ASIC Implementation of 32 and 64 bit Floating Point ALU using Pipelining

Dave Omkar R.
Student
VIT University
Vellore, TamilNadu

Aarthy M.
Assistant Professor
VIT University
Vellore, TamilNadu

ABSTRACT
The 32-bit and 64-bit Floating point Arithmetic Logic Unit is a main part in the design of computers. The Aim of this paper is high performance through the pipelining concept compared to non-pipelining. This ALU includes all the arithmetic and logical operations. The Pipelined modules are independent of each other. The novelty is to design pipelined modules like left shift, right shift, increment, decrement and logical modules. The Arithmetic pipelined modules are also modified. These modules use single and double precision IEEE 754 standard to carry out the required operation. All modules in the ALU design are realized using Verilog HDL. Test vectors are given to the inputs of the floating point ALU to testify its functionality. The simulation is carried out with ModelSim 6.5b simulator and RTL synthesis is done with RTL Compiler tool in Cadence. Physical design of this architecture is done with SoC Encounter cadence tool in 180nm technology.

General Terms
Algorithm, Floating point number.

Keywords
, ALU, ASIC, IEEE 754, LSB, MSB, Verilog HDL.

1. INTRODUCTION
Floating Point numbers are used when there is necessity numbers to be very large or to be very small [1]. Floating point representation has its advantages of its resolution and accuracy compared to fixed point number representation. Numbers in the floating point are represented in the form of bit string. This bit string is combination of sign bit, mantissa and exponent power. This representation is called IEEE 754 standard [2].The single precision of floating Point is shown in Fig 1[2].

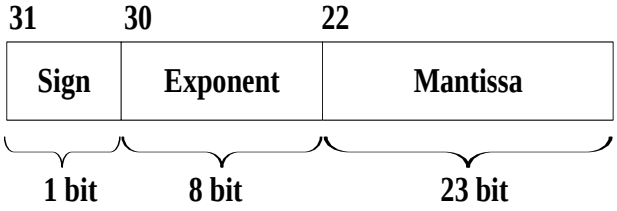

Fig 1: Basic IEEE 754 standard format for single precision

For double precision IEEE 754 standard, the difference in the Fig 1 is Exponent is 11 bit wide and mantissa is 52 bit wide.

The format for the single precision is written below.

$$n = (-1)^s \, 2^{e-127} \, (1.f) \qquad (1)$$

Where 0<e<255 and$f \neq 0$.

For double precision, the difference is in the exponent. It is 1023 instead of 127 and the range of e is 0<e<2047.

ALU is a digital module that performs all the arithmetic and logical operations. It is an important block in CPU. Depending on the selection bits ALU executes the appropriate operation and gives the result. Along with ALU output there are also status bits which represent exception in the arithmetic operations. They are result zero, overflow, and underflow, divide by zero and normal operation. Pipelining is a special technique to give the faster output and reduce the delay in the design. It allows many operations to occur in parallel. Pipelining reduces the critical path in the circuit hence increases the speed.

Generally in Pipelining, each operation of the stage is performed at each clock pulse and concurrently the output of the previous stage is given to the next stage so there is no waste of clock pulse in the pipelining [3].Implementing pipelined architecture of floating point ALU gives faster results. The proposed 32 and 64 bit proposed floating point ALU carry out 16 different arithmetic and logical operations with pipelining [4]. The modified addition, multiplication and division algorithms of the floating point numbers are designed using Verilog HDL. The proposed Left shift, Right shift, Increment, Decrement and all logical modules are also implemented for Single precision and double precision.

In Proposed Pipelined modules, there are maximum 6 stages as shown in the Fig 2 .So, after 6 clock pulse, the first output comes and at 7^{th} clock pulse second output comes. It reduces the number of clock pulses.

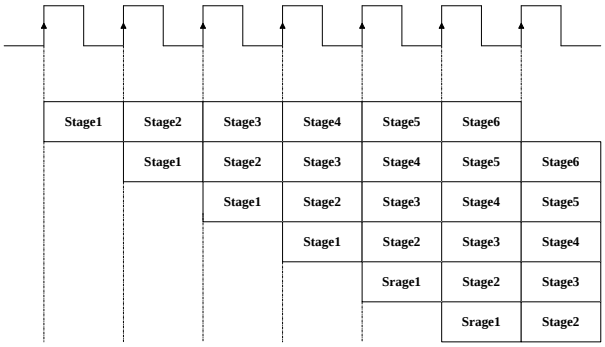

Fig 2: Stages in Pipelining

2. BACKGROUND

The main target of the previous work was to implement 16 bit floating point ALU using pipelined modules in VHDL[1].It can be viewed in Fig 3. The sub-objectives were to design pipelined addition and sub traction. The operations are limited to only four arithmetic operations like addition, subtraction, multiplication and division [4].

The addition, subtraction, multiplication and division were done by using arithmetic operator. The previous work has been done for 16 and 32 bit floating point ALU [4]. The maximum number of stages up to pipelining was up to 4.

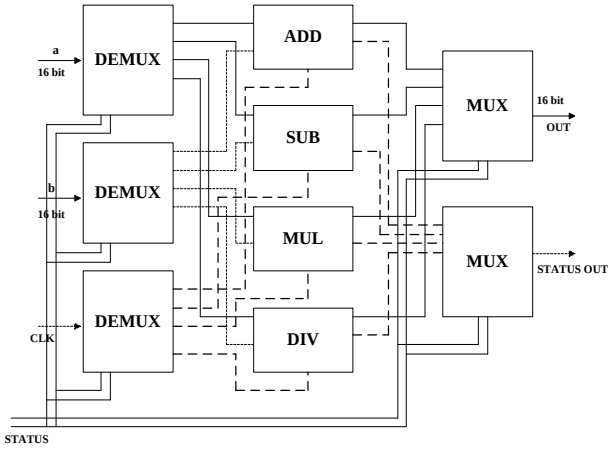

Fig 3: Top level view of the ALU design

3. DESIGN AND METHODOLOGY

The new architecture of 32 bit and 64 bit floating point ALU with pipelined modules has been implemented which contains all the arithmetic as well as logical operations These modules have 4 or more than 4 pipelined stages.

3.1 Modified Top Level architecture of 32-bit ALU

As shown in Fig 4, the modified top level architecture of 32-bit floating point ALU consists of 3 levels. In, first level there are 3 demultiplexers .first demultiplexer is for selecting the first operand and second demultiplexer is for selecting the second operand and last demultiplexer is to select the clock. In second level, there are 16 blocks consists of all the arithmetic and logical operations depending on the selection bits as shown in the TABLE 1. These blocks have two outputs ALU out and status. Third level consists of 2 multiplexer .First multiplexer is used to select the ALU operation and second multiplexer is used to select status bits. These status bits are shown in TABLE 2.

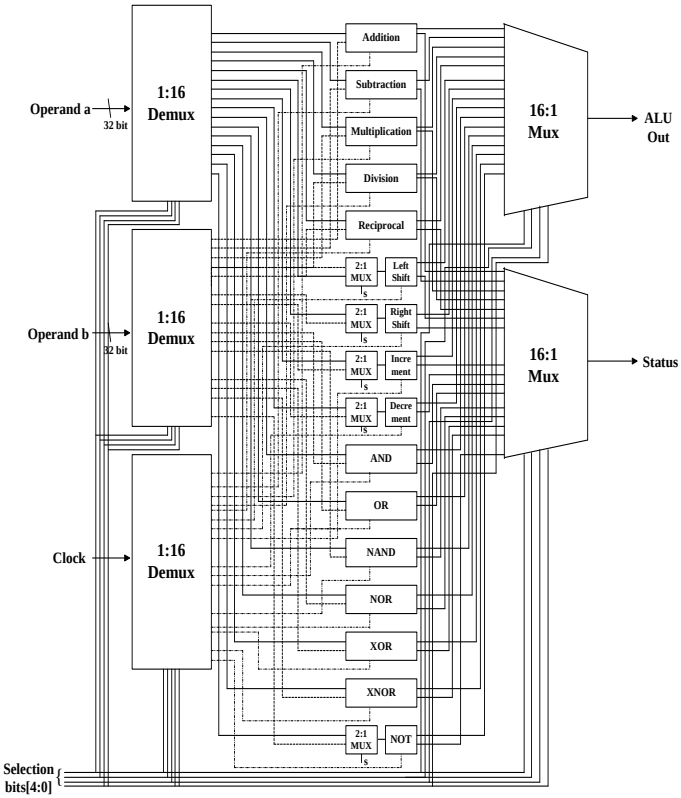

Fig 4: Modified top level view of 32 bit Floating Point ALU

Table 1. Selection of ALU operation

No.	Selection bits[3:0]	ALU Operation
1	0000	Addition
2	0001	Subtraction
3	0010	Multiplication
4	0011	Division
5	0100	Reciprocal
6	0101	Left Shift
7	0110	Right Shift
8	0111	Increment
9	1000	Decrement
10	1001	AND
11	1010	OR
12	1011	NAND
13	1100	NOR
14	1101	XOR
15	1110	XNOR
16	1111	NOT

Table 2. Status bits and status

No.	Status bits[2:0]	Status
1	000	Result zero
2	001	Overflow
3	010	Underflow
4	011	Normal Operation
5	100	Divide by Zero

3.2 Modified Top level architecture of 64-bit ALU

The difference between 32 and 64 bit floating point ALU is in giving the size of the operands. For 64 bit floating point ALU, the operands A and B are 64 bits wide, because This ALU uses double precision IEEE 754 standard format. So, the top view of 64 bit floating point is constructed as in the Fig 4. But the inputs and output is 64 bit instead of 32 bit.

3.3 Modified 32-bit and 64-bit Pipelined floating point Addition/Subtraction module

The algorithm and architecture for the 32 and 64 bit pipelined floating point addition/subtraction has been designed.

3.3.1 Modified Addition/Subtraction Algorithm

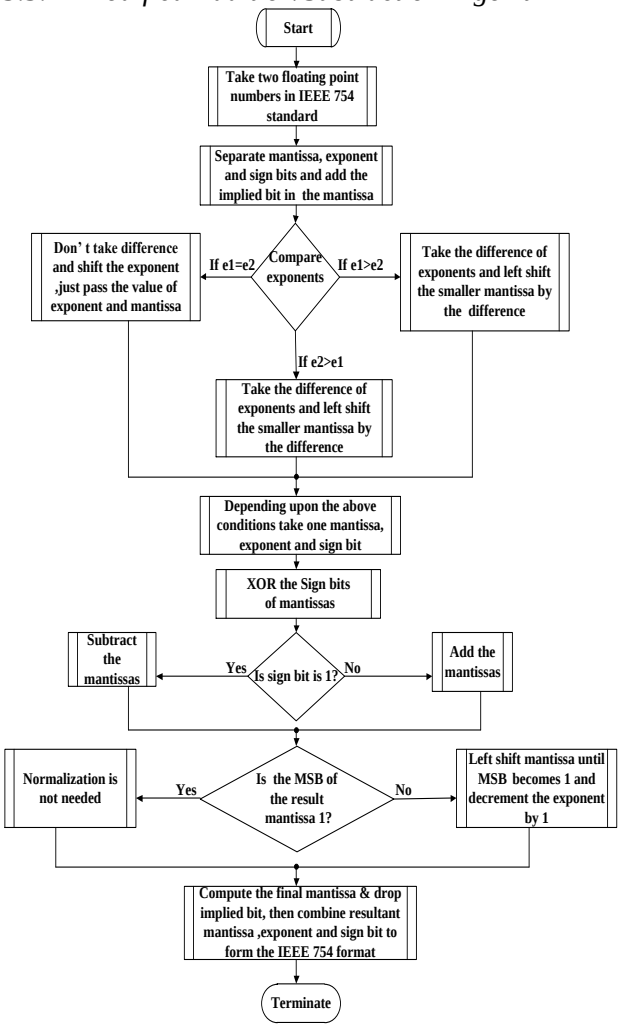

Fig 5: Flowchart for modified addition/subtraction algorithm

For 64 bit add/sub, The procedure is same but the mantissa and exponent bits are 52 bits and 11 bits wide. For 32 and 64 bit subtraction, the change is in the operation of the mantissa i.e. addition becomes subtraction and vice versa.

3.3.2 Modified Pipelined Addition/Subtraction architecture

For this addition/subtraction algorithm, new 32 bit pipelined addition/subtraction module has been implemented. D-FF is used for the pipelining. It is shown in Fig 6.

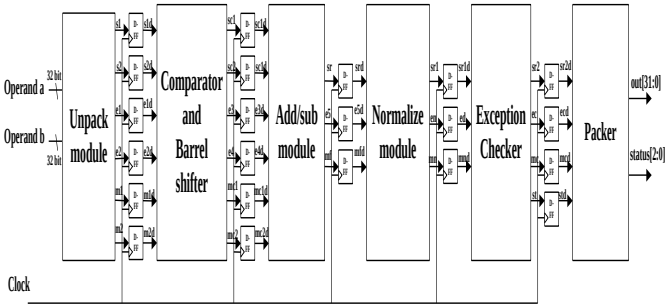

Fig 6: 32- bit pipelined add/sub architecture

The working of the above architecture is explained below.

3.3.2.1 Unpack module

This module will separate mantissa, exponent and sign bit from floating point numbers. It will also add the implied bit to the mantissa.

3.3.2.2 Comparator and Barrel Shifter

This module will compare the exponents of the operands and shift the smaller exponent by the difference of their exponents.

3.3.2.3 Add/sub module

This module will add or subtract depending upon their signs. This sign is determined by doing XOR of the two sign bits of the operands. Here 24x24 Ripple carry adder for single precision and 53x53 Ripple carry adder are used for addition because of its simplicity. For sub traction, the Ripple borrow sub tractor is used .It uses full sub tractor instead of full adder.

3.3.2.4 Normalize module

In this module, the normalizing of the final result is carried out. If MSB of the addition result is 0, the mantissa is left shifted until the MSB becomes 1 and Exponent should be decremented.

3.3.2.5 Exception Checker

In this module, exceptions are checked like overflow, underflow, result zero and normal operation after checking mantissa and exponent. If exponent is 255 then "overflow" exception will be raised. If exponent is 1, then "underflow" exception will be raised. If both operands are zero, "Result zero" exception will be raised. Otherwise "Normal Operation" will be raised.

3.3.2.6 Packer

Packer module will combine the resultant sign bit, exponent and mantissa. It will drop the implied bit from the resultant mantissa.

The modified 64-bit pipelined floating point add/sub module is implemented by changing the mantissa and exponent bits in the operands.

3.4 Modified 32-bit and 64-bit Pipelined floating point Multiplication module

The algorithm and architecture for the 32 and 64 bit pipelined floating point multiplication has been implemented.

3.4.1 Modified Multiplication Algorithm

It is shown in Fig 7.

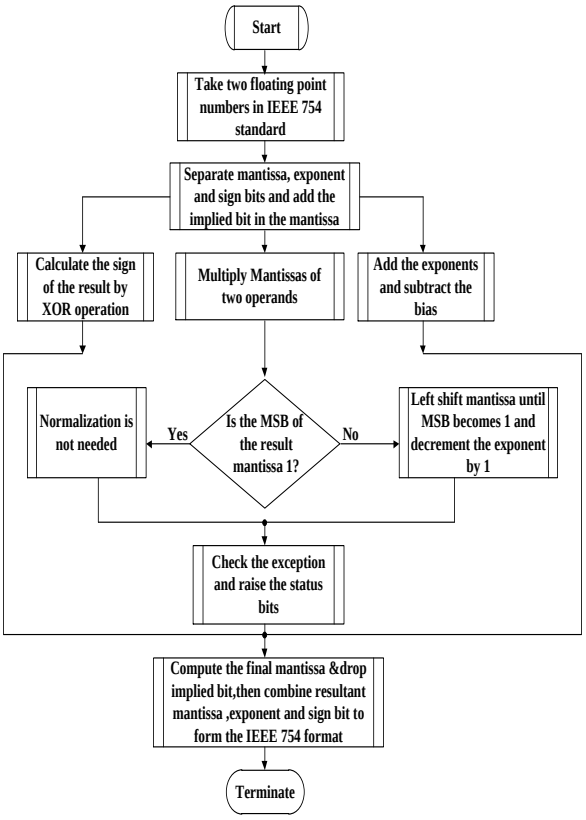

Fig 7: Flowchart for modified multiplication algorithm

For 64 bit multiplication, the procedure is same but the mantissa and exponent bits are 52 bits and 11 bits wide instead of 23 bit in mantissa and 8 bit in exponent in 32 bit multiplication.

3.4.2 Modified Pipelined Multiplication architecture

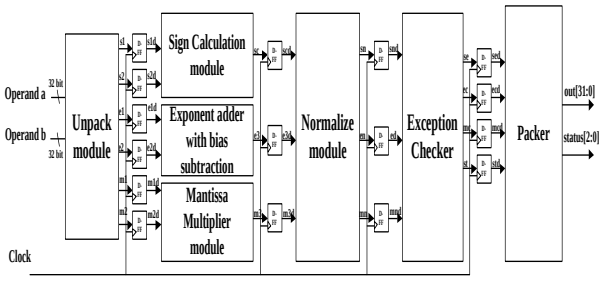

Fig 8: 32-bit pipelined multiplication architecture

The working of the Fig. 8 is explained below.

3.4.2.1 Sign Calculation module

This module calculates the output sign of the resultant mantissa by doing the XOR operation of the two sign bits of the operands .If the resultant sign is 0 ,then the result is positive and vice versa.

3.4.2.2 Exception adder with bias subtraction

It computes the result exponent by adding the exponents of two operands with bias subtraction of $(01111111)_b$

3.4.2.3 Mantissa Multiplier module

This module will calculate the multiplication of the two mantissas. Here the multiplication should be done for 24x24 in single precision and 53x53 in double precision .But to reduce the area 12x12 multiplication has been done in the implementation of mantissa multiplier module .For the multiplication carry save multiplier has been used because of its less use number of half adder and full adder.

The 64 bit pipelined floating point multiplication module is implemented by changing the mantissa and exponent bits in the operands.

3.5 Modified 32-bit and 64-bit Pipelined floating point Division module

3.5.1 Modified Division Algorithm

It is shown in Fig 9.

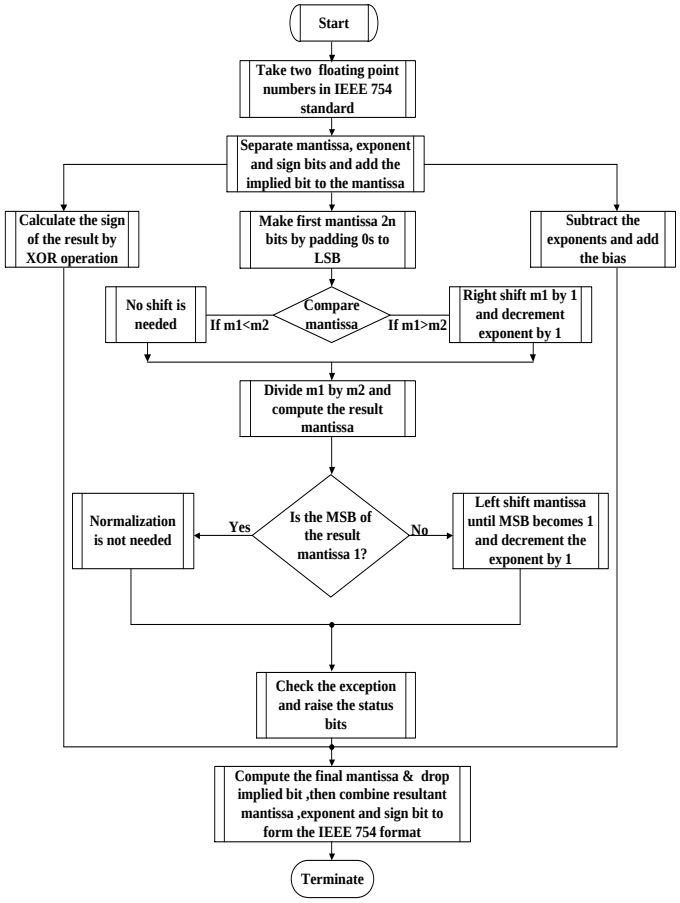

Fig 9: Flowchart for modified division algorithm

3.5.2 Modified Pipelined Division architecture

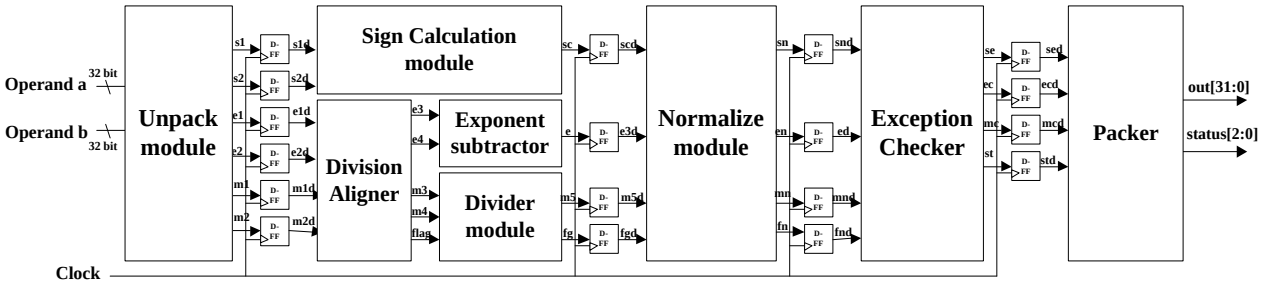

Fig 10: 32 bit pipelined division architecture

The working of the Fig.10 is explained below.

3.5.2.1 Divide Aligner

This module will align the mantissa to get the desired result. As mentioned in the division algorithm, if first mantissa is greater than second mantissa, Division Aligner will shift the first mantissa by 1 and decrement the exponent by 1.It also indicates the division flag for the exception like divide by zero, result zero and normal operation.

3.5.2.2 Divide Module

This module will divide the mantissa by the restoring method. It is explained below.

-First shift left the dividend by 1.

-Subtract the divisor. If the carry is 1 do not restore. If carry is 0 i.e. answer is negative then restore by adding back to the divisor.

-Place the carry as the LSB of the intermediate answer.

-Do this procedure up to n –iterations, where n is number of bits in the divisor. Here n is 24 bits for single precision and 53 bit for double precision.

3.5.2.3 Exponent Sub tractor

It subtracts the exponents of two operands and adds the bias of 127 in single precision and 1023 in double precision.

3.6 The Proposed 32 and 64-bitFloating Point Reciprocal Architecture

For the reciprocal architecture, the algorithm is same as division algorithm. The only difference is that the first mantissa is always 1.The architecture for Reciprocal architecture is shown in Fig 11.

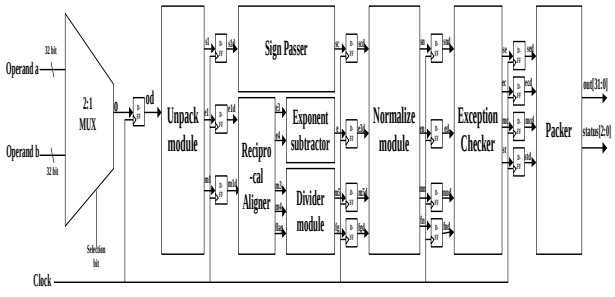

Fig 11: 32- bit proposed pipelined Reciprocal architecture

3.6.1 2:1 Multiplexer

This multiplexer will select one operand out of two operands which is to be reciprocal.

3.6.2 Sign passer

This module just passes the sign of the operand to the next module.

3.6.3 Reciprocal Aligner

It will align the mantissa of the operand. In detail, it compares the mantissa of 1 to the mantissa of the given operand and aligns that mantissa according to the division algorithm which is given in the Section 3.3.2.

3.6.4 Exponent Sub tractor

It subtracts the exponent of the operand with the exponent of 1 and adds the bias of 127 in single precision and 1023 in double precision.

3.6.5 Divide Module

Divider module will divide the mantissa of 1 to the mantissa of the operand. These mantissas are taken from the previous module Reciprocal aligner which aligns the mantissa according to the conditions mentioned in the division algorithm.

3.7 The Proposed 32 and 64-bitFloating Point Left shift and Right Shift architecture

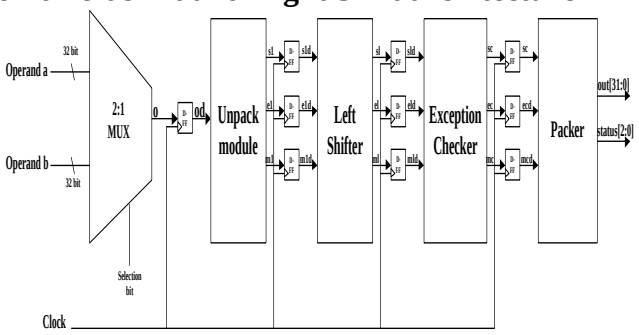

Fig 12: 32-bit proposed pipelined Left Shift architecture

As Shown in Fig 12, the unpack module, Exception checker and Packer module are same as described in the section 3.3.2. The new block is left shifter. It is explained below.

3.7.1 Left Shifter

This module will shift the mantissa part of the floating point i.e. the exponent will be incremented by 1.

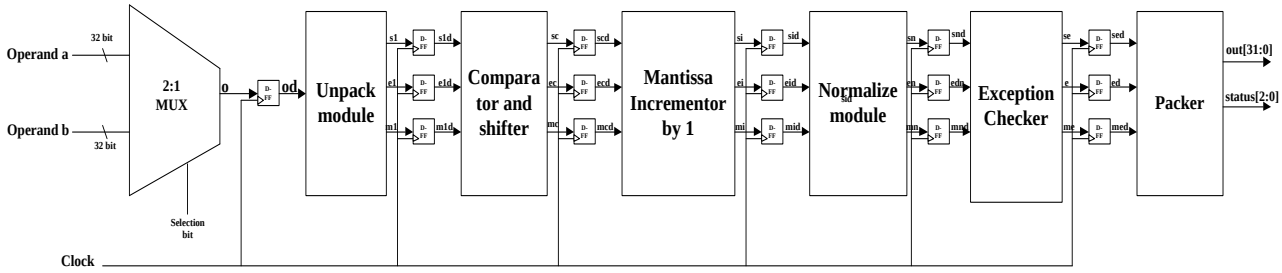

Fig 13: 32- bit proposed pipelined Increment architecture

For Right shift architecture, the difference is in the Right shifter instead of Left shifter in the Fig. 12.In Right shifter, the exponent will be decremented by 1 to get the operand right shifted.

3.8 The Proposed 32 and 64-bitFloating Point Increment and Decrement architecture

The working of Fig.13 is explained below.

3.8.1 Comparator and shifter

This module will compare the exponent of the operand to the 011111111 in single precision and 01111111111 in double precision, because to increment the number by 1, add the mantissa to 1. So, the single precision IEEE 754 standard format of 1 is 3F800000 and double precision is 3FF0000000000000.Comparator will compare this exponent and shifter will shift the mantissa of 1 by the difference of the exponent of the operand and exponent of 1.

3.8.2 Mantissa Increment by 1

This module will add or subtract the mantissa with the mantissa of 1 depending upon the sign of the first operand.

For Decrement architecture, the difference is in the Mantissa decrement by 1 instead of Mantissa increment by 1 in the Fig 13.Decrement architecture, the mantissa will be subtracted with 1 instead of addition.

3.9 The proposed 32-bit and 64-bit Pipelined Logical modules

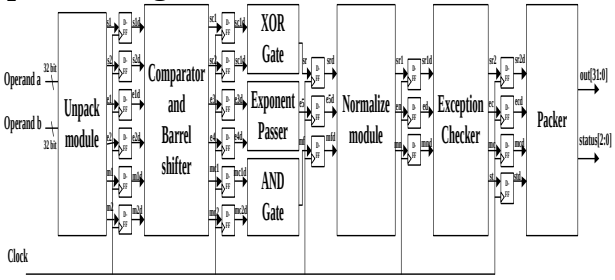

Fig 14: 32-bit proposed pipelined Logical AND module

3.9.1 Comparator and shifter

The resultant sign is calculated by the XOR operation of the sign bits of the two operands.

3.9.2 Exponent Passer

This passer will pass the value of the output exponent to the next module

3.9.3 AND Gate

This gate is used to perform the Logical AND operation of the mantissas of two operands.

The Logical modules like OR, NOR, NOT, XOR & XNOR are implemented by changing the gate in the Fig 14 instead of AND gate. The new 64 bit pipelined floating point Logical modules for the above operations are implemented by changing the operand bit size 64 instead of 32bit.

4. RESULTS AND DISCUSSION

The Simulations has been done in ModelSim 6.5 by giving the different test vectors to the 32 and 64 bit Floating point ALU with pipelined modules. The Simulation results are shown by merging the two operations of the ALU. The Synthesis results in 180 nm of both ALU are shown in the TABLE 3.

For 32-bit and 64-bit operations of ALU, the inputs and outputs are in the form of IEEE 754 standard. For example, the addition & subtraction are performed as under.

Operand 1= $(21.43)_d = (41ab70a4)_h = (40356e147ae147ae)_h$.
Operand 2 = $(7.23)_d = (40e75c29)_h = (401ceb851eb851ec)_h$.
Output = $(28.67)_d = (41E55C29)_h = (403cae147ae147ae)_h$.
For Subtraction,
Operand 1= $(15.25)_d = (41740000)_h = (402e800000000000)_h$.
Operand 2 = $(-5.5)_d = (c0b00000)_h = (C016000000000000)_h$.
Output = $(20.75)_d = (41a60000)_h = (4034c00000000000)_h$.

4.1 Simulation Results for 32-bit and 64-bit Floating Point ALU

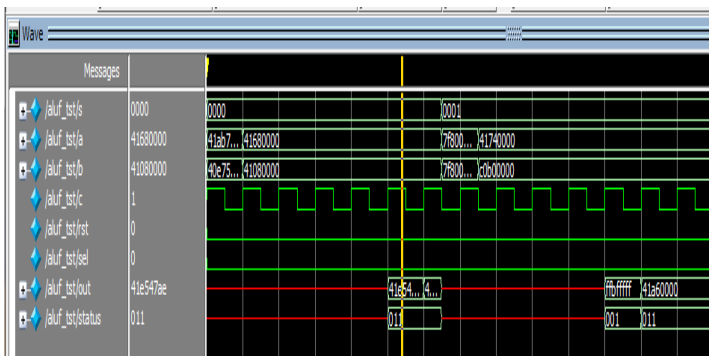

Fig 15: output of 32-bit Floating Point Addition and Subtraction

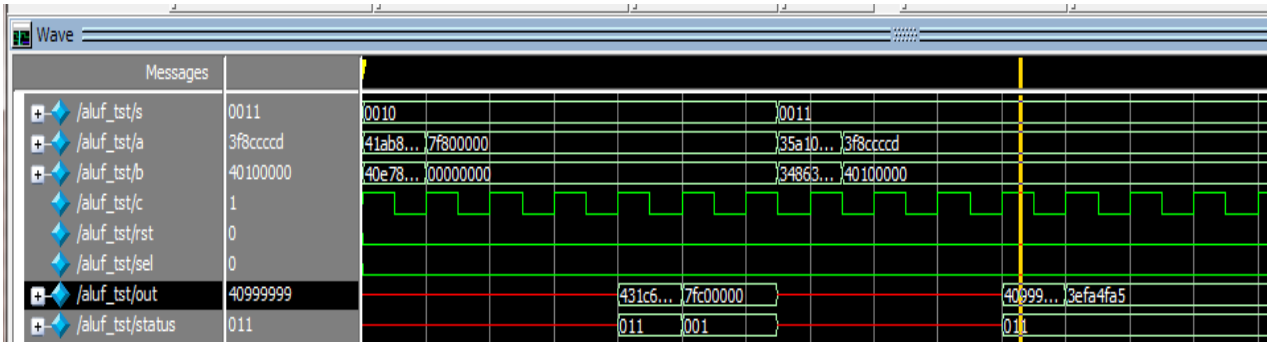

Fig 16: output of 32-bit Floating Point Multiplication and division

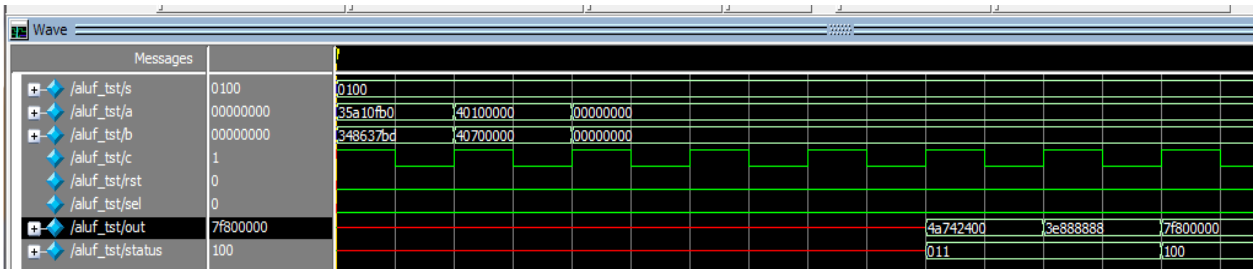

Fig 17: output of 32-bit Floating Point Reciprocal

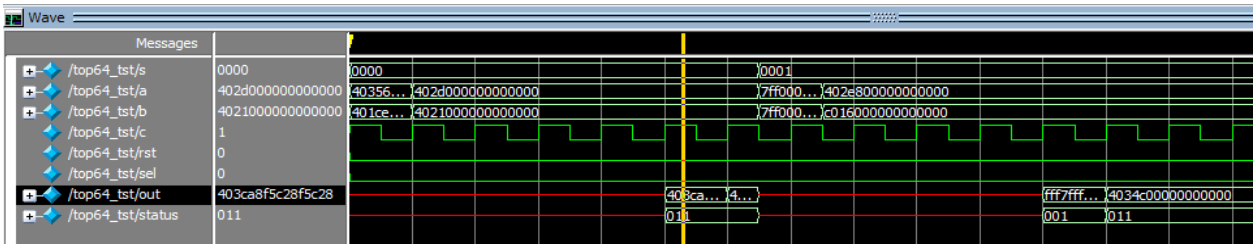

Fig 18: output of 64-bit Floating Point Addition and Subtraction

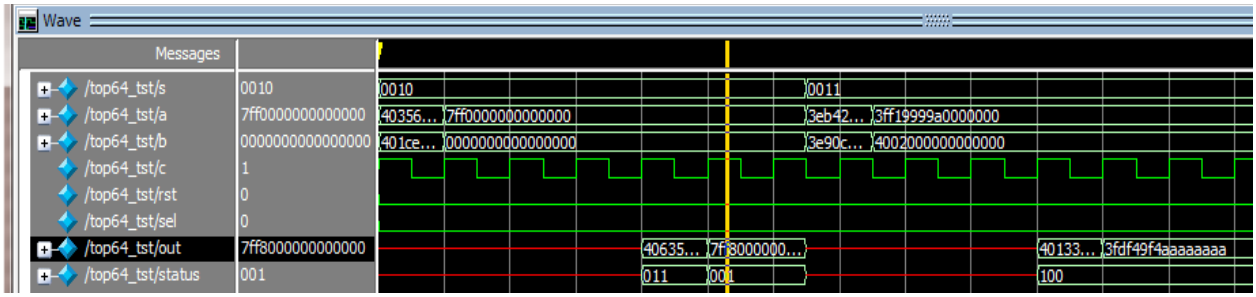

Fig 19: output of 64-bit Floating Point Multiplication and Division

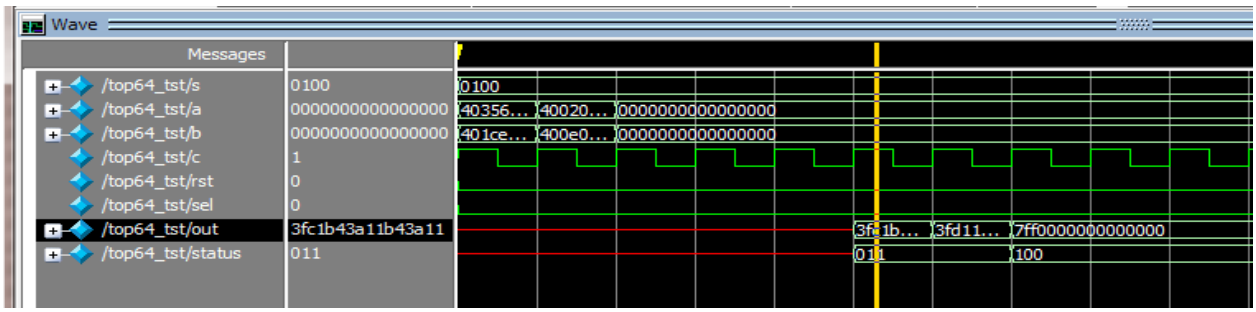

Fig 20: output of 64-bit Floating Point Reciprocal

4.2 Synthesis results

4.2.1 Report summary of area, power and delay

Table 3 Parameters for 32 bit Floating Point ALU with and without Pipelining

Parameters	32-bit Floating point ALU with Pipelining	32-bit Floating point ALU without Pipelining
Cells	41092	37621
Cell Area(mm^2)	0.936860	0.820796
Power(nW)	213482898.96	15799143.81
Worst path delay (ns)	0.891	16.231

Table 4 Parameters for 64 bit Floating Point ALU with and without Pipelining

Parameters	64-bit Floating point ALU with Pipelining	64-bit Floating point ALU without Pipelining
Cells	136564	103096
Cell Area(mm^2)	3.1175726	2.763565
Power(nW)	971665797.7	773982042.61
Worst path delay (ns)	1.018	34.323

From the Table 3 and Table 4, it can be summarized that the delay is less for 32 and 64 bit Floating Point ALU with Pipelining compared to without pipelining. The frequencies of Operation of 32-bit Floating Point ALU and 64-bit Floating Point ALU with Pipelining are 1.122GHz and 0.9823GHz respectively.

4.2.2 RTL Schematic

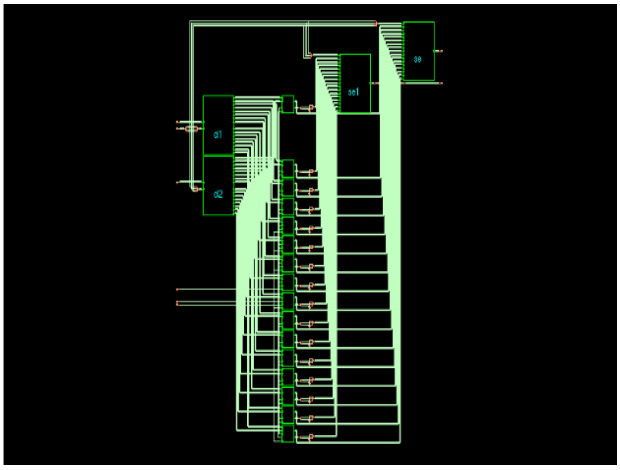

Fig. 21: RTL Schematic of 32-bit Floating Point ALU with Pipelining

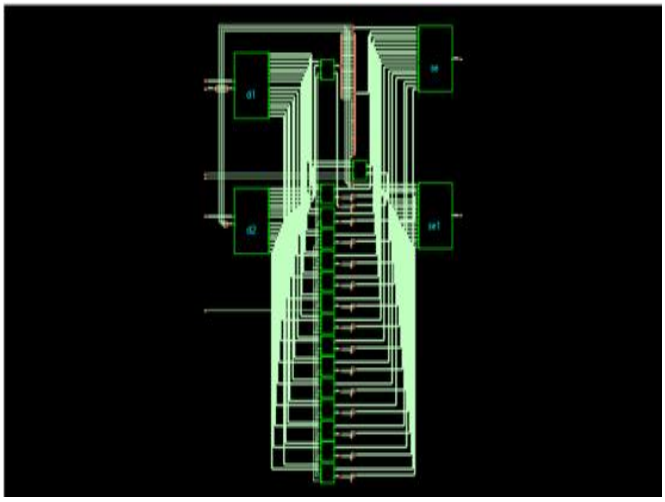

Fig. 22: RTL Schematic of 64-bit Floating Point ALU with Pipelining

4.3 Backend Results

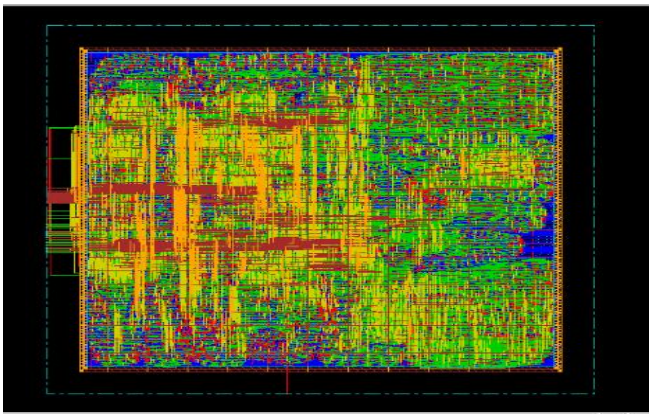

Fig 23: Chip Layout of 32-bit floating point ALU with pipelining

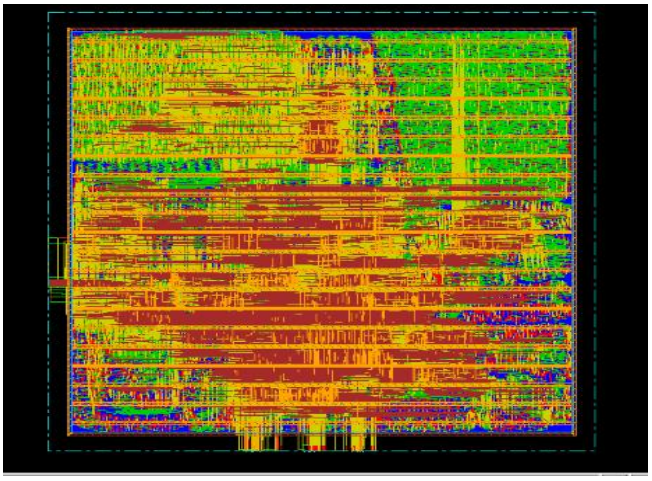

Fig 24: Chip Layout of 64-bit floating point ALU with pipelining

5. ACKNOWLEDGEMENTS

It is pleasure to thank MR. JAYKRISHANAN P .Asst. professor from the department of SENSE (School of Electronics engineering), VIT University for helping in the project work. His guidance, encouragement and suggestions are helpful from the starting to the end of this project**.**

6. CONCLUSION AND FUTURE WORK

In this paper, The 32-bit and 64- bit floating point ALU using Pipelining are implemented successfully and the comparison of 32 and 64 bit floating point ALU using pipelining has done with 32 and 64 bit floating point ALU without using pipelining with respect to area, power and delay. The simulation with different test vectors is done in Modelsim 6.5b. The Rounding logic for the floating point numbers after doing the required arithmetic and logical operations can be implemented for 32 and 64 bit floating point ALU. The Power got after the synthesizing can be lowered by different low power techniques. For the complete analysis of the ASIC design, one can do the post-layout simulation (Formal verification) that was left in this paper.

7. REFERENCES

[1] Rajit Ram Singh, Asish Tiwari, Vinay Kumar Singh, Geetam S Tomar," VHDL environment for floating point Arithmetic Logic Unit -ALU design and simulation", 2011 International Conference on Communication Systems and Network Technologies.

[2] Kai Hwang Book,"Advanced Computer Architecture".

[3] ANSI WEE STD 754-1985, "IEEE Standard for Binary Floating-Point Arithmetic", IEEE, New York, 1985.

[4] Mamu Bin Ibne Reaz, MEEE, Md. Shabiul Islam, MEEE, Mohd. **S.** Sulaiman, MEEE," Pipeline Floating Point ALU Design using VHDL" ICSE2002 Proc. 2002 , Penang, Malaysia.

[5] Shao Jie, Ye Ning, Zhang Xiao-Yan," An IEEE compliant Floating-point Adder with the Deeply Pipelining paradigm on FPGAs", 2008 International Conference on Computer Science and Software Engineering.

[6] Prashant Gurjar, Rashmi Sola Pooja Kansliwal, Mahendra Vucha, "VLSI Implementation of Adders for High Speed ALU.

[7] A. Anand Kumar Book," Fundamentals of Digital Circuits".

[8] V.Narasimha rao, V.Swathi," Normalization on floating point multiplication using Verilog HDL", International Journal of VLSI and Embedded Systems-IJVES, ISSN: 2249 – 6556.

[9] Poornima M, Shivaraj Kumar Patil, Shivukumar , Shridhar K P , Sanjay H," "Implementation of Multiplier using Vedic Algorithm" International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Volume-2, Issue-6, May 2013.

[10] Itagi Mahi P. and S. S. Kerur " Design and Simulation of Floating Point Pipelined ALU Using HDL and IP Core Generator" ISSN 2277– 4106©2013 INPRESSCO.

[11] Sukhmeet Kaur, Suman, Manpreet Singh Manna, Rajeev Agarwal," VHDL Implementation of Non Restoring Division Algorithm Using High Speed Adder/Subtractor" International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 2, Issue 7, July 2013.

[12] Shuchita Pare, Dr. Rita Jain,"32 Bit Floating Point Arithmetic Logic Unit A LU Design and Simulation,"IJETECS, Vol-1, Issue 8, December 2012.

[13] Deepti Shrivastava,Rajesh Nema," Double Precision floating point ALU Implementation using VHDL" ,International Journal of Advanced Electronics &communication systems Approved by CSIR-NISCAIR ISSN NO:2277-7318.

[14] V.Vinay Chamkur, Chetana. R," Design and Implementation of IEEE-754 Addition and Subtraction for Floating Point Arithmetic Logic Unit ", in Proceedings of International Conference on Computer Science, Information and Technology, Pune, ISBN-978-93-81693-83-4, 23rd June, 2012.

[15] Surendra Singh Rajpoot, Nidhi Maheshwari, D.S. Yadav, "Design and Implementation of efficient 32-bit floating Point multiplier using verilog", International Journal of Engineering and Computer Science,Vol-2 ,Issue 6,June 2013,Page no.2098-2101.

[16] A book on "Verilog HDL: A Guide to Digital Design and Synthesis" by. Samir Palnitkar , second edition.

[17] A User Manual on "GUI Guide for Encounter® RTL Compiler" by cadence®, Product Version 6.1, June, 2006.

[18] A User Manual on "Using Encounter® RTL Compiler" by cadence®, Product Version 9.1, September 14, 2009.

[19] Website:http://babbage.cs.qc.cuny.edu/IEEE-754.old/32bit.html.

[20] Website:http://www.academic.marist.edu/~jzbv/architecture/MultiplicationDivisionFP.htm.