

# Design of Arithmetic and Logic Coprocessor based on Logarithmic Number System

**Abraham N R Singh M.Sc.**  
 Assistant Professor, Department of Electronics  
 St. Johns College,  
 Palayamkottai, Tamilnadu, India  
 abrahamsingh@gmail.com

**T.K. Sethuramalingam M.Tech., (Ph.D)**  
 Associate Professor, Department of ECE  
 Karpagam College of Engineering,  
 Coimbatore, Tamilnadu, India  
 tksethuramalingam@gmail.com

**Abstract**—3D Graphical computing requires more complicated operations such as multiplication, division, square root, power etc., that are computed more faster than conventional method by means of using logarithmic number system (LNS). The proposed implementation is supported in a new set of linear equations, which allows calculating the approximation of the logarithm and antilogarithm binary functions. This design deals with study of logarithmic number system and implementation of an arithmetic and logic unit based on the logarithmic number system.

**Keywords**-Arithmetic logic unit, binary antilogarithm, binary logarithm, hybrid number system, logarithmic number system.

\*\*\*\*\*

## I. INTRODUCTION

With the remarkable progress in the very large scale integration (VLSI) circuit technology, many complex circuits unthinkable yesterday become components easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. This means that not only the [17] conventional computer arithmetic methods, but also the unconventional ones are worth investigation in new designs.

Numbers play an important role in computer systems. Numbers are the basis and object of computer operations. The main task of computers is computing, which deals with numbers all the time [17]. Representing numbers in computer systems is a new issue. The logarithmic number system (LNS) has been studied to simplify arithmetic computations for lower computation complexity [16], high computation speed, and small gate counts however

### LOGARITHMIC ARITHMETIC

OPERATION		NORMAL ARITHMETIC	LOGARITHMIC ARITHMETIC
Multiplication	MUL	$z = x \cdot y$	$X + Y$
Division	DIV	$z = x / y$	$X - Y$
Reciprocal	RCP	$z = 1/x$	$-X$
Square Root	SQRT	$z = \sqrt{x}$	$X \gg 1$
Reciprocal SQRT	RSQ	$z = 1/\sqrt{x}$	$-X \gg 1$
Square	SQR	$z = x^2$	$X \ll 1$

## II. MITCHELL APPROXIMATION

Mitchell introduced the binary logarithmic converting algorithm Mitchell's logarithm and antilogarithm calculations require only shifting and counting operations [16],[5]. The zero detector is to ensure a zero output if any of the inputs is zero. The position of the leading 1 bit in each input is identified by shifting the input bits left until the most significant bit is a "1," decrementing a counter each time, which is initially loaded with the word size. The final values of these counters is known as characteristic, remaining input bits is known as mantiza.

### A. Realizing Mitchell approximation

The above normalization procedure can be formulated as follows. Let

$$b_m b_{m-1} \dots b_0 \cdot b_{-1} \dots b_{-p} \quad (1)$$

be the binary representation of number B where  $b_m$  is the most significant nonzero bit. By factoring out the weight carried by  $b_m$ , the rest becomes in between of 1 and 2. That is,

$$B = 2^m + \sum_{i=-p}^{m-1} 2^i b_i \quad (2)$$

$$= 2^m (1 + x) \quad (3)$$

Where  $i - m < 0$  and  $1 \leq 1 + x < 2$ . Hence,

$$\log_2 B = m + \log_2(1 + x), \quad (4)$$

In the Taylor series of  $\log_2(1 + x)$  taking only the linear term and let  $\log_2(1 + x) \approx x$  then

$$\log_2 B = m + x \quad (5)$$

In binary

$$\log_2 B = a_n \dots a_0 \cdot b_{m-1} \dots b_{-p} \quad (6)$$

where  $a_n \dots a_0$  is the binary representation of  $m$ , and  $b_{m-1} \dots b_{-p}$  is the binary representation of fraction  $x$ , from the lower order part of  $B$  in its binary representation.  $m$  in the above is referred to as *characteristic* and is actually the number of bits between the most significant nonzero bit and the binary point in number  $B$ . One can see later that  $m$  can be easily obtained by simple shifting and counting operations

The error resulted from this method is

$$\varepsilon(x) = \log_2(1 + x) - x. \quad (7)$$

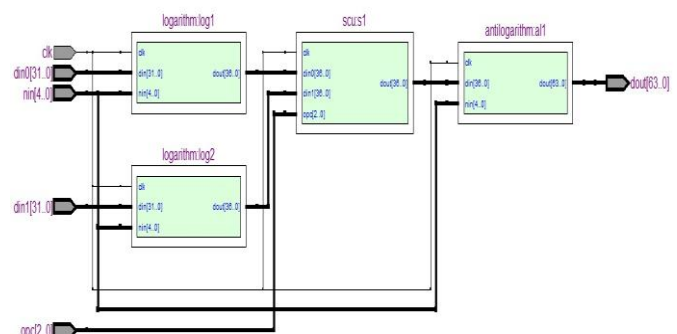


Fig. 1 - Architecture of ALU

Clearly  $\varepsilon(x)$  is independent of  $m$  and depends only on  $x$ . Let

$$\frac{\partial \varepsilon(x)}{\partial x} = 0, \tag{8}$$

$$\frac{\log_2 e}{1+x} - 1 = 0, \tag{9}$$

$$x = \log_2 e - 1 = 0.44 \tag{10}$$

when the maximum error occurs. Thus in the Mitchell approximation the maximum error is 0.086. Then we go instead of using single straight line approximation [1],[3]. to several straight line approximation can be used. In this paper we describe the thirty two region piecewise linear straight line approximation to find logarithm and antilogarithm.

### III. LOGARITHMIC ARITHMETIC LOGIC UNIT

Moreover, the algorithm of Mitchell does not require any storage of coefficients, because this estimate is based on the approximation according to most significant bit position of the argument to be evaluated [2]. The LAU computes the complex functions such as multiplication, division, and square-root operations by using only simple addition, subtraction, and shift operations [5]. The top architecture of the proposed LAU is shown in following figure The unit is composed of two binary logarithmic converters (LOG2s) in the first stage, a simple calculation unit (SCU), and a binary antilogarithmic converter (antilog2) in the second stage. The SCU is composed of an inverter, an adder / subtractor (ADD/SUB), and a barrel shifter (BSH).  $x$  and  $y$  are the operand for the LAU and  $op$  selects the required operation. Since the logarithmic converter takes a longer time than the exponential converter does [11], [12], the SCU is located in the second stage to distribute the time budget to each pipeline stage evenly. While the lookup table-based methods involve memory overhead, Mitchell’s algorithm does not require memory however, it incurs some loss of accuracy. Thus, several researchers have proposed methods to improve the accuracy in Mitchell’s algorithm.

#### A. Logarithmic Converter Block

In general, the piecewise interpolation methods were used in the binary logarithm conversion algorithms In this study, we divide the fraction part into thirty two regions to further reduce its error rate and use the straight linear interpolation in each region. shows the proposed architecture of the logarithmic converter block.  $x$  is an operand to be converted into the logarithmic number, and  $n$  is the number range selection bit. The logarithmic converter block is composed of 32-bit count

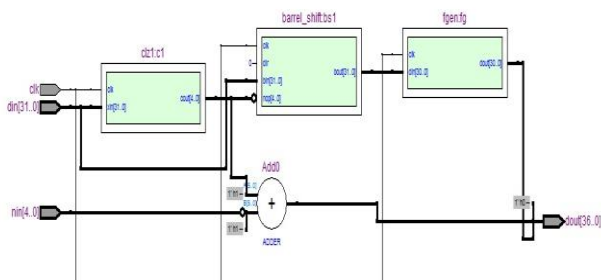


Fig. 1 - Architecture for logarithmic converter

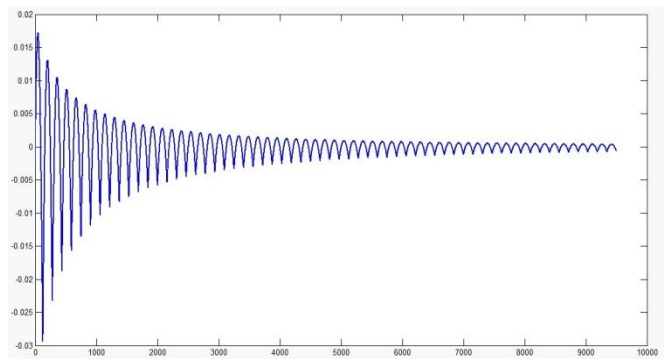


Fig. 3 - Error analysis for logarithmic converter

leading zero (CLZ). Let  $x$  be a fixed-point 32-bit input which has the variable number range of  $Q_m.n$ . Its value can be written as

$$x = 2^k(1 + f), \tag{11}$$

Where  $k$  is the characteristic value of the logarithm in Mitchell’s equation and  $n$  is the number range decision value [14]. Executing the variable number range operation, the characteristic value  $k$  is replaced by the new value of  $k'$ , which is equal to  $k - n$  is the fraction parts in the range  $[0, 1]$  located on the right-hand side of the leading bit. Taking the binary logarithm for both sides of, the following equations[15], can be obtained

$$\log_2 x = k' + n + \log_2(1 + f), \text{ where } k' = k - n \tag{12}$$

$$\log_2(1 + f) \cong a_i \cdot f + b_i, \quad \text{where } i = 0, 1, \dots, 7 \tag{13}$$

where  $\log_2(1 + f)$  is the fractional part after the binary logarithmic conversion. This term is calculated by piecewise interpolation represented in  $a_i$  and  $b_i$ .  $a_i$  and  $b_i$  are the coefficients which have 10 and 16-bit resolution

Architecture of the logarithmic converter block.  $x$  is an operand to be converted into the logarithmic number, and  $n$  is the number range selection bit. The logarithmic converter block is composed of 32-bit count leading zero (CLZ), BSH, a characteristic generator (CGen), and fractional part generation block (FPGen) The CLZ block calculates the number of the leading zero bits of the input. The five bits of the CLZ block output decide the characteristic value and the amount of shift value of BSH. BSH converts the input number range of  $Q_m.n$ . Since the fractional part is divided into thirty two regions, thirty two different coefficients are necessary for thirty two regions. Each coefficient is obtained through fitting to the real value by varying and in order to reduce the complexity because they have a direct effect on the hardware implementation. Since the coefficients[9], are the fraction numbers with powers of two as their denominators, they can be calculated by only shifters and adders. When defining the coefficients, the error range should be considered as well. After shifting operation, the fractional part is generated by the FPGen block and the characteristic part is generated by the CGen block. The above two values are combined to give the logarithmic conversion result.

To calculate the 2’s complement subtraction operation, the inverters and the adders are necessary. In this study, the coefficients consist of only the addition operations so that the internal overhead of the inverters and the adders can be removed. detailed architecture of the proposed FPGen is shown in Fig 4. FPGen generates the approximated fractional value of  $f$ . It is implemented by a hardwired shifter, MUX, carry save

adder (CSA), and carry propagating adder (CPA). The final result is calculated by adding these compensation values to the value of the original fraction part. Although the fractional part has very high resolution, the coefficients and are selected to minimize the number of MUXes and the adders in order to reduce the latency and power consumption. Therefore, the APP carries out the summation of five terms using the CSA tree and CPA [8]. Since this converter receives input values with variable precision of the actual characteristic value is computed by Qm.n. The conversion result is obtained by packing the zero, sign, characteristic value, and the approximated fraction value

After shifting operation, the fractional part is generated by the FPGen block and the characteristic part is generated by the

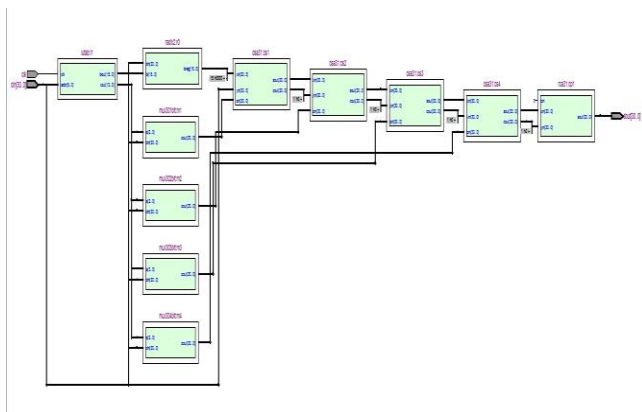


Fig. 4 - Architecture for FPGen

CGen block. The above two values are combined to give the logarithmic conversion result. Since the delay time of FPGen is the longest, it is important to optimize the FPGen to get the high operating frequency.

**B. Antilogarithmic Converter Block**

In general, the piecewise interpolation method is also used for the antilogarithmic converting algorithm. The fixed point representation of the antilogarithm is composed of the six most significant bits representing the integer part  $2^k$  and the remaining bits of the fractional part representing  $2^f$ . Fig. 4 shows the architecture of the antilogarithmic converter block. is an operand to be converted into antilogarithmic number, and is the number range selection bit as it is in the logarithmic converter. The antilogarithmic converter is composed of the integer part generation block, FPGen, and BSH. The computation time is shorter than the logarithmic converter because the integer part and the fractional part are calculated separately. CGen calculates the integer part by using simple addition of , sign bits, and . FPGen is composed of a lookup table (LUT), carry save adders (CSAs), and a CPA similar to the logarithmic converter's. The final BSH shifts the fractional part value by the result value of the CGen block and then makes the output number range from Q6.31 to Qm.n as specified by the number range decision input.

Radix two: assesses whether the input data is a power of two. This block receives as input the approximate fractional Mitchell. Based on this information and with the help of a comparator with evaluates zero mantissa [5], This, to determine whether the input data or not a power of two. If the

64 Region Approximation	logarithm	antilogarithm
Maximum positive error	0.017%	0.0098%
Maximum negative error	0.029%	0.0049%

mantissa of the input data is equal to zero, concludes that the data is a power of two and at therefore not necessary to make any approach regions [7]. This is because when assessing the binary logarithm base on a number that is power of two, the result of this logarithm always is an integer. If, however, the input data is not power two, we proceed to make the approach by region, and that this result is composed of an integer part and a fractional part. The architecture and operation is the same block used to approximate the function

Fractional unit: making the approximation [10], antilog of the value of the fractional part of input value. The architecture and operation is equal to the block used to approximate the function.

TABLE I.  
 LALU OPCODE

Selection (op)	Operation
000	MUL
001	DIV
010	RCP
011	SQRT
100	RSQ
101	SQR

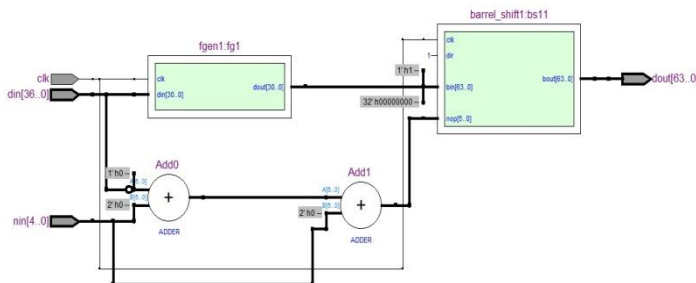


Fig. 5 - Architecture for anti log

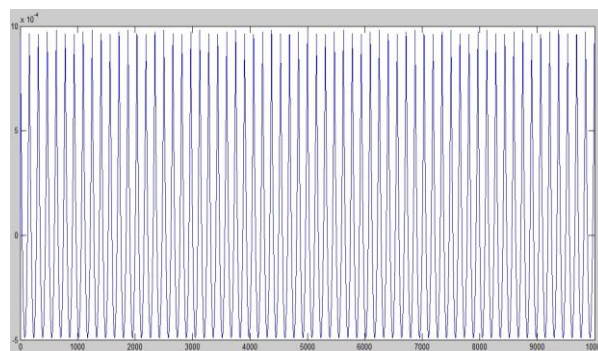


Fig. 6 - Error analysis for anti log converter

$$2^x = 2^{k+f} = \begin{cases} 2^f \cdot 2^k, & x \geq 0 \\ 2^{k-1} \cdot 2^{1-f}, & x < 0 \end{cases} \quad (14)$$

TABLE II.

APPROXIMATION ERROR FOR THIS WORK



Decoder: receiving as input, the value of the entire log and returns a vector output zeros, except for the bit position input value. This value corresponds to the bit more significant value of the integer part of the antilog Binary. A barrel shifter is a combinational logic device/circuit that can shift or rotate a data word by any number of bits of decoder in a single operation.

Concatinator: is responsible for concatenating the result obtained between the integer and fractional part binary logarithm. Taking as reference signal entry, it determines how many bits correspond to the integer and fractional result respectively.

#### IV. SIMULATION RESULTS

Designed ALU takes addition subtraction in normal and other process in logarithmic domain. Logarithmic converter converts inputs into log domain which is *lout0* and *lout1*. TABLE II describe the opcodes (*opc*) for LAU *din0* and *din1* are 32bit input which is given to logarithm converter. Logarithmic converter output is given to simple calculation unit which is capable of calculating the simple addition, subtraction, shift instead for multiplication, division, and square root. Based on given opcode. In logarithmic and antilogarithmic converter has 32 bit barrel shifter that capable for shift any direction either left or right depends on direction (*dir*)input, and number position can be shifted by using another control data port (*nop*)In the logarithmic converter has 32 bit input (*din*) the count leading zero block is used find the position of first MSB non zero bit which is rerepresented as 5 bit (*cout*), the input number range (*nin*) also 5 bit. The number of position in barrel shifter is taken as 5 bit  $cin = 31 - cout$ , the characteristic of logarithmic number is calculated as  $cg = cout - nin$ . The mantisa of the logarithmic number was calculated using floating point generation (FPGen) block. This is 31 bit and also characteristic is 6 bit thus the output of log









 /alu/din0	8388608	8388608	
 /alu/din1	65536	65536	
 /alu/opc	000	000	001
 /alu/nin	0	0	
 /alu/dout	549755813888	549755813888	128
 /alu/lout0	49392123904	49392123904	
 /alu/lout1	34359738368	34359738368	
 /alu/sout	-53687091200	-53687091200	15032385536

Fig. 7 - simulated output for 32 bit ALU

has 37 bit. Antilogarithmic converter has input from simple calculation unit. Antilogarithmic converter is completely reverse process of Logarithmic converter block. The characteristic of input is 6 bit, which is given to the decoder that decodes the characteristic into first leading MSB non zero position of the output Then the barrel shifter is used to concatenate the mantisa by means of left shifting and the number of position shifted (*cg*) is depends on the input number range (*nin*) and characteristic ( $din[36:31]$ ). A 32-bit ALU was designed and simulated based on logarithmic number system using modelsim simulator. Fig. 7 shows the simulation output of LALU for multiplication of two inputs data as *din0*, *din1* and number range *nin* opcode *opc* and also output as *dout*. The Table II represents the maximum positive and negative

approximation error of both logarithm and antilogarithm of this work.

#### V. CONCLUSION

A 32-bit Logarithmic arithmetic unit was designed successfully. A 32-bit Logarithmic arithmetic unit consists of a binary logarithmic converter, an adder, a shifter, and a binary exponential converter. It uses sixty four-region piecewise-linear interpolation approximation algorithms and supports a variable number range to compute complex functions fast and accurately. In future my works towards to modify that design for high speed.

#### REFERENCES

- [1] M.G. Arnold, S. Collange, "A Real/Complex Logarithmic Number System ALU," *IEEE Trans. Comput.*, vol. 60, no. 2, pp. 202-213, Feb. 2011.
- [2] T. Filippov, M. Dorojevets, A. Sahu, A. Kirichenko, C. Ayala, O. Mukhanov, "8-Bit Asynchronous Wave-Pipelined RSFQ Arithmetic-Logic Unit," *IEEE Trans. Applied Superconductivity*, vol. 21, no. 3, pp. 847-851, June 2011.
- [3] P. Kornerup, J.-M. Muller, A. Panhaleux, "Performing Arithmetic Operations on Round-to-Nearest Representations," *IEEE Trans. Comput.*, vol. 60, no. 2, pp. 282-291, Feb. 2011.
- [4] M. Azarmehr, M. Ahmadi, G.A. Jullien, R. Muscedere, "High-speed and low-power reconfigurable architectures of 2-digit two-dimensional logarithmic number system-based recursive multipliers," *IET Circuits Devices Syst.*, Vol. 4, Iss. 5, pp. 374-381, Feb. 2010.
- [5] S. Carrillo, H. Carrillo, F. Viveros, discussed about "Design and Implementation of an Arithmetic Processing Unit Based on the Logarithmic Number System," *IEEE Latin America Trans.*, vol. 8, no. 6, pp. 605-617, Dec 2010.
- [6] H. Fu, O. Mencer, W. Luk, "FPGA Designs with Optimized Logarithmic Arithmetic," *IEEE Trans. Comput.*, vol. 59, no. 7, pp. 1000-1006, July 2010.
- [7] C. Chen, "Error analysis of LNS addition/subtraction with direct-computation implementation," *IET Comput. Digit. Tech.*, vol. 3, Iss. 4, pp. 329-337, Oct. 2008.
- [8] K. Johansson, O. Gustafsson, L. Wanhammar, "Implementation of elementary functions for logarithmic number systems," *IET Comput. Digit. Tech.*, Vol. 2, No. 4, pp. 295-304, Sep. 2007.
- [9] T. Stouraitis, M. Olivieri, S. Smorfa, F. Pappalardo and G. Visalli "Analysis and Implementation of a Novel Leading Zero Anticipation Algorithm for Floating-Point Arithmetic Units," *IEEE Trans. Circuits and Systems-II: Express Briefs*, Vol. 54, No. 8, pp. 685-689, Aug. 2007.
- [10] H. Kim, B. Nam, H. Woo and J.Sohn "A 231MHz, 2.18mW 32-bit Logarithmic Arithmetic unit for Fixed-Point 3D Graphics system" *IEEE Journal of solid State Circuits*, vol. 41, Nov. 2006.
- [11] R. Muscedere, W.C. Miller, V. Dimitrov, G.A. Jullien, "Efficient Techniques for Binary-to-Multidigit Multidimensional Logarithmic Number System Conversion Using Range-Addressable Look-Up Tables," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 257-271, March 2005.
- [12] E.M. Schwarz, M. Schmookler, S. Dao Trong, "FPU Implementations with Denormalized Numbers," *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 825-836, July 2005.
- [13] J.-H. Sohn, Y.-H. Park, C.-W. Yoon, R. Woo, S.-J. Park, and H.-JYoo, "Low-power 3D graphics processors for mobile terminals," *IEEE Commun. Mag.*, vol. 43, no. 12, pp. 90-99, Dec. 2005.
- [14] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low power logarithmic converter," *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1421-1433, Nov. 2003.
- [15] J. Coleman, E. Chester, C. Softley, and J. Kadlec, "Arithmetic on the European Logarithmic Microprocessor," *IEEE Trans. Computers*, vol. 49, no. 7, pp. 702-715, July 2000.

- [16] J. N. Mitchell, "Computer multiplication and division using binarylogarithms," *IRE Trans. Electronic Computers*, vol. 11, pp. 512–517, Aug. 1962.
- [17] Mi Lu, "Arithmetic and Logic in Computer Systems," Wiley-Interscience A John Wiley & Sons, Inc., Publication



**Abraham N R Singh**, This author was born in Tamil Nadu, India, in 1983 and received the UG degree from the Manonmaniam Sundaranar University, India, in 2003. Further he received his PG degree from Annamalai University, India, in 2005. He is working as an Assistant Professor in the Department of Electronics, St. Johns College, Palayamkottai, Tamilnadu. His area of interest in Embedded systems.



**T.K. Sethuramalingam**, This author was born in Tamil Nadu, India, in 1981 and received the UG degree from the Manonmaniam Sundaranar University, India, in 2001. Further he received his PG degree from Bharathidasan University, India, in 2003. He received his M.Tech (Embedded Systems) in PRIST University in 2013. He is working as an Associate Professor in the Department of ECE, Karpagam College of Engineering, Coimbatore, Tamilnadu, India. He visited foreign countries and presented his research publications. He is a Member in IEEE, ISSS, IACSIT, IAENG. His research interests and publications have been in the areas of Embedded systems, VLSI design and MEMS.