Energy Efficient High Speed Floating Point Arithmetic Unit

Somya Kumawat, Arpan Shah, Ramesh Bharti

***Abstract*— Energy Efficient High Speed Floating Point Arithmetic Unit is introduced in this paper where power optimization has been done. In this paper, the concept of Floating Point (FLP) operation into a single arithmetic logic unit (ALU) that can perform multiplication, subtraction and addition more rapidly, accurate and less complex has been reviewed. The merit of floating point is that precision is maintained with a wide dynamic range, where fixed point numbers lose precision**

***Index Terms*— ALU, FLP, IEEE-754, IEEE-854**

I. INTRODUCTION

Floating-point arithmetic was taken as a mysterious topic by several people. This was rather surprising because floating-point was omnipresent in computer systems. Almost every language had a floating-point data type; computers from PCs to supercomputers have floating-point accelerators [1]; most of the compilers would be called upon to compile floating-point algorithms from time to time; and practically every operating system must react to floating-point exceptions such as flooded [2]. An arithmetic-logic unit (ALU) was the part of a computer processor (CPU) that carried out arithmetic and logic operations on the operands in computer instruction words. In several processors, the ALU was split into two units, an arithmetic unit (AU) and a logic unit (LU). Some processors contained one or more AU - for example, one for fixed-point operations and another for floating-point operations. Usually, input and output approach to the processor controller, main memory (random access memory or RAM in a PC), and input/output devices had been directly done the ALU [2] - [3]. A bus was an inputs and outputs flow along an electronic lane. The input consisted of an instruction word (occasionally called a machine instruction word) that contained an operation code (occasionally called an "op code"), one or more operands, and occasionally a format code. What operation to perform and the operands were used in the operation was told by the operation code to the ALU (For example, two operands might be added together or compared logically) [4].The format might be combined with the op code and it was told, for example, whether this was a floating-point or a fixed-point instruction. The output consisted of a result that was placed in a storage register and settings that indicated whether the operation was performed successfully [5]. (If it wasn't, some sort of status would be stored in a everlasting place that was sometimes called the machine status word). [6]

Usually, the ALU included storage places for input operands, operands that were being added, result stored in an accumulator, and shifted results [7]. Gated circuits controlled the flow of bits and the operations performed on them in the sub units of the ALU. The gates in these circuits were controlled by a series logic unit that used a particular algorithm or series for each operation code. Multiplication and division were done by a series of adding or subtracting and operations which done shifting in the arithmetic unit [8]. There were several ways to represent numbers which are negative. In the logic unit, one out of 16 possible logic operations could be performed - such as comparing two operands and identifying where bits do not match. A floating-point group could be used to represent with a fixed number of digits, numbers of dissimilar orders of magnitude: e.g. the distance between galaxies or the diameter of an atomic nucleus could be expressed with the same unit of length [9] - [10]. The result of this dynamic range was that the numbers that could be represented were not uniformly spaced; the diversity between two consecutive numbers grows with the selected scale.

Floating-point numbers were well defined by IEEE-754 (32 and 64 bit) and IEEE-854 (variable width) specifications. Floating point had been used in processors and IP for years and was a well-understood system. This was a sign magnitude system, where the sign was processed a different way from the magnitude [10]. There were many concepts in floating point that made it unusual from our common signed and unsigned number notations. These came from the definition of a floating-point number [11].

All the floating point numbers were made up of following components:

- *Sign:* it indicated the sign of the number (0 positive and 1 negative)=1 bit
- *Mantissa*: it place the value of the number=23 bits
- *Exponent*: it contained the value of the base power (biased)=8 bits
- *Base*: the base (or radix) was suggested and it was common to all the numbers (2 for binary numbers

We had used the following packages [11] - [12]:

Denormalize – Boolean: This was used to turn on and off denormal numbers. The default was true (permit denormal numbers)

Round_style – round_type: This was used to denote the rounding style to be used. The default was “Round_nearest” and the most hardware was taken by it. A truncation, round_inf and round_neginf round up or down were done by Round_zero depending on whether the number was positive or negative.

Somya Kumawat, student M.Tech (VLSI), Jagan Nath University, Jaipur, India.
Arpan Shah, Assistant Professor Department of ECE, Jagan Nath University, Jaipur, India.
Ramesh Bharti, Associate Professor Department of ECE, Jagan Nath University, Jaipur, India.

Check_error – Boolean: Infinity processing and turns off NAN. At the commencement of every operation these checks were needed and if we had already checked once we needed not check again.
Guard_bits – natural: This was the number of extra bits used on each operation to preserve accuracy. The default was 3. But rounding was automatically turned off, if we took this down to zero.
No_warning – Boolean: Permit us to turn off the "metavalue" warnings by setting this to "false".
Float_exponent_width: For conversion routines, it was default. Set by default to the size of a 32 bit floating point number (8).
Float_fraction_width: For conversion routines, it was default. Set by default to the size of a 32 bit floating point number (23).

II. ALGORITHMS

A. *Adder*

1. Take two floating points numbers.
2. Separate the numbers into mantissa and exponent.
3. Find the larger exponent. Let say E_x and E_y.
4. If E_x=E_y then only add the mantissa together.
5. If exponents are not same then normalize to higher exponent ,let $E_x > E_y$ then
    a. Find the difference between exponents E_x-E_y
    b. Shift mantissa of smaller number right by difference E_x-E_y
    c. Set exponent of result E_z to bigger exponent E_x
6. Now that exponent are identical add mantissa together, mantissa of result $M_z = M_x + M_y$.
7. Set sign of result according to previous result.

B. *Subtractor*

1. Take two floating points numbers.
2. Separate the numbers into mantissa and exponent.
3. Find the larger exponent. Let say E_x and E_y.
4. If E_x=E_y then only subtract the mantissa.
5. If exponents are not same then normalize to higher exponent, let $E_x > E_y$ then
    a. Find the difference between exponents E_x-E_y
    b. Shift mantissa of smaller number right by difference E_x-E_y
    c. Set exponent of result E_z to bigger exponent E_x
6. Now that exponent are identical, subtract mantissa, mantissa of result $M_z = M_x + M_y$
7. Set sign of result according to previous result.

C. *Multiplier*

1. Take two floating points.
2. Separate the numbers into mantissa (M_x, M_y)and exponent (E_x, E_y).
3. Calculate the sign of the result from the sign of the inputs.
4. Add the exponents $E_x + E_y$.
5. Multiply the mantissa $M_x * M_y$.
6. Store the result by combining the sign, mantissa and exponent.

III. RESULTS

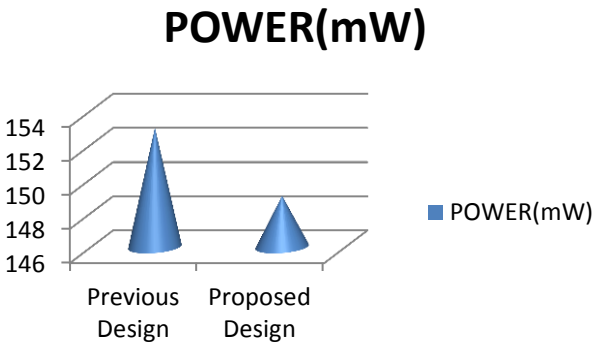


Fig 1 Graphical Comparison of Power

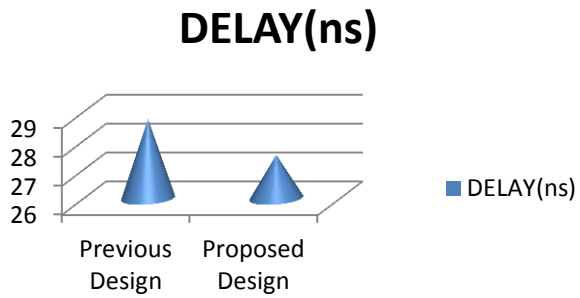


Fig 2 Graphical Comparison of Delay

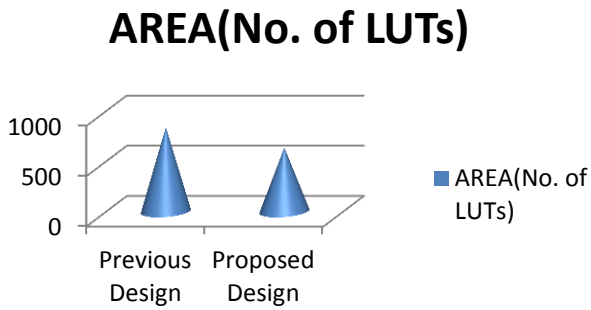
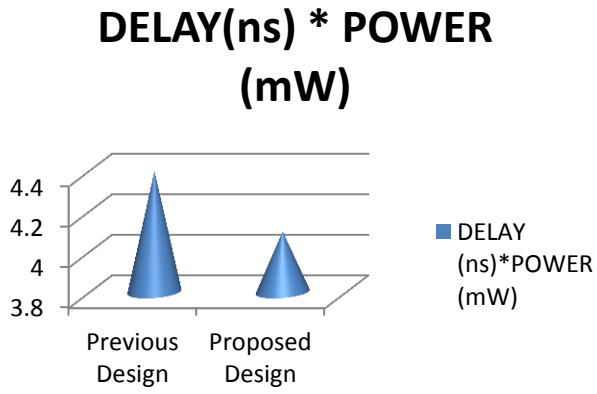


Fig 3 Graphical Comparison of Area

Fig 4 Graphical Compar



ison of Delay * Power

	POWER	DELAY	AREA	DELAY(ns)* POWER

	(mW)	(ns)	(No. of LUTs)	(mW)
PREVIOUS DESIGN	153	28.743	838	4.397
PROPOSED DESIGN	149	27.505	634	4.098

Table 1 Overall Comparison of Previous Design and Proposed Design

IV. CONCLUSION

Table 1 shows the Overall Comparison of Previous Design and Proposed Design. It was clear that there is a decrease in circuitry, power consumption, time consumption as compared to the previous design, also the overall performance (delay (ns) *power (mW))is reduced by 6.8%.

REFERENCES

[1] Jagannath Samanta, Mousam Halder, Bishnu Prasad De" Performance Analysis of High Speed Low Power Carry Look-Ahead Adder Using Different Logic Styles" International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-6, pp. 330–336 2013

[2] Kihwan Jun and Earl E. Swartzlander, "Modified Non-restoring Division Algorithm with Improved Delay Profile and Error Correction" Signals, Systems and Computers (ASILOMAR), pp. 1460-1464, 2012

[3] Nicolas Boullis and Arnaud Tisserand, "On digit-recurrence division algorithms for self-timed circuits", in Research Report published at Institut National De Recherche En Informatique Et En Automatique, France, 2012

[4] Jongwook Sohn and E. E. Swartzlander, Jr. , "Improved Architectures for a Fused Floating-Point Add-Subtract Unit," *IEEE Trans. on Circuits and Systems-I*, vol 59, pp. 2285-2291, Oct. 2012.

[5] Xilinx13.4, Synthesis and Simulation Design Guide", UG626 (v13.4) January 19, 2012.

[6] Manish Kumar Jaiswal, Ray C.C. Cheung, "VLSI Implementation of Double-Precision Floating-Point Multiplier Using Karatsuba Technique", 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum

[7] M.Al-Ashrafy, A.Salem and W.Anis,"An Efficient Implementation of Floating Point Multiplier " Electronics Communications and Photonics Conference(SIECPC) 2011 Saudi International, pp.1-5,2011

[8] Mohamed Al-Ashrfy, Ashraf Salem and Wagdy Anis "An Efficient implementation of Floating Point Multiplier" IEEE Transaction on VLSI 978-1-4577-0069-9/11@2011 IEEE, Mentor Graphics

[9] M. K. Jaiswal and R. C. C. Cheung, "High Performance FPGA Implementation of Double Precision Floating Point Adder/Subtractor", in International Journal of Hybrid Information Technology, vol. 4, no. 4, (2011) October

[10] M. Al-Ashrafy, A. Salem, W. Anis, "An Efficient Implementation of Floating Point Multiplier", Saudi International Electronics, Communications and Photonics Conference (SIECPC), (2011) April 24-26, pp. 1-5.

[11] M. Al-Ashrafy, A. Salem, W. Anis, "An Efficient Implementation of Floating Point Multiplier", Saudi International Electronics, Communications and Photonics Conference (SIECPC), (2011) April 24-26, pp. 1-5.

[12] D. Sangwan and M. K. Yadav, "Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic", in International Journal of Electronics Engineering, (2010), pp. 197-203.

[13] Hani Hassan Mustafa Saleh, *Fused Floating-Point Arithmetic For DSP*, Ph.D. Dissertation, University of Texas at Austin, 2009.

[14] E. Quinnell, E. E. Swartzlander, Jr. , and C. Lemonds, "Bridge FloatingPoint Fused Multiply-Add Design," *IEEE Trans. on VLSI Systems*, vol. 16, pp.l727-1731, 2008

Somya Kumawat, Student of M.Tech in Jagan Nath University, Jaipur. I have completed my M.Tech (VLSI) in 2015 from Jagan Nath University and B.Tech degree in 2013 from Rajasthan Technical University. I am currently working in the VLSI field

Arpan Shah, Assistant Professor Department of ECE in Jagan Nath University, Jaipur, India. He has completed his M.Tech (VLSI) in 2012 and publish various research papers in field of VLSI and Embedded System. He has guided more than 10 M.Tech students for their research work.

Ramesh Bharti, Associate Professor Department of ECE in Jagan Nath University, Jaipur. He is currently pursuing PhD from Jagan Nath University. He has completed his M.Tech (ECE) in 2010 from MNIT Jaipur, and B.E degree in 2004 from Rajasthan University. He is currently working in the wireless communication.