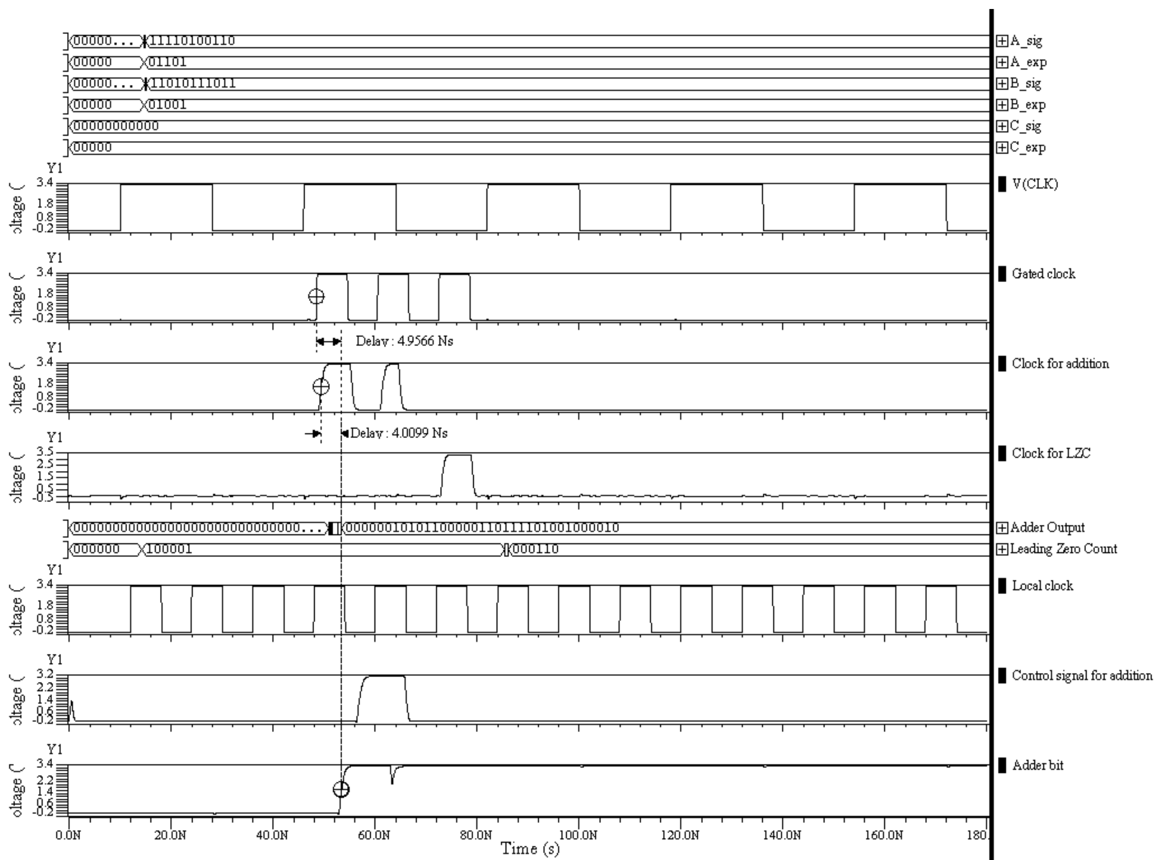


Figure 8-4 : Waveform of the second clock cycle

Third clock cycle

The third clock cycle deals with the actual left shift or normalization. Rounding to nearest was also performed for the result of the normalized output. The final result is

given by ‘Final_sig’ and ‘Final_exp’ in the waveform. The final result should be $1.344 * 2^0$.

INPUTS/OUTPUT	BINARY	DECIMAL
A	0 01101 1110100110	$1.912 * 2^{-2}$
B	0 01001 1010111011	$1.607 * 2^{-6}$
C	0 01111 0101010100	$1.332 * 2^{0}$
Final Result	0 01111 0101100001	$1.344 * 2^{0}$

Table 8-1 : Table of results for $A \; X \; B + C$

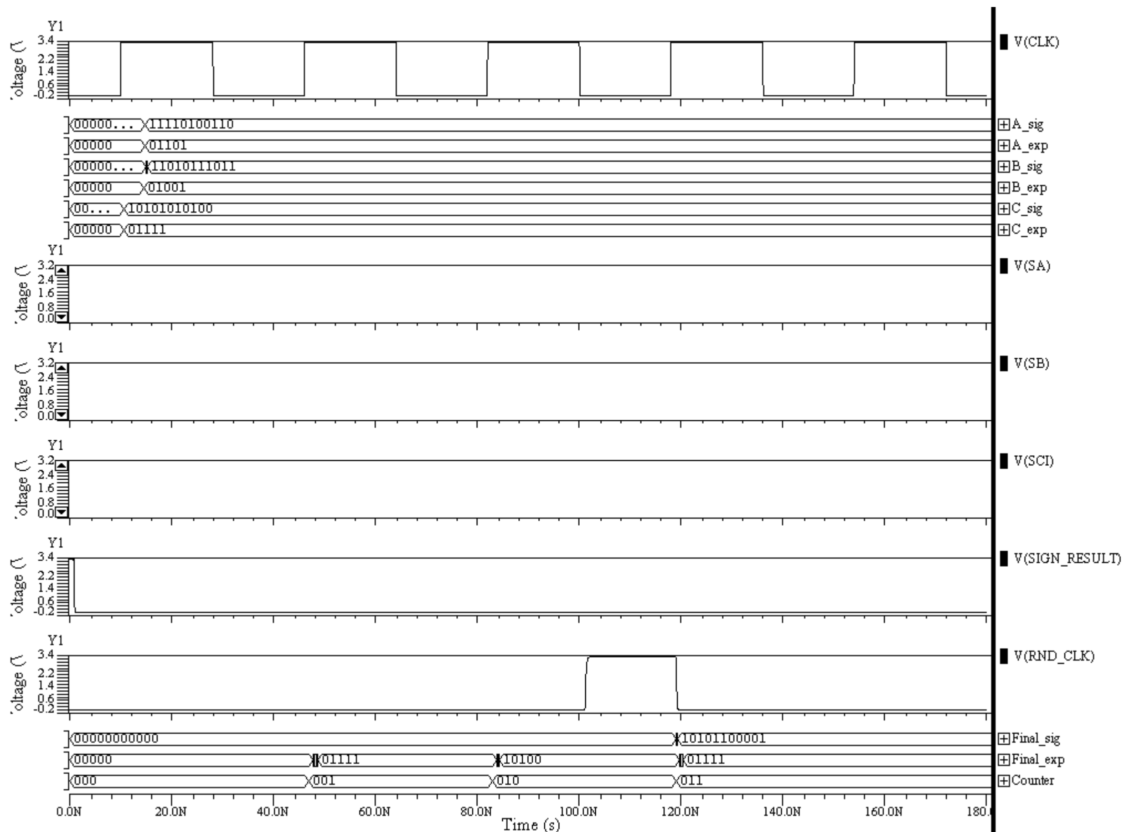


Figure 8-5 : Waveform of the third clock cycle

Case 2 ($(-A*-B)-C$)

The following test involves the computation of $(-A * -B) - C$

A: $-1.912 * 2^{-2}$; B: $-1.682 * 2^{-10}$;C:$-1.332 * 2^{4}$.

The result is $-1.332 * 2^{4}$ which is obtained at the start of the fourth clock cycle. All values are clocked at the positive edge of clock. The answer is given by ‘Final_sig’ and ‘Final_exp’.

INPUTS/OUTPUT	BINARY	DECIMAL
A	1 01101 1110100110	$-1.912 * 2^{-2}$
B	1 00101 1010111011	$-1.682 * 2^{-10}$
C	1 10011 0101010100	$-1.332 * 2^{4}$
Final Result	1 10011 0101010100	$-1.332 * 2^{4}$

Table 8-2 : Table of results for $(-A\ X -B) - C)$

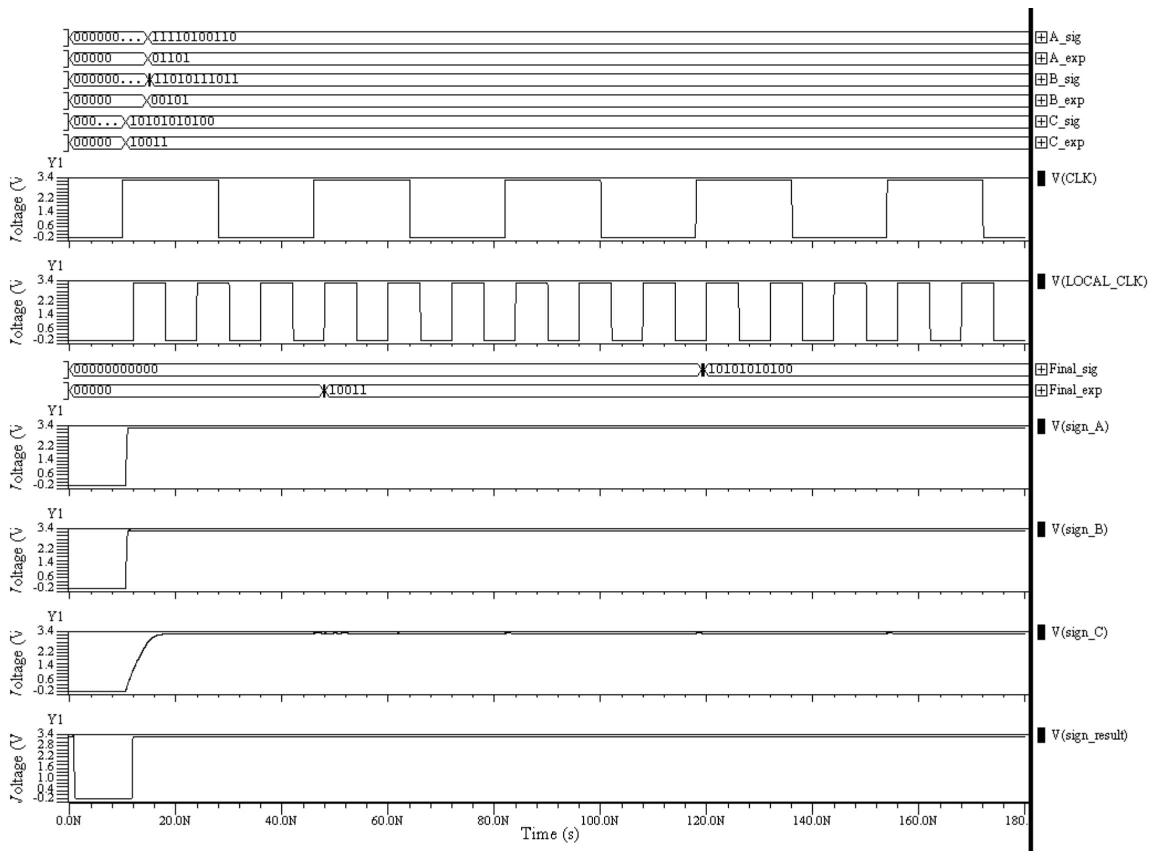


Figure 8-6 : Waveform of $(-A\,X - B - C)$

Result of FP Division

The following test case involves the computation of A/B

$A = 1.044 * 2^{6}$; $B = -\ 1.633 * 2^{1}$; $X_0 = 1.234 * 2^{-2}$

The first cycle of operation computes BX_0 and stores the value in memory. The second cycle computes $2 - BX_0$ and AX_0 and stores these values in memory. The third cycle computes BX_0 (2- BX_0). The fourth clock cycle uses the previous value and computes $(2 - BX_0(2 - BX_0))$. This cycle also involves the computation of $AX_0(2 - BX_0)$.

These values are also stored in memory. The last cycle computes $AX_0(2-BX_0)(2-BX_0(2-BX_0))$.

INPUTS/OUTPUT	BINARY	DECIMAL
A	0 10101 0000101110	$1.044 * 2^{6}$
B	1 10000 0011101111	$-1.633 * 2^{1}$
X_0	0 01101 1010001001	$1.234 * 2^{-2}$
Final Result	1 10011 0100011101	$-1.278 * 2^{4}$

Table 8-3 : Table of results for FP division

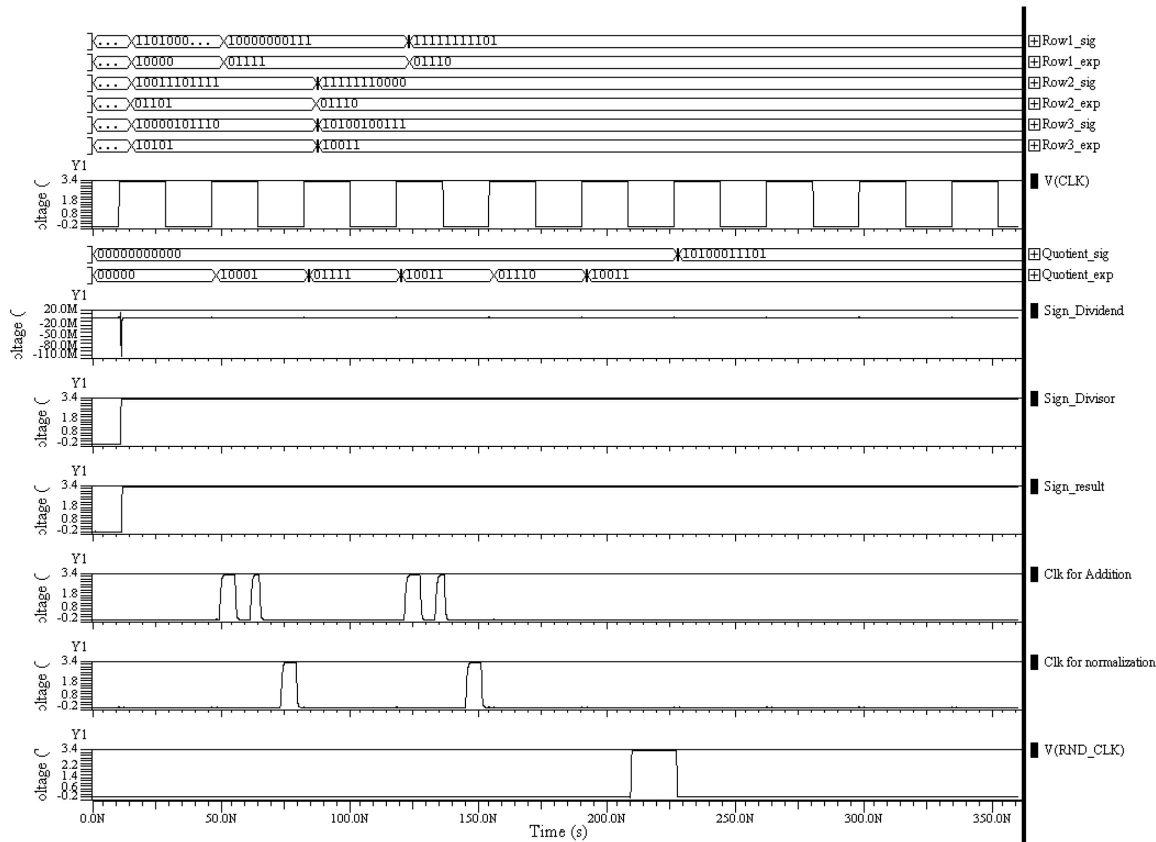


Figure 8-7 : Waveform of FP division

FP exceptions

The underflow and overflow exceptions were handled as part of this thesis. ‘Underflow_Exception’ and ‘Overflow_Exception’ are the signals that indicate exception and also make corresponding changes to the significand and the exponent.

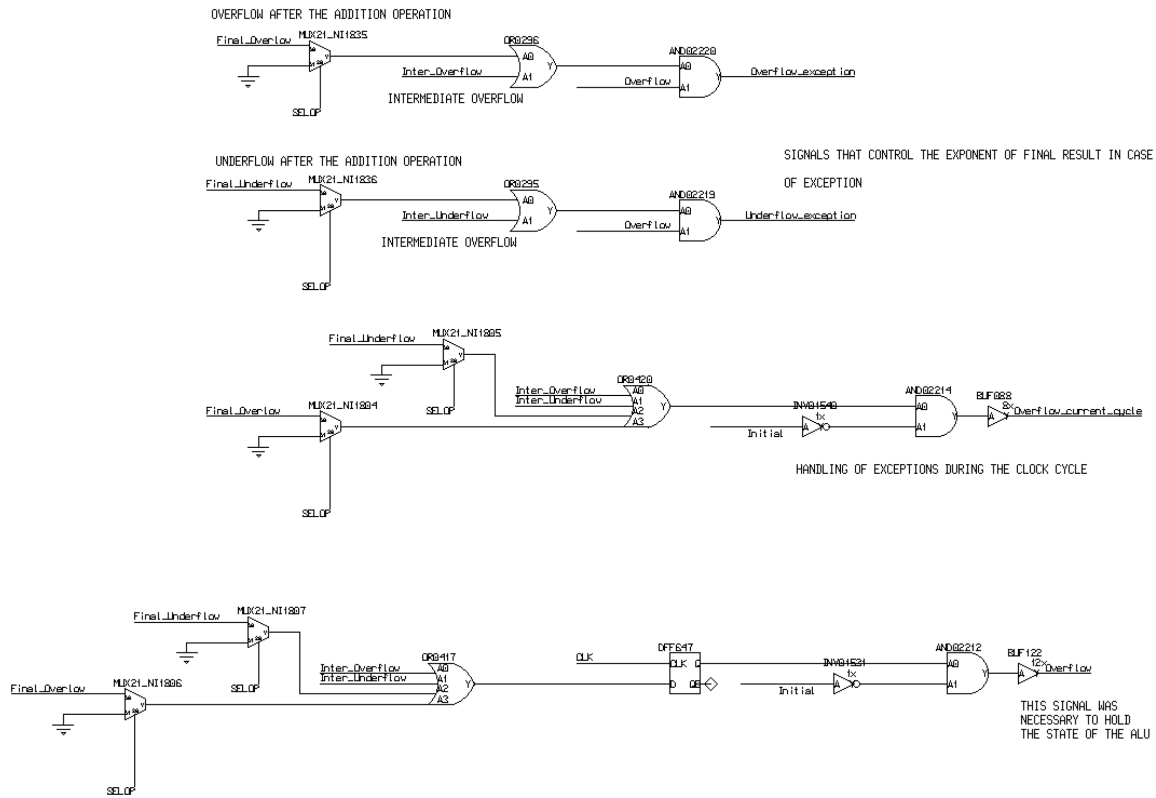


Figure 8-8 : Schematic of circuits that handle FP exceptions

The entire ALU is held in the same state of operation if an exception occurs. The clocks that control the various components of the ALU are gated by the exception flag. As a result, the different components cannot be activated. Figure 8-8 and Figure 8-9 shows the circuits that handle such situations. The final result is manipulated to represent such exceptions. The overflow and underflow would make the significand of the output as ‘0’. The exponent is given as 11111 or 00000 to represent $infinity$ and zero.

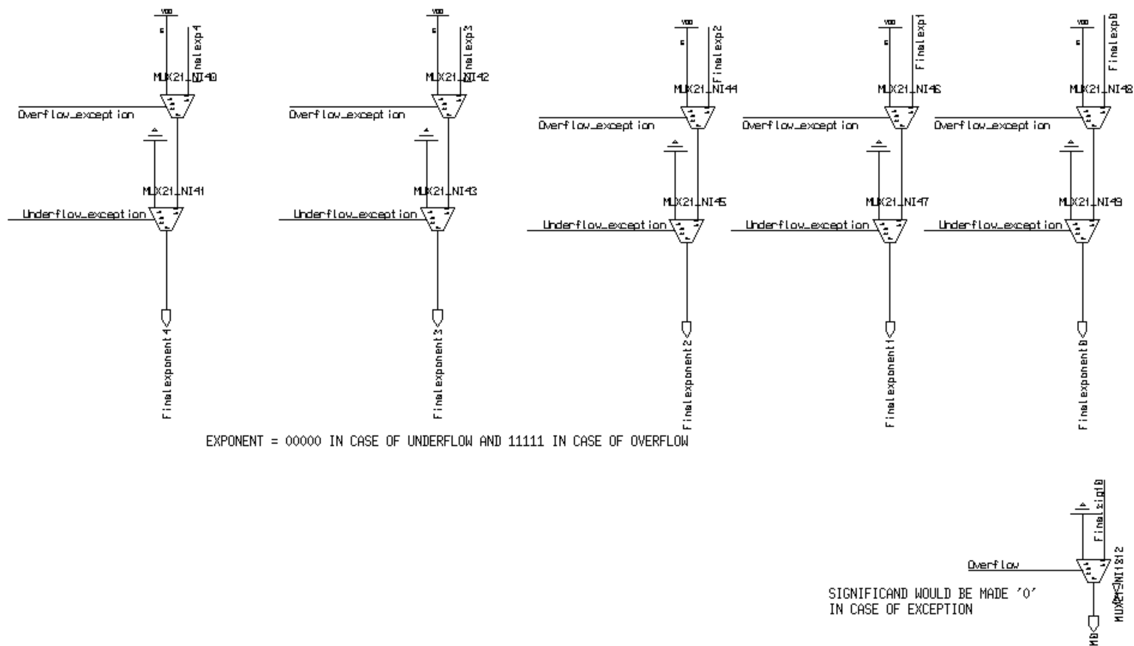


Figure 8-9 : Schematic of update of exponent and significand in case of FP exception

<u>FP Underflow</u>

The underflow exception results when the FP number is too small to be represented by a normalized FP number. A flush to zero was performed to denote such small numbers. This was performed for both FMA and FP division. A case of underflow for FMA is shown in Table 8-4.

INPUTS/OUTPUT	BINARY	DECIMAL
A	0 00001 1110100110	$1.912 * 2^{-14}$
B	1 00011 1010111011	$-1.682 * 2^{-13}$
X_0	0 01010 0101010100	$1.332 * 2^{-5}$
Final Result	1 00000 0000000000	-0

Table 8-4 : Table of results of underflow in FMA

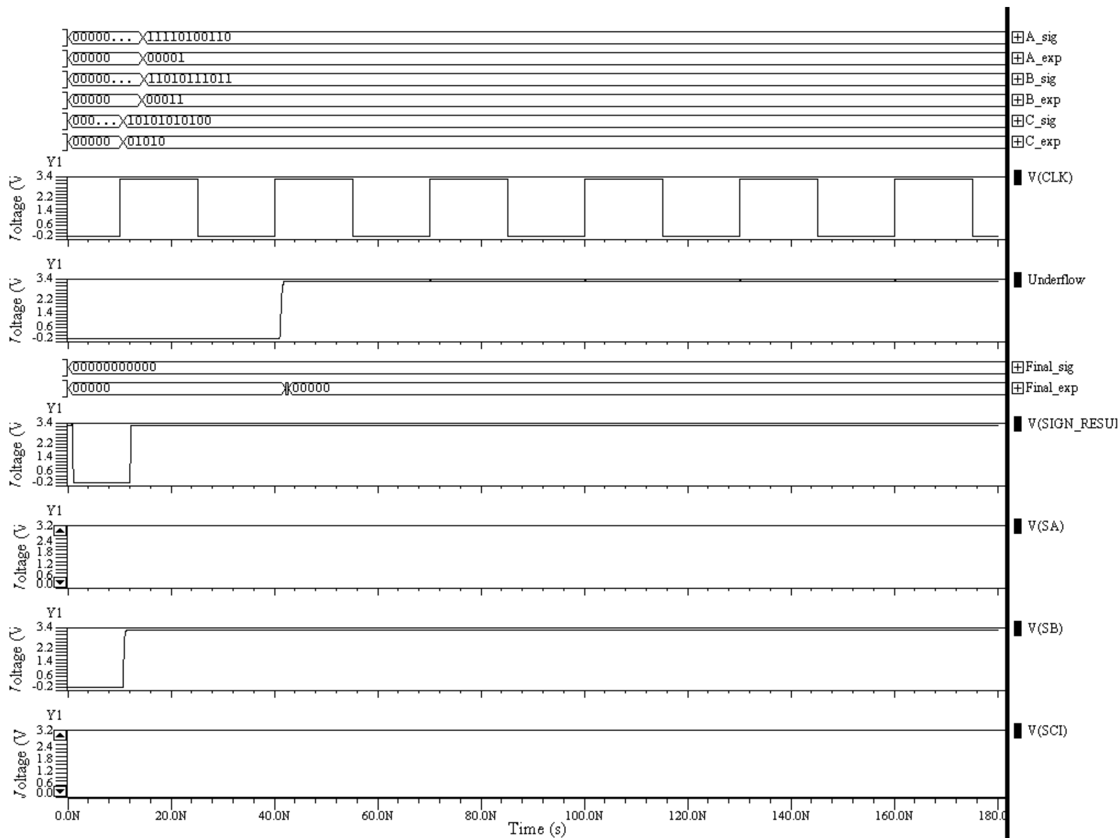


Figure 8-10 : Waveform of underflow exception in FMA

FP Overflow

The overflow exception occurs when the number is too large to be represented as a normalized FP number. The output is shown as ±infinity based on the sign of the result. A test case of FP overflow is shown in Table 8-5 and Figure 8-11 .

INPUTS/OUTPUT	BINARY	DECIMAL
A	0 11010 1010111000	$1.679 * 2^{11}$
B	0 01001 0000100010	$1.046 * 2^{-6}$
X_0	0 10100 1110110101	$1.926 * 2^{5}$
Final Result	0 11111 0000000000	$+ infinity$

Table 8-5 : Table of overflow results in FP division

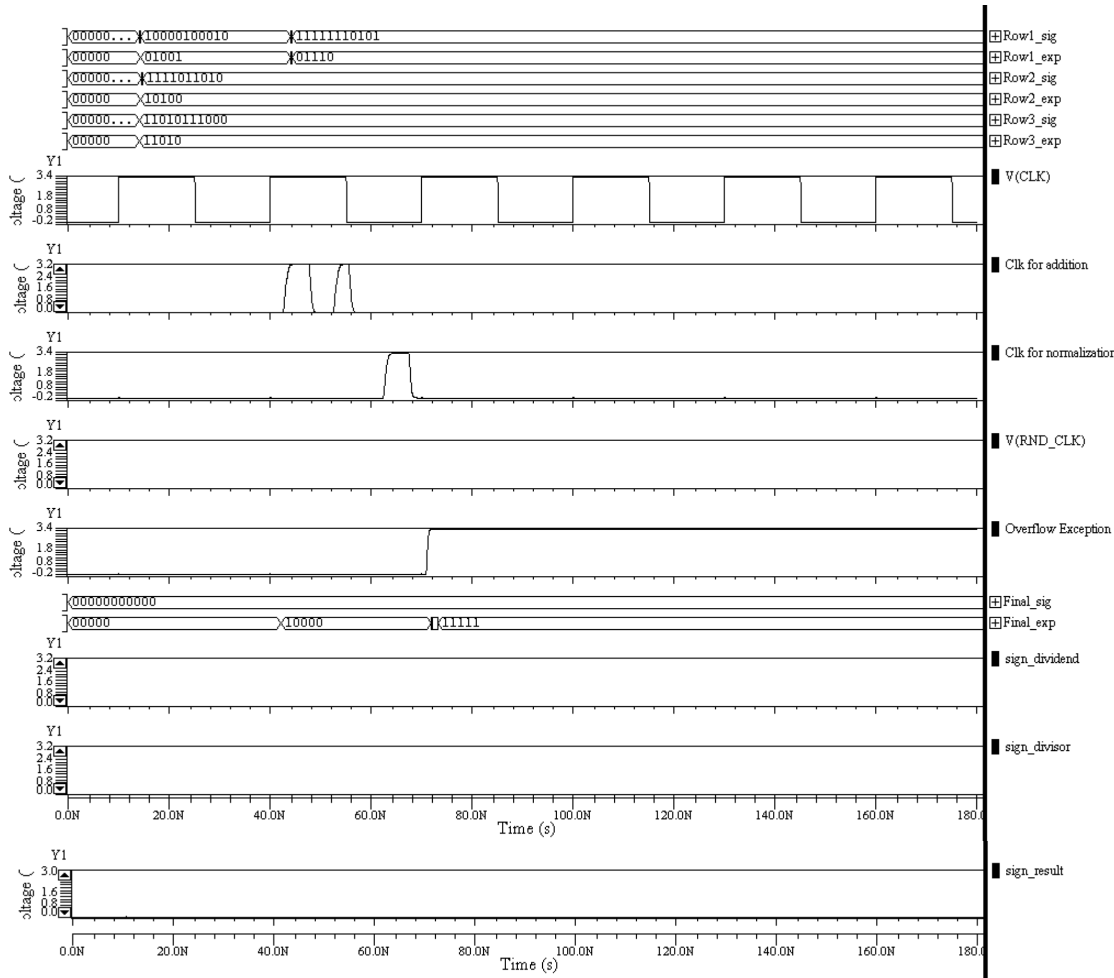


Figure 8-11: Waveform of overflow exception in FP division

Conclusion

The 16 bit FP ALU was designed and implemented successfully. The FMA offered significant advantages when compared to the separate FP adder, FP multiplier and FP divider. The separate FP adder and FP multiplier would have produced a scalar product in four clock cycles and is a very hardware intensive solution. The FMA offered the advantage of combining all the operations in a single hardware. Normal FMA operations compute the result in three clock cycles.

Different architectures were investigated as possible options for the FMA. The reasons for the choice of FMA are given in chapter three. A fast multiplier was implemented with the least possible number of reduction levels. The multiplier is the most complex component and very few changes are possible in the multiplier. The standard cell multiplexer was used in components like the shifter. Looking back at the implementation, the pass gate multiplexer could have been used to improve the speed of operation. It is believed that such an implementation might not entirely result in latency reduction but would have resulted in significant power reductions. It is also believed that the components of the standard cell library are area intensive and power hungry.

The design of the adder is another implementation where an alternate approach could have been used. The architecture of the FMA negated any benefits that could be made possible by implementing with carry look ahead adders. In other words, the architecture defined the way in which the design was implemented. An asynchronous approach is also very difficult for such a complex design.

All the required specifications mentioned in chapter one were met successfully. The material in this thesis could be expanded to include calculations involving denormalized and special numbers. Such an improvement could also help in including exceptions like NaN and division by zero. Another area of improvement is the implementation of the initial look up table for the iterative division algorithm. A fast SRAM based cache memory could have been designed to store the initial look-up value. This thesis consists of a design to prove the functionality of the ALU. Testing methods like BIST, exhaustive test set for all logic blocks could also be included to expand the testing scheme that was currently performed in this thesis. Such improvements would replicate the design of major components used in a micro-processor.

APPENDICES

APPENDIX A

Floating point number system

The IEEE has standardized the representation for binary floating-point numbers as IEEE 754 and this standard is popularly used in modern computers. There are several formats with increasing levels of accuracy. The 16 (Half), 32 (Single), 64 (Double) and 128 (Quadruple) bits are the FP precisions used in IEEE 754.The different formats are explained in this appendix.

IEEE 754 standard specification for binary32/single precision:

SIGN BIT	EXPONENT WIDTH	SIGNIFICAND PRECISION
1	8	24 (23 bits explicitly stored)

Table A-3: Table of 32 bit format for FP numbers

There is one implicit leading bit in the actual significand whose value is '1' except for one condition, when the exponent is stored with all zeros. While representing the number it will have only 23 bits, but the actual precision is 24 bits.

IEEE 754 standard specification for binary64/double precision:

SIGN BIT	EXPONENT WIDTH	SIGNIFICAND PRECISION
1	11	53 (52 bits explicitly stored)

Table A- 4 : Table of 64 bit format for FP numbers

This also contains a single implicit bit. While representing the number it will have only 53 bits, but the actual precision is 54 bits. This binary format uses 8 bytes. As a

generalization, all the inputs to the floating point ALU were considered to normalized with a hidden bit of 1.

Least commonly uses floating point formats

There is an 80 bit floating point representation which is also called as extended precision format/long double/quadruple precision. There are other excessively small floating point formats called as minifloats and microfloats. Although these formats are not used popularly there is certain significance for these formats. Microfloats are floating point format using 8 bits or less which can fit in a byte. This format is very useful for learning and working with floating point's arithmetic using pencil and paper. It can be useful as an educational tool for implementing floating point algorithms.

Minifloat is a floating point format which uses 16 bits or less for the representation. The hardware support for this floating point format is provided by nVidia graphics cards including GeForce FX and Quadro FX 3D and it also used by Industrial Light and Magic company for their OpenEXR standard. Pixar use it as the native format for raw output rendered frames i.e. before converting to a compressed format like DVD, HDTV, or imaging on photographic film for exhibition in a theater. This format is sufficient to represent light levels in a rendered image. When comparing the 32-bit floating-point format with 16-bit format it presents quite a few advantages, it requires only half of the memory space, the addition and multiplication operations takes less than half the time of computation and about 1/4 as many transistors. The computer-graphics field completely utilizes the power of the floating-point format because they use the value of floating-point to represent pixels.

APPENDIX B

Architectures of Fused multiply add

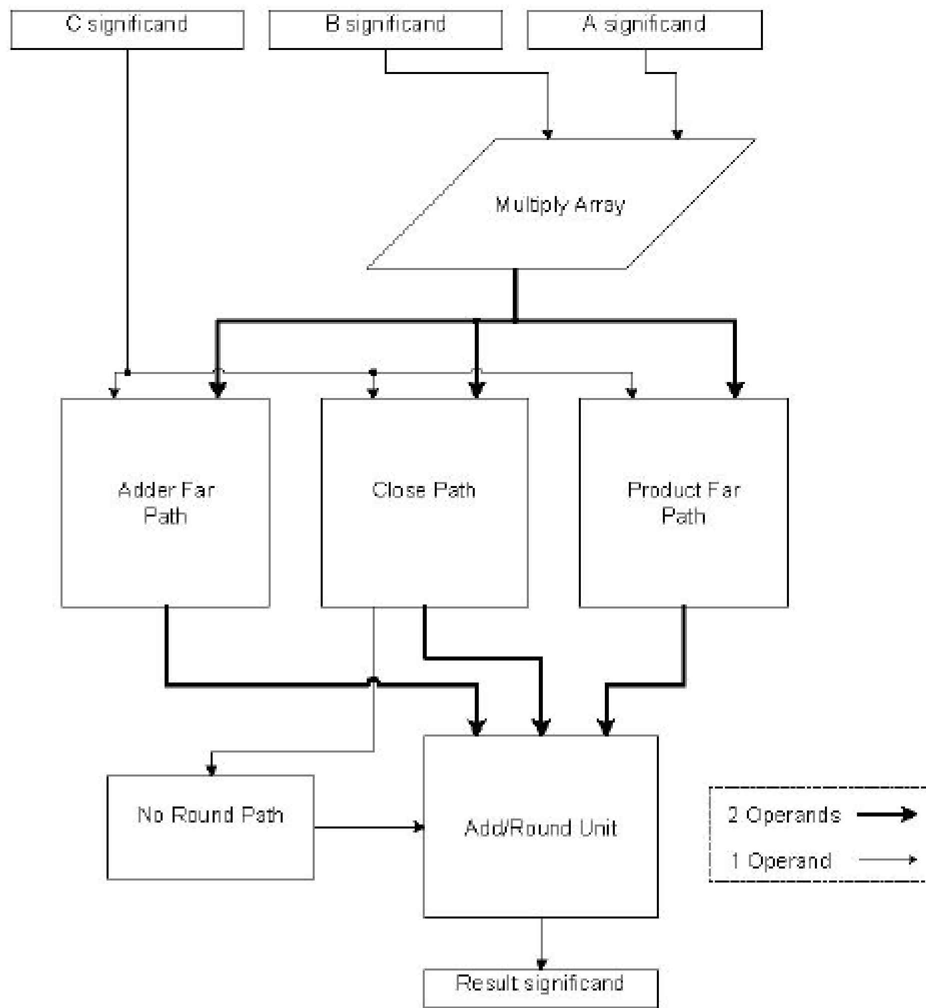


Figure B- 1: Block diagram of three path FMA

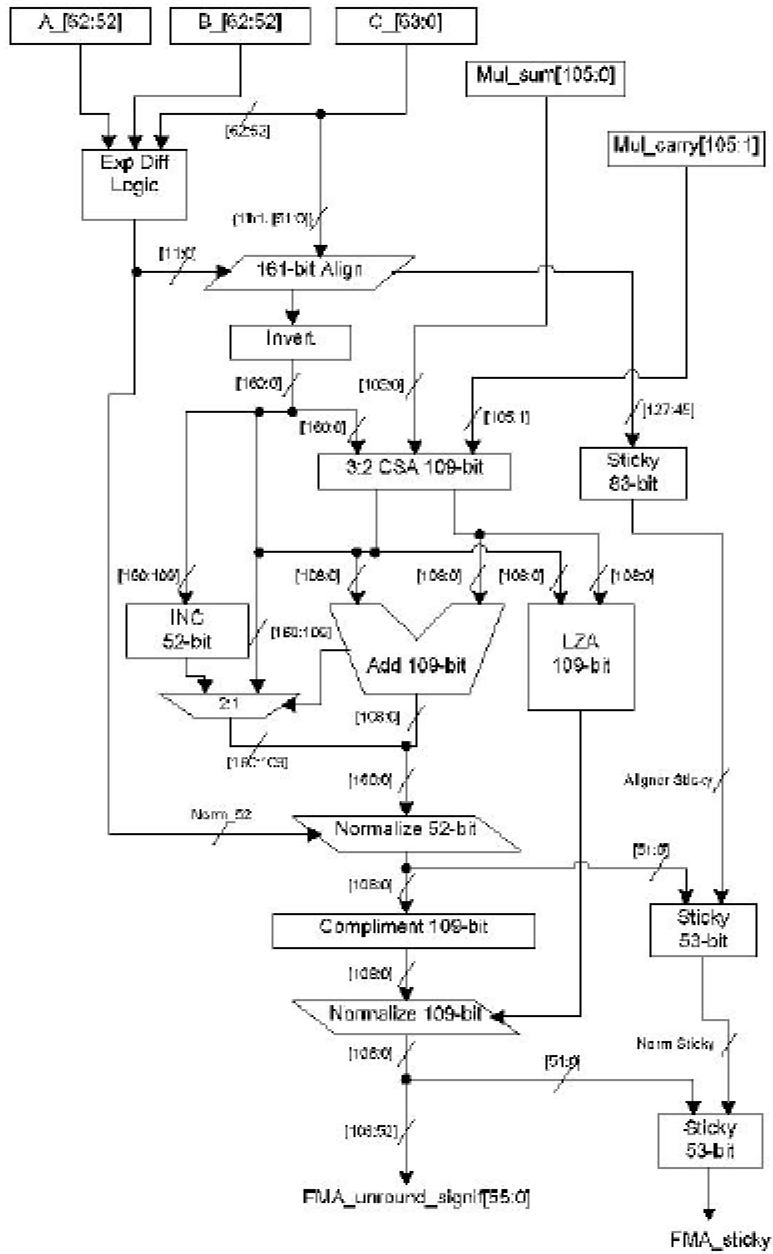


Figure B- 2: Block diagram of bridge FMA

APPENDIX C

Capacitance calculation for load of Multiplier

Calculation of capacitance for load

The next stage after the multiplier block is the D register which triggers the output of the multiplier into the adder block.

Oxide Capacitance

Oxide Capacitance = $C_{ox} = \frac{\varepsilon_o \varepsilon_{ox}}{T_{ox}} = \frac{8.854*10^{-12}*3.9}{7.8*10^{-9}} = 4.427 * 10^{-3} \frac{F}{m^2} = 4.427 \frac{fF}{\mu m^2}$

Overlap capacitance per unit width

$C_{OL} = CGDO + CGSO$, where $CGDO$ and $CGSO$ are the overlap capacitance per unit gate width and are obtained from the model file.

$C_{OL_n} = 2.91 * 10^{-10} + 2.91 * 10^{-10} = .582\ fF/\mu m$

$C_{OL_p} = 2.49 * 10^{-10} + 2.49 * 10^{-10} = .498\ fF/\mu m$

Gate capacitances of D flip flop

The gate capacitance = $C_{Gn} + C_{Gp} + (C_{OL})_n W_n + (C_{OL})_p W_p = 4.427 * .35 * 15 * .2 + 4.427 * .35 * 27 * .2 + .582 * 15 * .2 + .498 * 27 * .2 = 17.45 fF$

A variation of 50% was used as a load for the capacitance at each node of the multiplier.

APPENDIX D

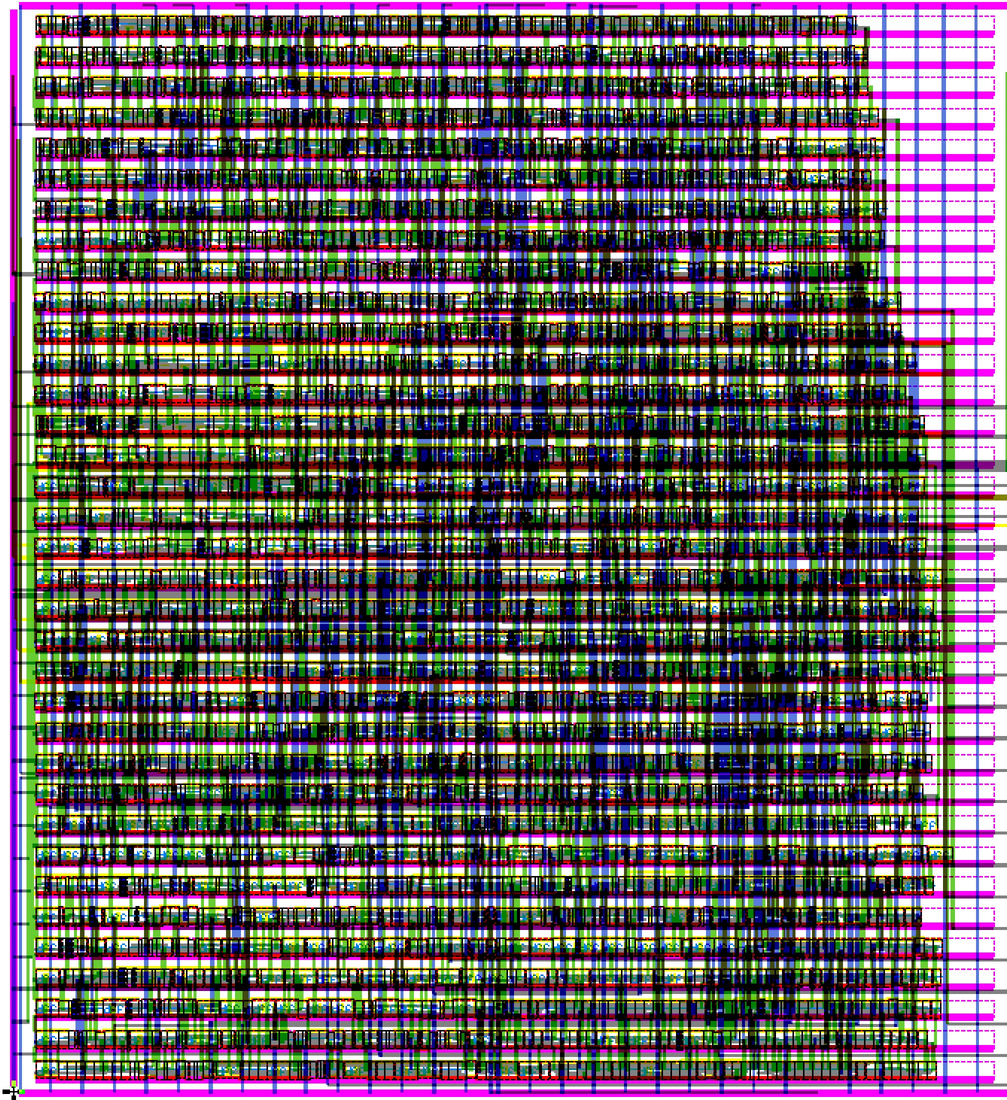


Figure D- 1: Complete layout of ALU

REFERENCES

[1] "IEEE Standard for Floating-Point Arithmetic", Microprocessor Standards Committee of the IEEE Computer Society

[2] Lacassagne et all., "16-bit floating point instructions for embedded multimedia applications, Computer Architecture for Machine Perception", 2005. CAMP 2005. Proceedings. Seventh International Workshop on, p 198- 203, July 2005

[3] Eilert, J. et all, "Using low precision floating point numbers to reduce memory cost for MP3 decoding" in 2004 IEEE 6th Workshop on Multimedia Signal Processing, p 119-22, 2004

[4] Goldberg, David, "What every computer scientist should know about floating-point arithmetic", ACM Computing Surveys, v 23, n 1, p 5-48, Mar 1991

[5] Oehler, R.R. Blasgen, M.W., IBM RISC System/6000: architecture and performance, Volume: 11, Issue: 3, Jun 1991

[6] Quinnell, E.; Swartzlander, E.E.et all." Floating-point fused multiply-add architectures" in 2007 41st Asilomar Conference on Signals, Systems and Computers (ACSSC '07), p 331-7, 2008

[7] P. Montuschi, L. Ciminiera, A. Giustina, "Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient", IEE Proceedings - Computers and Digital Techniques -- November 1994 -- Volume 141, Issue 6, p. 317-324

[8] C. S. Wallace, "A Suggestion for a Fast Multiplier", Vol. 13, No. 14, IEEE Transactions on Electronic Computers (Feb. 1964), pp. 14-17.

[9] Bickerstaff, K.A.C. (Dept. of Electr. & Comput. Eng., Texas Univ., Austin, TX, USA); Schulte, M.; Swartzlander, E.E., Jr. , "Reduced area multipliers", Source: Proceedings. International Conference on Application-Specific Array Processors (Cat. No.93TH0572-8), p 478-89, 1993

[10] Ohkubo, N. (Central Res. Lab., Hitachi Ltd., Tokyo, Japan); Suzuki, M.; Shinbo, T.; Yamanaka, T.; Shimizu, A.; Sasaki, K.; Nakagome, Y., "A 4.4 ns CMOS 54×54-b multiplier using pass-transistor multiplexer", Source: IEEE Journal of Solid-State Circuits, v 30, n 3, p 251-7, March 1995

[11] Pillmeier, M.R. (Rushmore Processor 2, Unisys Corp., Blue Bell, PA, USA); Schulte, M.J.; Walters, E.G., II, "Design alternatives for barrel shifters", Source: Proceedings of the SPIE - The International Society for Optical Engineering, v 4791, p 436-47, 2002

[12] Beaumont-Smith, A.Burgess, N. et all, "Reduced latency IEEE floating-point standard adder architectures" in Proceedings - Symposium on Computer Arithmetic, p 35-42, 1999

[13] Seidel, P.-M. ;Even,G et all, "On the design of fast IEEE floating-point adders"in Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001, p 184-84, 2001

[14] O. J. Bedrij, “Carry-Select Adder”, IRE Transactions on Electronic Computers, p. 340-344, 1962.

[15] A. Weinberger and J. L. Smith, “A Logic for High-Speed Addition”, National Bureau of Standards, Circ. 591, pp. 3-12, 1958.

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

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

[18] T. D. Han and D. A. Carlson, “Fast Area-Efficient VLSI Adders,” 8th symposium on Computer Arithmetic, May 1987.

[19] R. Ladner and M. Fischer, "Parallel prefix computation", J. ACM, vol.27, no 4, Oct. 1980, pp 831-838

[20] Knowles, S. , "A family of adders"in Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. \No.99CB36336), p 30-4, 1999

[21] J. Sklansky, "Conditional Sum Addition Logic," IRE Trans. on Electronic Computers, vol. EC-9, pp. 226--231 (June 1960).

[22] Farooqui, A.A. Oklobdzija, "Multiplexer based adder for media signal processing", LSI Syst. Lab., Sony US Res. Lab., San Jose, CA ;

[23] V. G. Oklobdzija, B. R. Zeydel, H. Dao, S. Mathew, R. Krishnamurthy, “Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders”, Proceedings of the International Symposium on Computer Arithmetic, ARITH-16, Santiago de Compostela, SPAIN, June 15-18, 2003.

[24] Vojin G. Oklobdzija ,Earl R. Barnes "On implementing addition in VLSI technology", Journal of Parallel and Distributed Computing archive Volume 5 , Issue 6 (December 1988) pp: 716 - 728, 1988

[25] A.Th. Schwarzbacher1, J.P. Silvennoinen1,2 and J.T.Timoney3,"Benchmarking CMOS Adder Structures".

[26] V. G. Oklobdzija and E. R. Barnes, "Some Optimal Schemes For ALU Implementation in VLSI Technology", Proceedings of the 7th Symposium on Computer Arithmetic ARITH-7, pp. 2-8, Reprinted in "Computer Arithmetic", E. E. Swartzlander, (editor), Vol. II, pp. 137-142, 1985.

[27] [1] V. G. Oklobdzija and E. R. Barnes, "Some Optimal Schemes For ALU Implementation in VLSI Technology", Proceedings of the 7th Symposium on Computer Arithmetic ARITH-7, pp. 2-8, Reprinted in "Computer Arithmetic", E. E. Swartzlander, (editor), Vol. II, pp. 137-142, 1985.

[28] Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v 2, n 1, 124-8, March 1994

[29] H Suzuki, Y Nakase et all, "Leading-zero anticipatory logic for high-speed floating point addition" in Proceedings of the Custom Integrated Circuits Conference, p 589-592, 1995

[30] Mei, Xiao-Lu1, " Leading zero anticipation for latency improvement in floating-point fused multiply-add units" in ASICON 2005: 2005 6th International Conference on ASIC, Proceedings, v 1, p 128-131, 2005

[31] G Dimitrakopoulos, K Galanopoulos, "Low-power leading-zero counting and anticipation logic for high-speed floating point units" in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v 16, n 7, p 837-850, July 2008

[32]Pravin Chander Chandran, Design of ALU and cache memory for an 8 bit microprocessor, MS Thesis, Dept. of Electrical and Computer Engineering, Clemson University, Dec 2008.

[33]David Monniaux,The Pitfalls of Verifying Floating-Point Computations, ACMTransactions on Programming Languages and Systems,Vol. 30,No. 3, Article 12, Publication date:May 2008

[34] Alexander Miczo, Rodolfo Betancourt and Maddumage Karunaratne, Functional Test Using Behavior Models by Manzer Masud, Compcon Spring '92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers., p 446-451, 1992

[35] P. Montuschi, L. Ciminiera, A. Giustina, "Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient", IEE Proceedings - Computers and Digital Techniques -- November 1994 -- Volume 141, Issue6, p. 317-324

[36] R.E. Goldschmidt, "Applications of Division by Convergence," MS thesis, Dept. of Electrical Eng., Massachusetts Inst. of Technology, Cambridge, Mass., June 1964

BOOK REFERENCES

[1] Numerical Computing IEEE Floating point arithmetic by Michael.L.Overton

[2] Computer Arithmetic by Behrooz Parhami

[3] Digital Integrated circuits - A design perspective